



Agency App Development

SAP Mobile Platform 3.0

DOCUMENT ID: DC-01-0300-01

LAST REVISED: November 2013

Copyright © 2013 by SAP AG or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Please see <http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark> for additional trademark information and notices.

Contents

Agentry App Development	1
Setting Up the Development Environment - Agentry Toolkit	1
Installing the Agentry Editor Plug-In and Eclipse Platform	4
Agentry Editor and Eclipse Platform Configuration Overview	11
Agentry OpenUI SDK for iOS Setup Overview	33
Agentry OpenUI SDK for Android Setup Overview	36
Agentry OpenUI for Windows Setup Information	42
Installing the Agentry Test Environment	42
Agentry SAP Framework Foundation Installation Overview	44
Installing the Agentry SDK	52
Developing Agentry Apps	53
Eclipse Preferences for the Agentry Editor Plug-In	53
Agentry Editor and Eclipse Platform Overview	57
Agentry Application Projects: Creating, Managing, and Publishing	75
Overview of Mobile Northwind Sample Application	138
Target Paths and the Property Browser	139
Rules: An Introduction	171
Rule Context	175
Rule Data Types	176
Rule Editor Introduction	179
Syclo Data Markup Language	189
SDML Syntax and Data Tag Expansion	190

Agentry Data Definitions Overview	194
Data Synchronization Overview: The Exchange Data Model	195
Data Synchronization: Data Filtering Overview .	197
Object Development Concepts and Considerations	198
Agentry User Interface Definitions Overview	213
Client User Interface Considerations and Guidance	218
Security Related Development Overview	219
Attached Documents and File Transfer: Key Concepts	226
The Agentry SDK	249
Agentry Language Reference	275
Application Level Definitions Overview	275
Module-Level Data Definitions Overview	312
Module-Level User Interface Definitions Overview	379
Rule Function Terms Overview	458
Syclo Data Markup Language	537
Agentry Test Script Overview	586
Agentry Java API	647
Agentry SAP Framework	811
Agentry SAP Framework	811
Agentry Device Client Branding SDK	989
Agentry Client Installer and Executable Branding	989
Agentry OpenUI API	993
Agentry Client OpenUI API Overview	993
Agentry OpenUI API for Android	996
Agentry OpenUI API for iOS	1133
Agentry OpenUI API for WPF	1200
Index	1259

Agentry App Development

Setting Up the Development Environment - Agentry Toolkit

Included within the SAP® Mobile Platform SDK are components for Agentry development, which make up the Agentry Toolkit. These components are intended for developers, implementors, and administrators.

With the exception of the Agentry SAP Framework Foundation, all the components described here can be found in the SAP Mobile Platform SDK. This SDK is available on the SAP Service Marketplace. After extracting archive that contains all tools available for the SAP Mobile Platform SDK, there are several different items available in various folders. For those components relative to Agentry development, this includes the installers and resources discussed here, as well as the Agentry Client installers.

The Agentry SAP Framework Foundation is an ABAP add-on installed to the SAP system to which the mobile application is to connect and with which it will synchronize data. Since this component is not installed to the same system as the other components of the Agentry development tools, but instead to the SAP system, it is provided as a separate download within the list of SAP Mobile Platform components on the SAP Service Marketplace website.

Each of these components is described briefly below and includes the location of the component with a path relative to the location where the SAP Mobile SDK archive was extracted.

Agentry Editor

File locations:

- 32 bit build: `AgentryToolkit\32-bitAgentryEditor\Agentry_7.0.x.x_EditorPlugin_x86.zip`
- 64 bit build: `AgentryToolkit\64-bitAgentryEditor\Agentry_7.0.x.x_EditorPlugin_x86_64.zip`

The Agentry Editor is the primary development tool for mobile applications built using the Agentry archetype within the SAP Mobile Platform. This is provided as a plug-in to the Eclipse IDE. The proper version of Eclipse must be installed and configured first, followed by the installation of the plug-in to the Eclipse implementation. The Agentry Editor provides a point and click development environment, within which the mobile application can be built or an existing mobile application can be modified for specific implementation needs.

Agentry OpenUI SDK & Android and iOS Agentry Client Rebranding

File locations:

- Agentry OpenUI SDK for Android: AgentryToolkit\AgentryOpenUISDK\SMPAgentryClientFramework-Android-7.0.x.x.zip
- Agentry OpenUI SDK for iOS: AgentryToolkit\AgentryOpenUISDK\SMPAgentryClientFramework-iOS-7.0.x.x.tgz
- Agentry OpenUI SDK for Windows - Sample Project Only: AgentryToolkit\AgentryOpenUISDK\Agentry_7.0.x.x_OpenUISampleDotNET.zip

The Agentry OpenUI SDK is introduced in the SAP Mobile Platform SDK 3.0 release. This SDK is available for Android, iOS, and Windows client platforms. It is provided to allow developers to create custom controls for the Agentry Client. These custom controls will then override the standard field edit types within the Agentry Client at runtime.

For both Android and iOS the client framework archives listed above provide the needed resources to develop custom controls using the OpenUI SDK. For the Agentry Client for Windows devices, this is not necessary. Rather, a project is created in Visual Studio targeting .NET 4.5 Framework, and which uses reflection to obtain information about the loaded assemblies for the OpenUI interface.

Additionally, the resources provided in these same SDK files also provide the necessary interface to support rebranding of the Agentry Client for Android and iOS. Windows clients are rebranded using the Agentry Device Client Branding SDK, as in previous versions of SAP Mobile Platform and/or Agentry Mobile Platform.

Agentry Test Environment

File location:

- Installer: AgentryToolkit\AgentryTestEnvironment\Agentry_7.0.x.x_Test_Environment.exe

The Agentry Test Environment provides an Agentry Client which runs within a test and monitoring interface. This interface provides tools to inspect data stored on the client device, including objects, transactions, complex tables, and data tables. Actions and transactions can be debugged as they are executed or instantiated. Client devices hardware features including barcode scanners and GPS units can be mimicked within the Agentry Test Environment to support full functional testing of client-side behavior of the mobile application. The Agentry Test Environment can also mimic various client device types and form factors.

Agentry Java API

File location:

- Agentry Java API: AgentryToolkit\AJAPI\Agentry-v5.jar
- Agentry Java API Javadoc: AgentryToolkit\AJAPI\Agentry-v5-doc.zip

The Agentry Java API is provided for development of mobile applications with a Java system connection. This API provides the interface between the Agentry Server and the Java

synchronization logic. This interface supports passing data between this logic and the Agentry Server, and also provides state, user, and other runtime information. Base classes for step definition logic, complex table and data table synchronization logic are included within the API and are extended by the logic written specific to the mobile application. The Agentry Java API is contained in a single Jar file, named `Agentry-v5.jar`. Also included is a ZIP archive containing the associated Javadoc content, which can be referenced as a resource within the Eclipse Java project. Details on creating a Java project in Eclipse, including how to add the associated documentation, are provided in a later procedure.

Agentry Branding SDK

File location:

- Windows CE Client Branding SDK: `AgentryToolKit\BrandingSDK\Agentry_ClientWinCE_Branding.exe`
- iOS and Android: *This functionality is encompassed by the OpenUI and the associated Agentry client framework resources provided as a part of the OpenUI SDK. Creating the projects as instructed for each of these will also provide the necessary framework to rebrand these clients.*
- Windows WPF and Windows Desktop: *The Branding SDK is provided from the standard client installers for each these, with special command line switches provided during execution. See the instructions provided for details.*

Rebranding the Agentry Client for Windows devices begins with the installation of the needed rebranding resources. For the Windows desktop Agentry Client builds, this is provided by running the Agentry Client installer with specified command line arguments. This may either be the Agentry Client installer for Windows 7 desktop or earlier, or the .NET Windows Client installer for Windows 8 desktops and tablets.

Rebranding of the Agentry Client for Windows mobile devices begins with the installation of the Agentry Branding SDK, which provides the necessary resources to rebrand and then package the Agentry Client executable for you application.

Rebranding the Agentry Client for both Android and iOS devices is accomplished within the framework provided by the OpenUI SDK. Icon images and UI text can be overridden within the framework and the Agentry Client rebuilt to include these branded resources using the framework provided as a part of the OpenUI SDK for these two platforms.

Agentry SAP Framework Foundation and Related Resources

Note: This component is not provided in the SAP Mobile Platform SDK. It is an ABAP Add-On and is installed only to SAP Systems and is available on the SAP Service Marketplace.

The Agentry SAP Framework Foundation is an ABAP Add-On that is installed to the SAP system with which the mobile application is to connect and synchronize. This component is only used for SAP systems; other back end systems will not use this component. The Agentry SAP Framework Foundation provides the framework within which the needed synchronization support for the mobile application is implemented. This includes exchange

data model components such as triggers, mobile data objects, rules, filters, and other similar pieces.

This component of the SAP Mobile Platform is provided on SAP Service Marketplace as an available download. However, if the intention of your installation is to deploy a packaged mobile application built on Agentry, this component should not be installed. Rather, the Agentry SAP Framework ABAP Add-On provided with that application, which will include application-specific components, should instead be installed. The Agentry SAP Framework Foundation is provided for user cases in which a new mobile application for an existing SAP system is being developed from the ground up.

Agentry SDK

The Agentry SDK installer is provided for two development scenarios. The first is when it is desired to develop ActiveX controls for use on Windows Mobile devices. The second is when it is necessary to develop interprocess communications between the Agentry Client and another process running on the mobile device. If neither of these scenarios are applicable to your environment, this component need not be installed. If the Agentry SDK is to be installed and used, see the installation procedure provided in the SAP Mobile SDK installation guide, as well as the information on the structure and usage of the API's within this SDK provided in the *Developer Guide: Agentry Applications*.

Installing the Agentry Editor Plug-In and Eclipse Platform

Prerequisites

The following items must be addressed prior to installing the Agentry Editor Eclipse plug-in:

- Determine if the proper installation for both Eclipse and the Agentry Editor plug-in is the 32-bit or 64-bit build.
- The SAP Mobile SDK 3.0 must have already been installed and access to the items installed by it must be available.
- Obtain the proper ZIP archive of Eclipse from one of the following two locations:
 - **32-Bit Build:** [Click Here for Eclipse 32-Bit Download](#) or enter the following URL in a web browser:

```
http://www.eclipse.org/downloads/download.php?file=/eclipse/
downloads/drops4/R-4.2.2-201302041200/eclipse-SDK-4.2.2-
win32.zip
```

- **64-Bit Build:** [Click Here for Eclipse 64-Bit Download](#) or enter the following URL in a web browser:

```
http://www.eclipse.org/downloads/download.php?file=/eclipse/
downloads/drops4/R-4.2.2-201302041200/eclipse-SDK-4.2.2-win32-
x86_64.zip
```

- Log into the intended host system as a user with Administrative privileges.
- Internet access is needed while the Agentry Editor plug-in installation is active to allow for any dependent modules to be downloaded to the Eclipse implementation.

Task

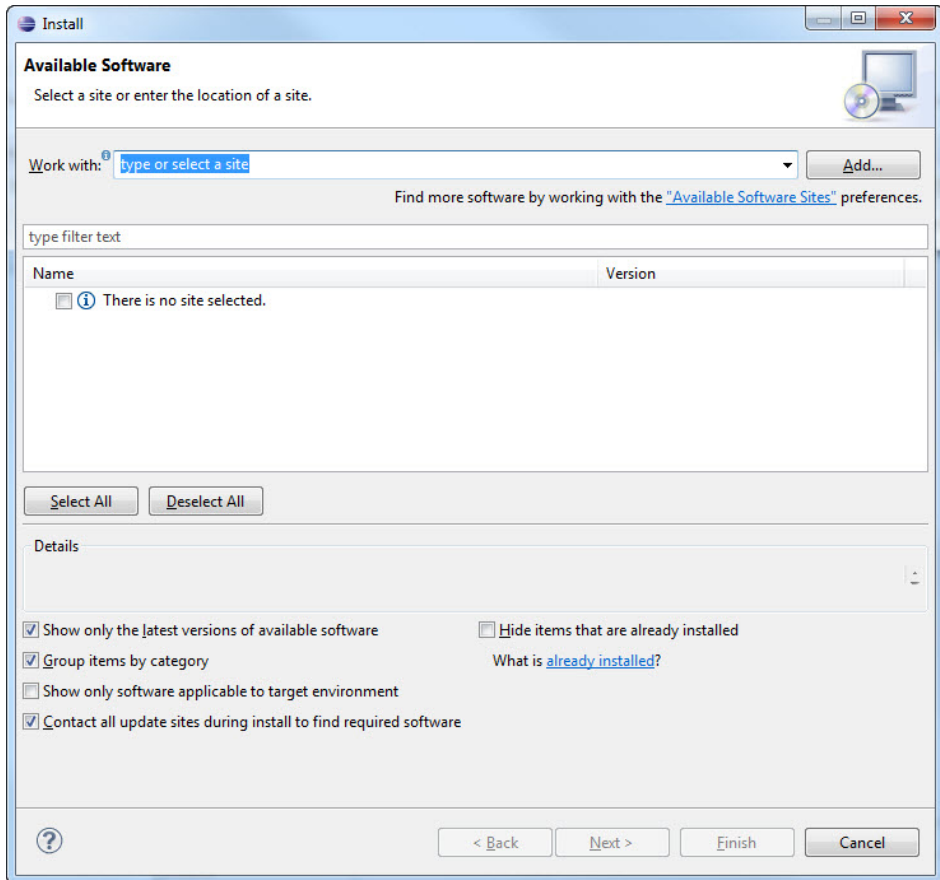
The Agentry Editor is provided as a plug-in to Eclipse. This procedure provides the steps necessary to first install Eclipse and then to install the Agentry Editor to this Eclipse implementation. As noted in the prerequisites, the Eclipse installer should be obtained from the eclipse.org website, with the proper 32-bit or 64-bit build downloaded from that site. Eclipse is provided in a ZIP archive which is extracted to the desired installation location.

Once this is complete, the Agentry Editor plug-in can be added to the Eclipse implementation according to the instructions provided here. The build of the Agentry Editor plug-in must match the Eclipse build, and both 32-bit and 64-bit Agentry Editor builds are available.

1. The Agentry Editor Eclipse plug-in is bundled within the SAP Mobile SDK 2.3 package provided on the SAP Service Marketplace. There are two files available and the one corresponding to either a 32-bit or 64-bit installation should be used for this procedure (as listed below). Make the ZIP archive available on the system where Eclipse is to be installed:
 - **32-Bit:** `Agentry_7.0.0.x_EditorPlugin_x86.zip`
 - **64-Bit:** `Agentry_7.0.0.x_EditorPlugin_x86_64.zip`
2. Navigate in a web browser to one of the two URL's provided in the prerequisites to install either the 32-bit or 64-bit build of the Eclipse platform. Installation of Eclipse involves extracting the contents of the archive to the desired installation location. No installer is run for this process.
3. Install the Java Runtime Environment for Eclipse. This can be installed to the host system using the default location for the JRE; alternately it can be installed to the same location as the Eclipse instance, in a directory named `jre`. For example, if Eclipse is installed to `C:\eclipse`, the JRE can be installed to `C:\eclipse\jre`. This is the recommended location, as installing the JRE here does not require any environment variables to be modified on the Windows host system, as Eclipse looks to this location first for the JRE. If installed to a different location, the paths for the `jre\bin` and `jre\lib` directories of Java installation must be added to the **Path** environment variable in Windows before attempting to start Eclipse.
4. Start Eclipse by executing the `eclipse.exe` executable file found in the extracted folder.
5. During startup you are prompted to select or create an Eclipse workspace. The proper selection is based on the following:
 - New Agentry Installation: Create a new workspace
 - Upgrading an Agentry 5.x or earlier installation: Create a new workspace

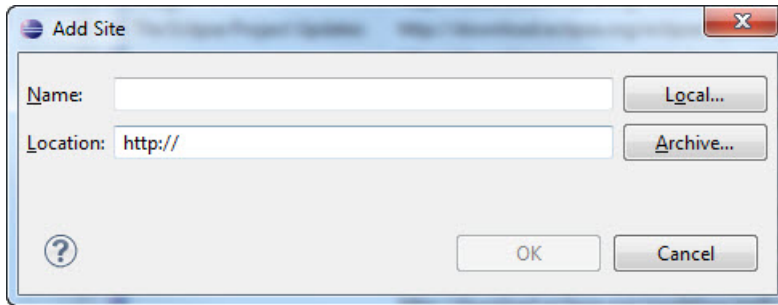
- Upgrading an Agency 6.0.x installation: Either select the existing workspace or create a new one and import the projects after the Agency Editor plug-in has been installed.
6. When Eclipse is running, select the menu item **Help | Install new Software...**

The Available Software dialog displays.



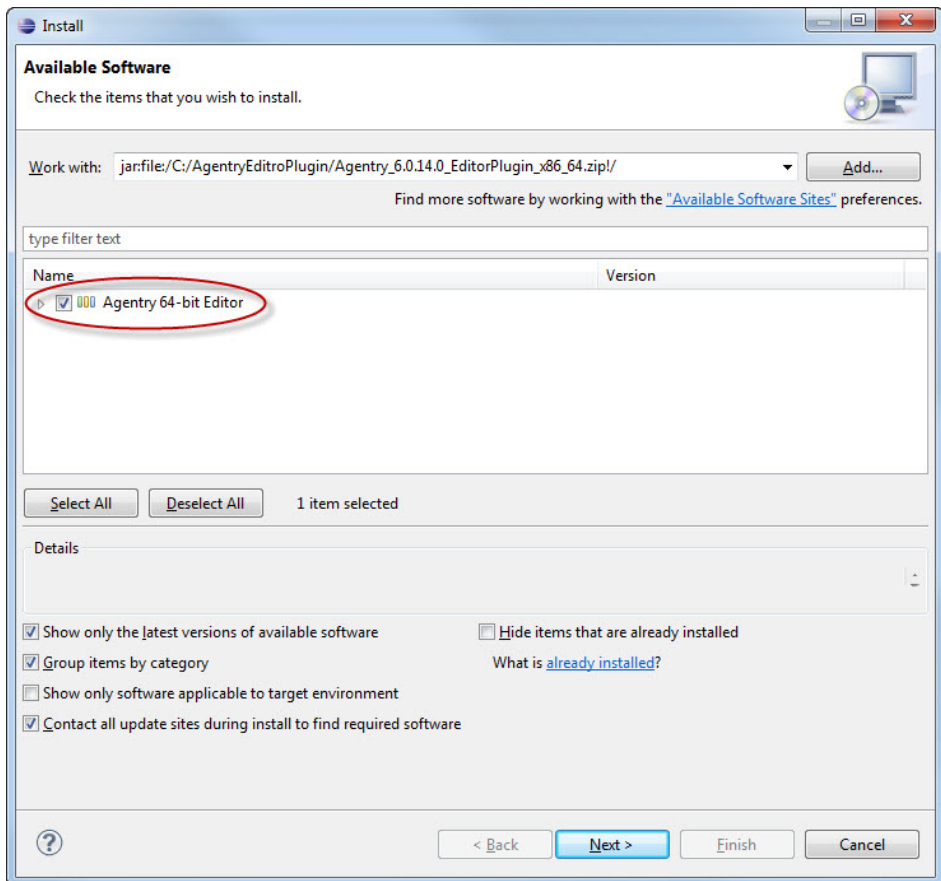
7. Click the [Add] button located to the right of the **Work with** field.

The Add Repository dialog displays.



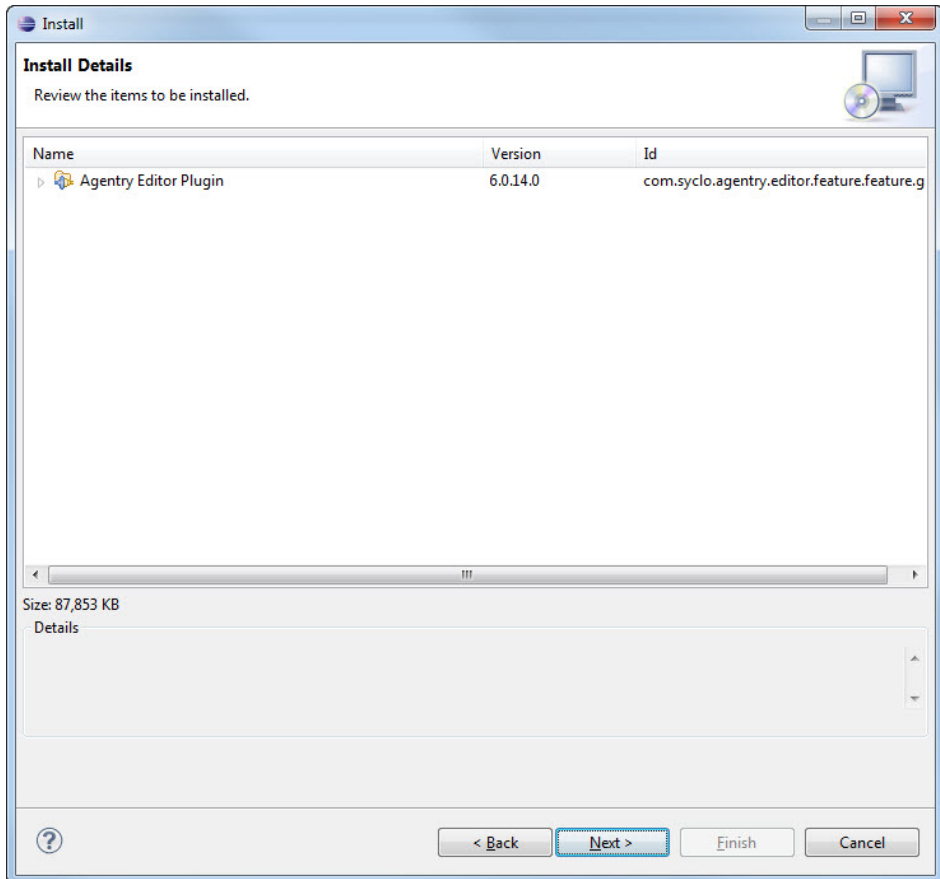
8. Click the **[Archive]** button to the right of the Location field. In the file explorer window now displayed, navigate to and select the archive file for the Agentry Editor plug-in from the first step in this procedure. Click **[OK]** after returning to the Add Repository screen.

This returns you to the Install dialog, where the Agentry Editor plug-in (either 32-bit or 64-bit) is listed.



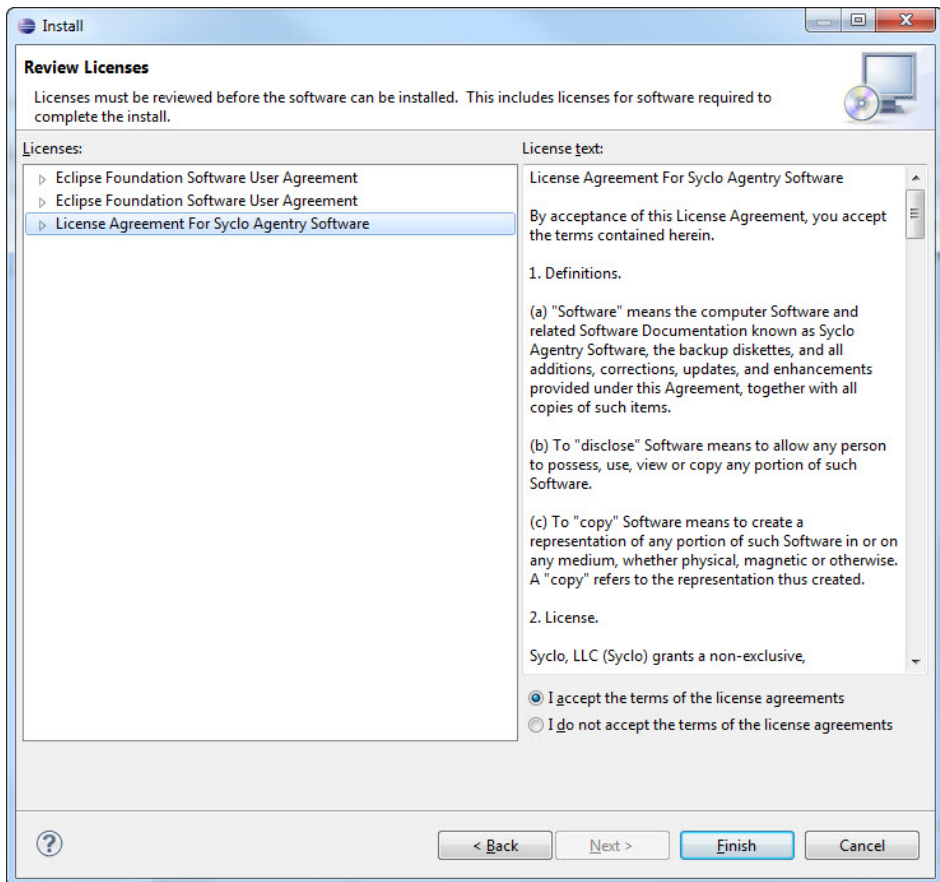
9. Check the box for the Agentry Editor and click the **[Next >]** button.

The Install Details dialog displays.

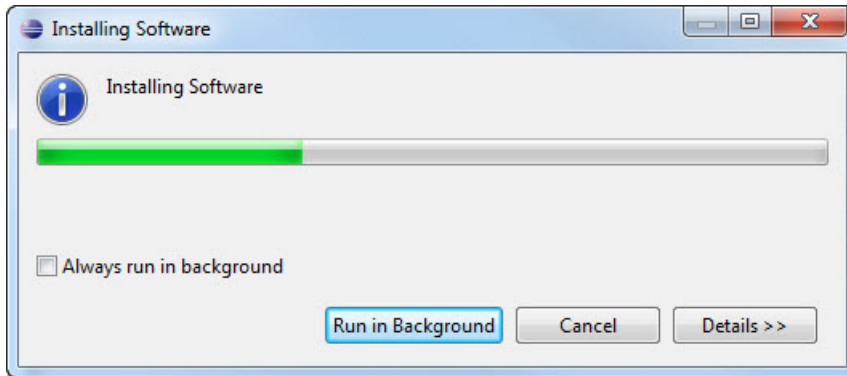


10. Click the **[Next >]** button to proceed.

The License Agreement dialog displays.

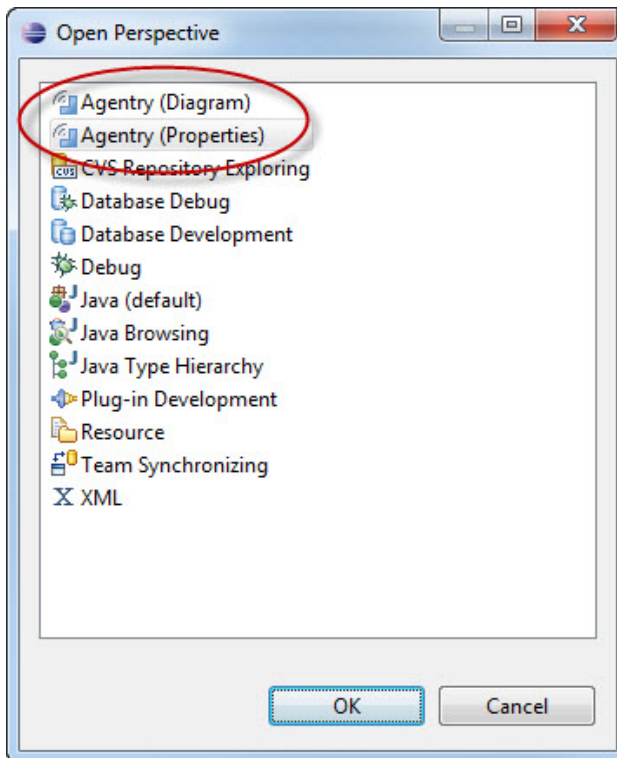


11. Select the radio button below the license information to accept the agreement and click the **[Finish]** button. During this process a security warning is displayed related to unsigned content. This is a result of certain `.jar` files not having been signed. You can safely continue with the installation by selecting the option to continue in the warning dialog. The installation will commence and a status dialog is displayed.



12. When the plug-in has been installed, a prompt is displayed to restart Eclipse. Select the **[Restart Now]** button. When Eclipse restarts, go to the workbench. Select the menu item **Window | Open Perspective | Other...**

The Open Perspective dialog displays, where the two Agentry perspectives are now listed.



13. Either of these may be selected to begin working with the Agentry Editor.

The Eclipse environment and Agentry Editor plug-in are installed on your system.

Next

Additional configuration of Eclipse is needed as it relates to the Agentry Editor plug-in. This configuration is performed within the Eclipse Platform and includes the following general items:

- Agentry projects work with several file types. You must create file associations within Eclipse for these types to allow the platform to properly handle, display, and edit them.
- Script files, including SQL and shell or batch scripts created in Agentry Editor are saved with a Unicode encoding. The default file encoding for an Eclipse workspace is different. You must modify file encoding options within the Eclipse preferences.
- When working with a database back end system, you must configure the Data Source Tools installed with Eclipse in order for the Agentry Editor connector studio to work with a database back end.

It may also be necessary to complete the configuration of one or more Agentry Servers. A publish from the Editor is a part of the Server configuration process as it relates to the Agentry application project's system connections. If connectivity was already configured between the Agentry Server and the back end system, the publish is not necessary for connection configuration.

Agentry Editor and Eclipse Platform Configuration Overview

After the Agentry Editor plug-in has been installed to Eclipse there are configuration tasks to be accomplished. All tasks are performed from within the Eclipse Platform itself. These tasks relate to several areas of the Agentry Editor and its management of application projects.

The information provided concerning these configuration tasks should be considered a recommended best practice. For a particular development environment, different options or configurations can be implemented based on need. Contributing factors and items of consideration include the overall uses for the Eclipse implementation.

Connector Studio and the Data Tools Platform

The Data Tools Platform (DTP) is a project for the Eclipse platform that provides several powerful tools for work with different data sources. The Agentry Editor leverages the power of this project by extending the tools provided in the Connectivity DTP subproject with the Agentry Connector Studio.

The Agentry Connector Studio is a tool within the Agentry Editor plug-in that allows for the development of object, transaction, and step definitions using the available schema information of a data source. When an Oracle or SQL Server database system is that data source there is some configuration needed within certain Connectivity components. Specifically, connections must be configured to these database systems in order to access this schema information.

The configuration of a connection includes the creation of a Driver Definition and a Connection Profile within the Connectivity tools. The procedure provided on configuring

these items will be an example on configuring these tools for the Agentry Editor. For complete information on all configuration options as well as descriptions on the functionality and uses for these tools see the Eclipse.org website.

Java Projects

If the application being developed or configured includes a Java Virtual Machine system connection, a Java project should be created within the Eclipse platform. This project should include the Agentry Java API version 5, as well as any other needed resources to properly build the Java logic for the mobile application's data synchronization. This will also expose the data members of the Java resources to be used to the Agentry Connector Studio. This will then allow for objects, transactions, synchronization steps, and properties to be defined based on the information provided by the Java packages in the Java project.

This process is performed using the Eclipse New Java Project wizard. The new Java project can be created before or after the Agentry application project is created, but does require the Agentry Development Server to be installed and accessible to the host system of the Eclipse platform, as this location includes the Agentry Java API JAR file to be used in the Java project.

File Associations and File Encoding

An Agentry application project can contain several different file types as a part of the project definitions. These can include scripts for SQL, batch files, and shell scripts, Java source files, bitmap files and large markup files related to XML processing. Configuration of the Eclipse Platform and the tools outside of, but still used by the Agentry Editor plug-in for Eclipse, can be needed in order for these different files to be handled correctly.

Also, most of the above mentioned files are stored as text files, with the obvious exception being bitmap files for image definitions. The encoding of these text files when created by the Agentry Editor is a unicode format of UTF-16. Files for other purposes outside of the Agentry application project may not be encoded in this format. Therefore the configuration related to file encoding must be performed in a manner that supports the different file encoding formats that may be needed.

Creating Java Projects for Agentry Java API Development

Prerequisites

Before creating a Java project in Eclipse for use with Agentry Java API development work, the following items must be addressed:

- The SAP Mobile SDK installer must have been successfully run on the same system as the Eclipse and Agentry Editor plug-in have been installed; or, to a system that is accessible from this system.
- All Java resources needed for the development work should be accessible and available to the Eclipse platform host system. This can include jar files as well as other code and documentation (i.e. Javadoc archives) resources that may be needed for the development work.

- SAP Java Connector (SAP JCo) version 2.1.8, or any later 2.x version, is required for development and runtime environments in which the Agentry SAP Framework Foundation is used. JCo version 3.0 or later is not supported at this time.
- Many mobile applications that make use of the Java Virtual Machine system connection type include multiple layers of Java code libraries that reside logically between the Agentry Java API and the logic for the specific mobile application. In such situations these other libraries must exist in a location that is always available to the Eclipse platform.
- Determine the proper version of the JRE to be used for the Java development portion of the application. The Ganymede release of Eclipse requires the JRE version 1.6. The development project may need a different version of the JRE based on the requirements of the back end system with which synchronization will occur through the Java system connection. A part of creating the Java project in Eclipse is the optional designation of the specific JRE version to use.
- The Java Perspective should be open in Eclipse. Note that by default this perspective is always open, though it can be closed at any time. To reopen this perspective, select the **Windows | Open Perspective | Other...** menu item in Eclipse. Select the Java (default) perspective in the list shown.

Task

This procedure provides the basic steps of creating a Java project to contain the development work performed with the Agentry Java API. When this procedure is complete a Java project will exist in the current Eclipse workspace that will include the resources needed for performing development work related to the Java Virtual Machine system connection type. This project will include:

- The Agentry Java API jar file as installed by the SAP Mobile SDK.
- A source folder location for Java logic created for the mobile application.
- Any other resources that may be needed for the project (e.g. `sapjco.jar`).

This procedure is not intended to be a comprehensive discussion on Java projects created in the Java Perspective in Eclipse, nor is it a discussion intended for the novice Java developer. A level of understanding and knowledge is assumed on the part of the reader related to these topics. Full information on the JDT project, which includes the Java Perspective, can be found at the Eclipse help site:

<http://help.eclipse.org/documentation>

At the above link see the documentation matching the Eclipse version you installed for the Agentry Editor plug-in.

1. Start the Java Project Wizard in Eclipse using one of the following methods:
 - Select the menu item **File | New | Java Project**.

- Right click in the Package Explorer View in the Java Perspective and select the pop-up menu item **New | Java Project**.
- Right click in the Project Explorer View provided with the Eclipse Platform and select the popup menu item **New | Project**. In the screen displayed, select the tree control item **Java | Java Project**.

Any one of these actions will display the New Java Project wizard.

2. In this first wizard screen, begin by giving the project a name. This will be the identifier for the Java project in the Eclipse Project and Package Explorer views. Then:
 - a) Select the Use Default Location checkbox to store the project resources within the default location within the current Eclipse workspace; alternately, deselect this box to specify a different location if needed.
 - b) In the JRE section specify the JRE version for the Java development project. This selection must match the needs of the back end system for which the mobile application development work will be performed. This need not be the same JRE version as Eclipse uses for its own execution. The version of the JRE to be used must be installed prior to making this selection. The third option is to select a Java execution environment.
 - c) The Project Layout specifies the location of the source and built class files for the Java project. The default is the separate locations for source and output files. The default folders are `src` and `bin`, respectively, which will reside as sub-folders to the project folder in the Eclipse workspace. These locations can be modified by clicking the [Configure default](#) hyperlink in this section.
 - d) The new project can be added to a Working Set. If unfamiliar with working sets in Eclipse, see the help topic “Working Sets” in the Eclipse publication *Workbench User Guide*.
3. Once these options are set, click the **[Next >]** button to proceed.

The next screen of the New Java Project wizard is displayed. This screen contains multiple tabs for further configuration of the new Java project’s build settings, including source file location, included projects, libraries, and the build order and export.

4. Set the Source options as needed for the given project. Options here include selecting additional source folders outside the workspace, adding new folders to the project within the workspace that will be used as source folders, and other source file-related tasks.
5. Select the Projects path to set other projects to be included on the build path for the current project. This project must reside in the same workspace as the new project being created. Once added, options exist to select the files or sources within the project to be used. The build order for the project selected here is set in the Order and Export tab.
6. Select the Libraries tab. Here is where the libraries related to the Agentry Java API, as well as other libraries needed by the back end are selected and included in the project. this can also include the SAP JCo jar file, if applicable to your implementation and development work. To add libraries to the project’s build path, begin by clicking the button to the right to add the type of library needed. For the Agentry Java API, click the Add External JARs... button. Navigate to the location of this Jar file, as provided by the SAP Mobile SDK

installer. Once added, you can expand the new node for this jar file and add the Javadoc source for the Agentry Java API. This is a ZIP archive named `Agentry-v5-doc.zip`.

7. Once the libraries have been added, select the Order and Export tab. Here the build order is specified for the libraries and external projects for the new Java project. The specific build order is completely dependent on the project being created. However, the JRE System Library will likely always be first. Within this order, the Agentry Java API library should be ordered before any libraries or projects that extend the Agentry classes or interfaces. Also, it is likely any resources from the back end system should be ordered before the mobile application-specific items, which may include “application suite” libraries, as it is likely these too will extend or access these items. Once the build order is set, click the **[Finish]** button to create the new Java project.

A new Java project will be created in the current Eclipse workspace. This project will now be accessible to the Agentry application project. When defining Java steps, or complex table or data table Java synchronization components, it will be possible to select items contained in the Agentry Java API, as well as those in the other libraries and projects for the Java project.

Next

Changes to the project’s configuration can be made by selecting the project in the Project Explorer or Package Explorer and selecting the Properties menu item.

Configuring Eclipse File Associations for Agentry Projects

Prerequisites

The following items must be addressed before performing this procedure:

- For Eclipse implementations to which the Agentry Editor plug-in was added, review the current file associations before making any modifications. Any current configuration of file associations for file types of `.sql`, `.bmp`, `.bat`, and `.sh` should be noted.
- Determine the desired editor within Eclipse to use for each of the file types that may be a part of the Agentry application project. This may be done now or during this configuration procedure.

Task

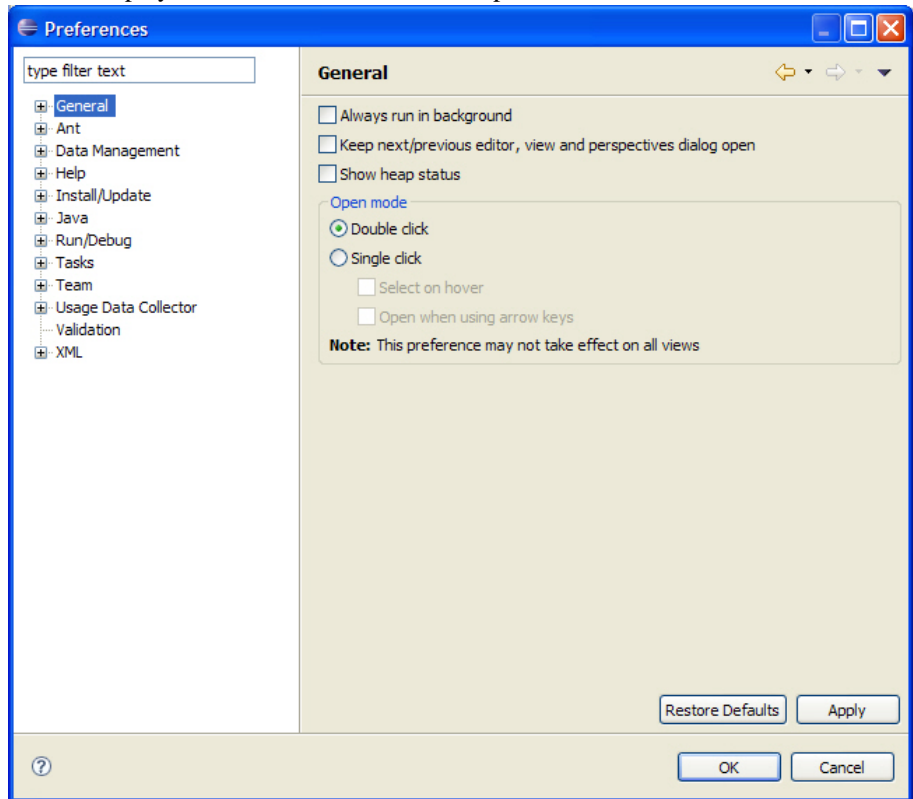
In this procedure the file associations within Eclipse will be configured for the file types that may be a part of an Agentry application project. These file associations will determine which editor or view within Eclipse will be used to display the file contents and allow those files to be edited. The file associations configured in this procedure are those available in Eclipse as provided by the Agentry Editor installer. If there are other tools available for a particular implementation, those may be used as preferred or desired. The process for creating these file associations is the same. The selection of the particular tool within Eclipse for a particular file

type is dependent on those available and preferred. Eclipse must be running to make these modifications.

Specifically the editors for bitmaps (.bmp), Windows batch files (.bat), SQL scripts (.sql), and Linux and Unix shell scripts (.sh for Agency application purposes) will be configured in this procedure. For shell scripts and batch files the text editor provided with Eclipse will be configured in this procedure. Other tools exist for these types of files as plugins to the Eclipse Platform. If those are to be used, review the documentation and instructions provided with those tools for configuration.

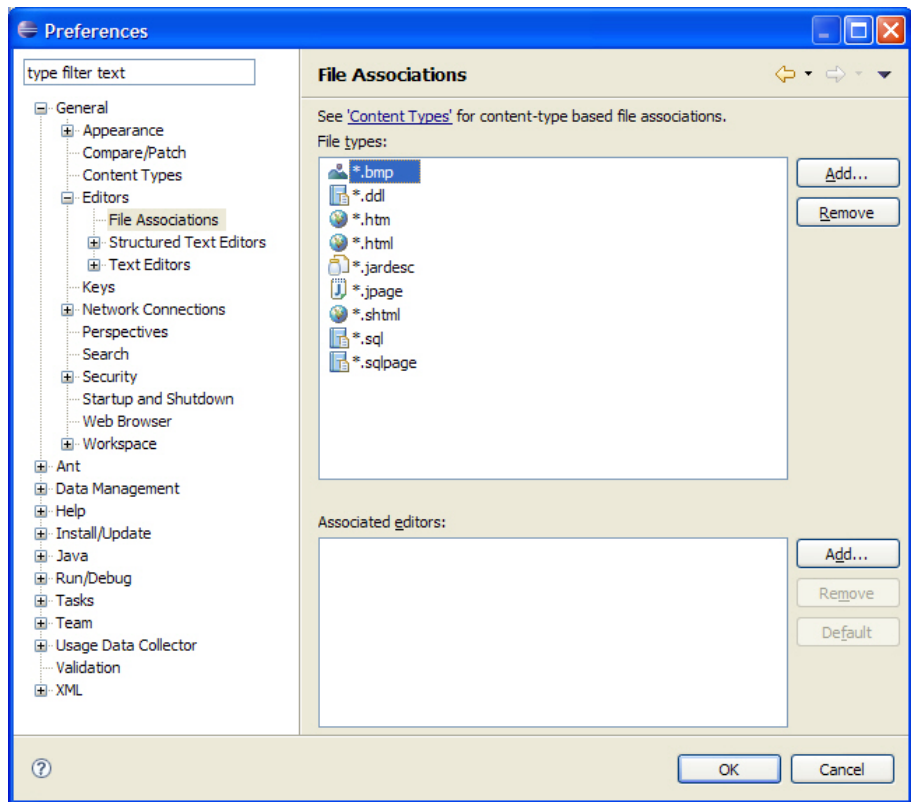
1. Select the Eclipse menu option **Window | Preferences**.

This will display the Preferences screen in Eclipse:



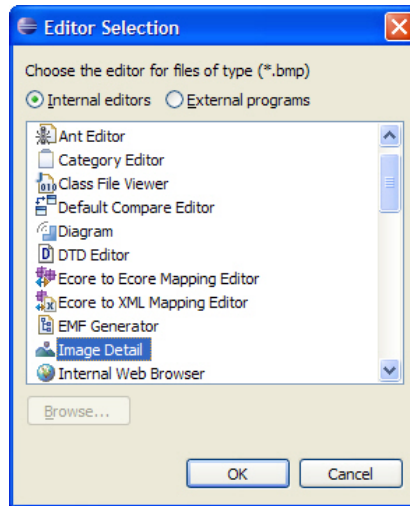
2. In the tree control on the left, select General | Editors | File Associations.

This will display the File Associations preference page:



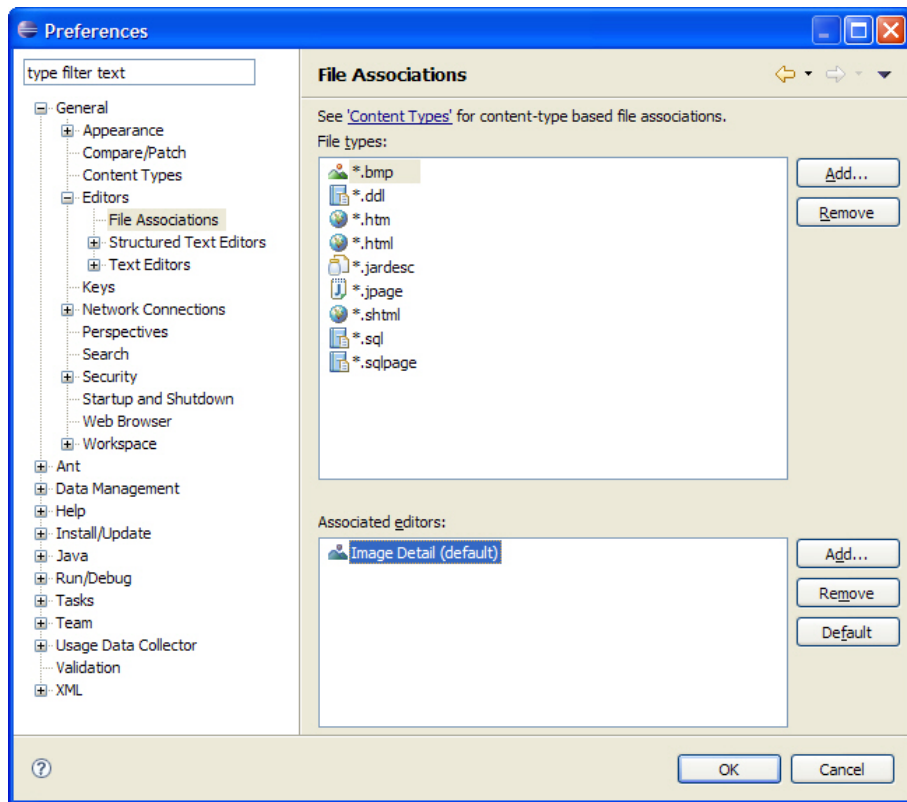
3. In the top list on this page are the file types for which associations can be made. The bottom list contains those editors or tools configured for the file type selected in the top list. File types can also be added to the top list. Begin by selecting the item * . bmp in the top list. The list of associated editors will be empty. Click the **[Add...]** button to the right of this empty list to create a file association.

This will display the Editor Selection list, where an internal or external editor can be selected. Here an internal editor will be used.



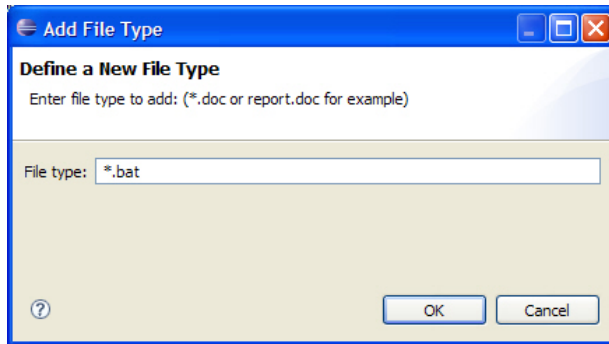
4. For a bitmap file you can select the Image Detail editor in this list. Then click the [OK] button to make this selection.

This returns you to the File Associations preference page, where the Image Detail editor is now configured as the default editor for bitmap files.



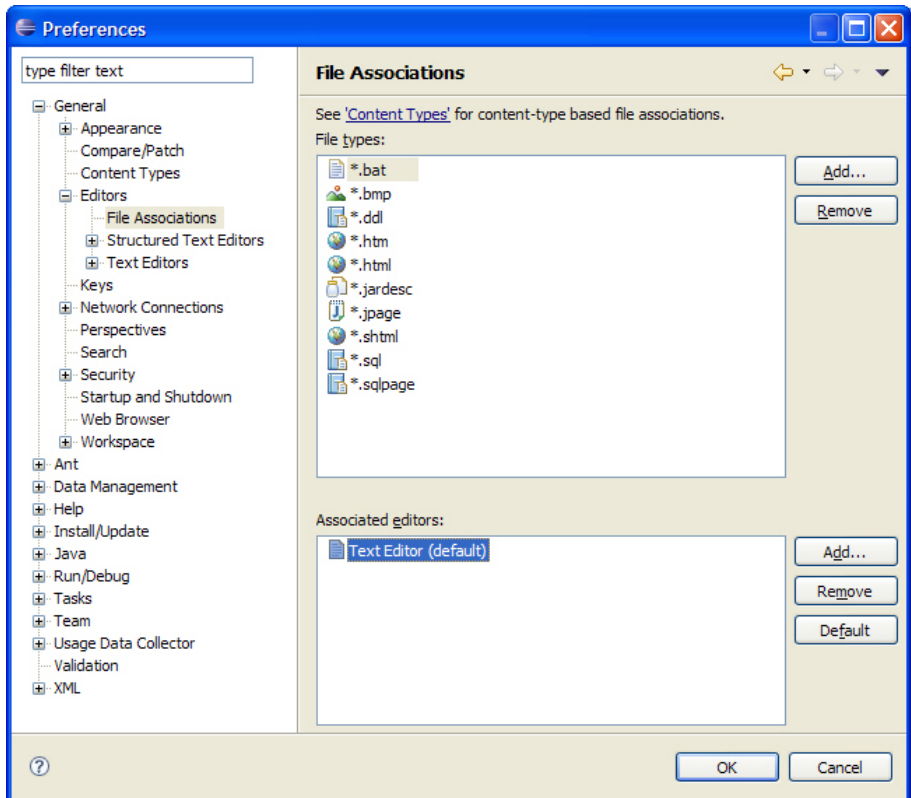
5. Next the remaining file associations can be created. Begin by checking the configuration of `.sql` files. By default the SQL File Editor is configured as the default for this file type. This is a part of the Data Tools package provided with Eclipse. You may select another editor if one is available, though this is the recommended editor for Agentry application projects.
6. The remaining file types that may need to be configured are those related to File System connections. These can be batch files for Windows deployments or shell scripts for Linux and Unix deployments of the Agentry Server. By default these file types are not listed in the Preference page for file associations. They can be added. To do this, click the **[Add...]** button to the right of the File types list.

This will display the Add File Type screen as shown next.



7. Enter the file extension in the format `*.ext` in the field provided and click the **[OK]** button.

This will return you to the File Associations preferences page where the new file type will now be listed.



8. This new file type can now be selected and an Editor can be associated with it, just as before. In the previous example Windows batch files are configured to have the Text Editor as the default editor.

9. This procedure can now be repeated for any other file types. In the case of Linux or Unix development, the recommended file type of `.sh` should be added and configured. While the file extensions are not required by these operating system types for proper execution, within the Agentry application project an extension is needed. This is so a file association can be configured for proper display and management of the files.

The file associations for those file types used in Agentry application projects have now been configured. In the examples provided in this procedure the following configurations have been made:

- Bitmap files will be displayed in the Image Detail editor.
- Windows batch files will be displayed in the standard Text Editor for Eclipse.
- Shell script files will be displayed in the standard Text Editor for Eclipse.

Configuring Eclipse File Encoding for Agentry Projects

Prerequisites

Before performing this procedure the following items must be addressed:

- The Agentry Editor plug-in and Eclipse Platform must be installed.
- The file associations should be configured for `.sql`, `.bat` or `.sh`, as needed, within the Eclipse Platform.
- Identify the file encoding format for any Java source files that may be opened within the same workspace as the Agentry application project.

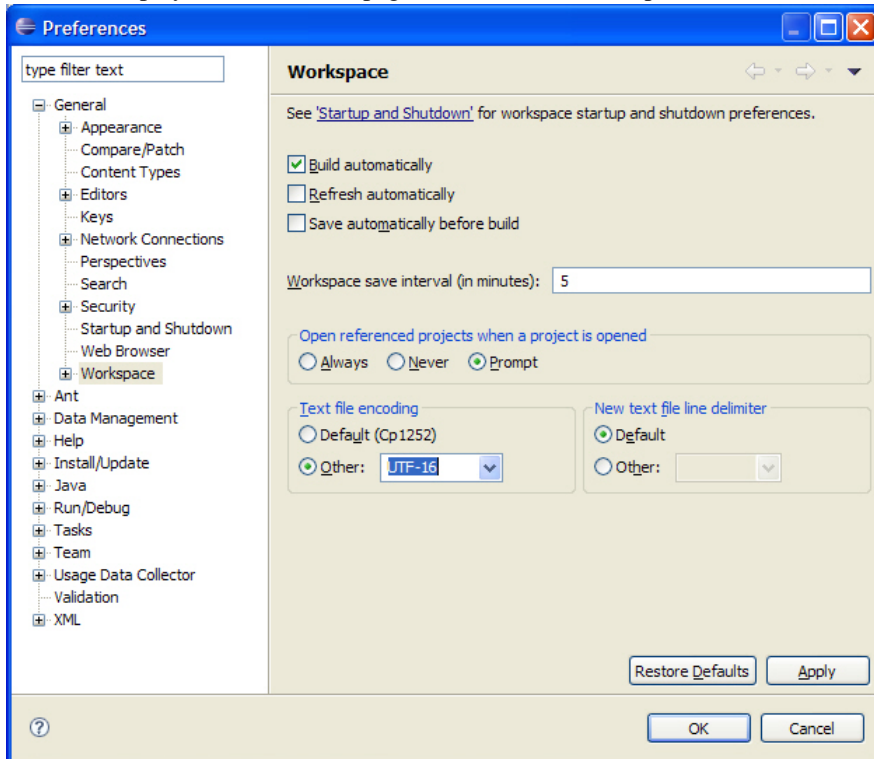
Task

This procedure describes the steps to configure the file encoding settings and preferences within the Eclipse platform as required by the Agentry Editor plug-in. The assumption is that the file editors for the various file types have been configured based on the recommendations provided by Syclo. If one or more different file editors are to be used for certain file types, the following procedure may or may not be valid. Those tools described here do not support or provide editor-specific file encoding, instead inheriting this behavior from the Eclipse workspace settings.

This procedure configures the current Eclipse workspace to use the file encoding format UTF-16, which is a unicode format. This will then result in all text editing tools treating all text files opened in any editor as if they are encoded in this format. Certain files external the Agentry application project but possibly a part of its overall implementation may not be encoded in this format. The primary example of such files would be Java files for a JVM system connection. The Java perspective within Eclipse does have a preference setting for file encoding. This is set for an individual Java project. The proper encoding setting for Java projects is dependent upon how those Java source files were encoded upon creation. The common option for such files is the UTF-8 encoding. However, other encoding formats are possible and this should be determined before making changes.

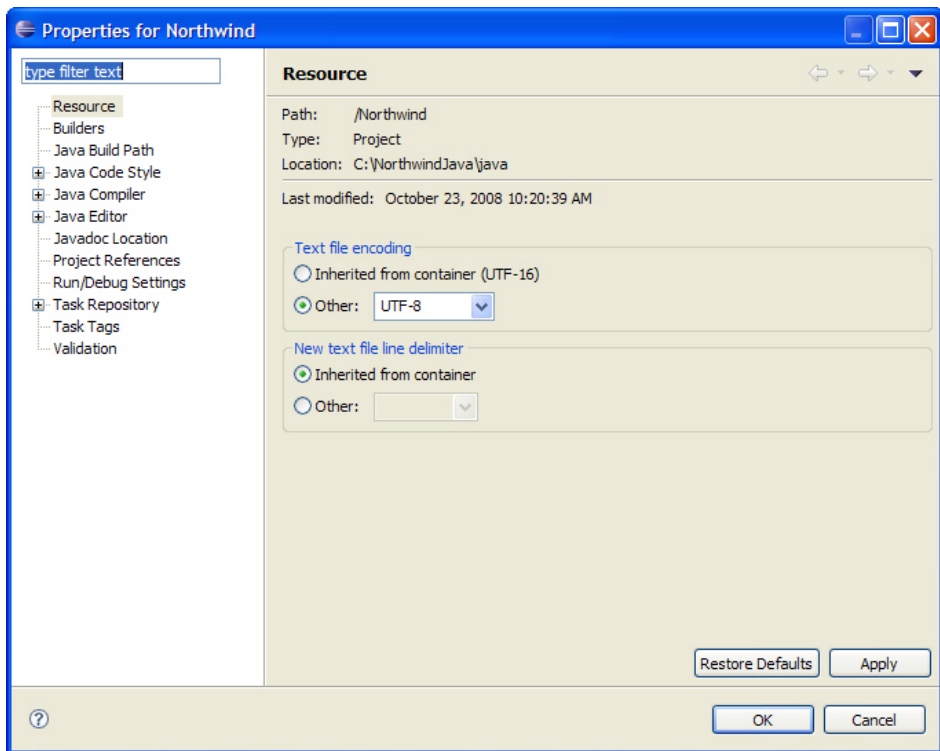
1. Start the Eclipse platform and open the workspace to be used for the Agency Editor. Once Eclipse is started, select the menu item **Window | Preferences**. In the tree control on the left of the Preferences screen select General -> Workspace.

This will display the Preferences page for the current workspace.



2. On this page look for the section **Text file encoding**. Select the **Other** radio button and then select the item UTF-16 in the associated drop down list.
3. Click the **[OK]** button to close the Preferences screen and apply the changes. If a Java project is not to be opened within this same workspace, or if the Java project's files have been encoded with UTF-16, no further actions are needed. Otherwise, continue on with this procedure.
4. Switch to the Java perspective in Eclipse and open the Java project for this same workspace. Select the menu item **Project | Properties**.

This will display the Properties page for the java project:



5. In the tree control to the left select the item **Resource**. Then look for the section **Text file encoding**. Select the radio button **Other** and then select the proper encoding option based on the encoding of the source files in the Java project. Note that if these files are encoded in either ASCII or UTF-8, then UTF-8 can be selected. For other file encoding formats select the proper item.
6. Click the [OK] button to close the Properties page and save these changes.

This procedure results in changing the Eclipse workspace's default file encoding for all text files to the unicode format UTF-16. This is the encoding in which all text files created by the Editor are formatted. The exception to this are java source files.

Creating a Connection Profile for the Agentry Connector Studio

Prerequisites

The following items must be addressed prior to performing this procedure:

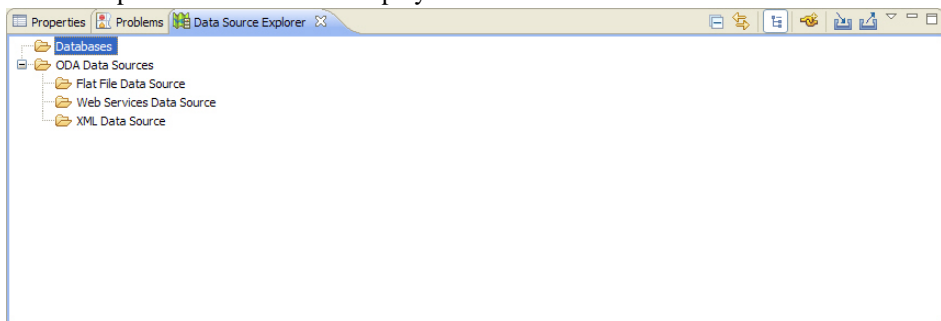
- The connection between the Data Source Tools and the target database uses JDBC. The desired JDBC drivers should be selected and the associated resource files (jar files or otherwise) should be located and noted. This information will be needed when creating a Driver Definition.


- If a suitable Driver Definition has already been created for the Data Source Tools, the name of the definition should be noted for use in the following procedure. In this case, a new Driver Definition will not need to be created.
- Information about the selected JDBC drivers should be gathered, specifically the attributes and values needed to configure a connection using the specific driver package. This will likely include the syntax for specifying the database host and server, login and password information, and the Java class or package within the JDBC driver to be used.

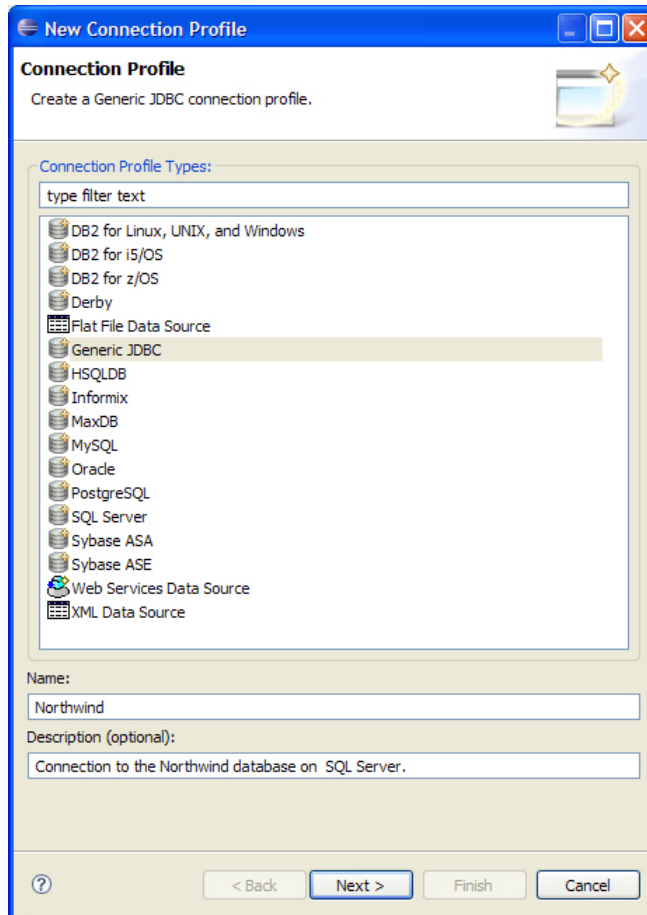
Task

The connection created in this process will be accessible through the Data Source Explorer view available in Eclipse. This view is not a part of the Agentry Editor Eclipse Plug-in but rather is provided with the Eclipse Platform. Therefore, detailed instruction and information on its functionality and behavior is not provided by Syclo. This information can be obtained from the Eclipse.org website. Once this connection is created, it will be listed in the Data Source Explorer view. From here you will be able to perform all functionality supported by the Data Source Tools plus use the Connector Studio functionality provided with the Agentry Editor.

1. Open the Data Source Explorer view by selecting the menu item **Window | Show View | Other...**. In the tree control displayed, expand the Connectivity node and select the Data Source Explorer item. This will display the view as shown next:



2. You can now create a new connection to a database using this view by either right-clicking the Databases node and selecting the menu item **New...**; or by clicking the tool bar button . Either will display the New Connection Profile Wizard.



3. This screen lists the database types to which a connection can be made. This selection is largely based on not just the database type, but also on the selected JDBC driver package to be used. For this example, the Connection Profile Type to be used is Generic JDBC. Give the connection profile a name and a description and click the **[Next >]** button.

New Connection Profile

Specify a Driver and Connection Details

Enter a database name.

Drivers: Generic JDBC Driver

Properties

General Optional

Database:

URL:

User name:

Password:


☐ Save password

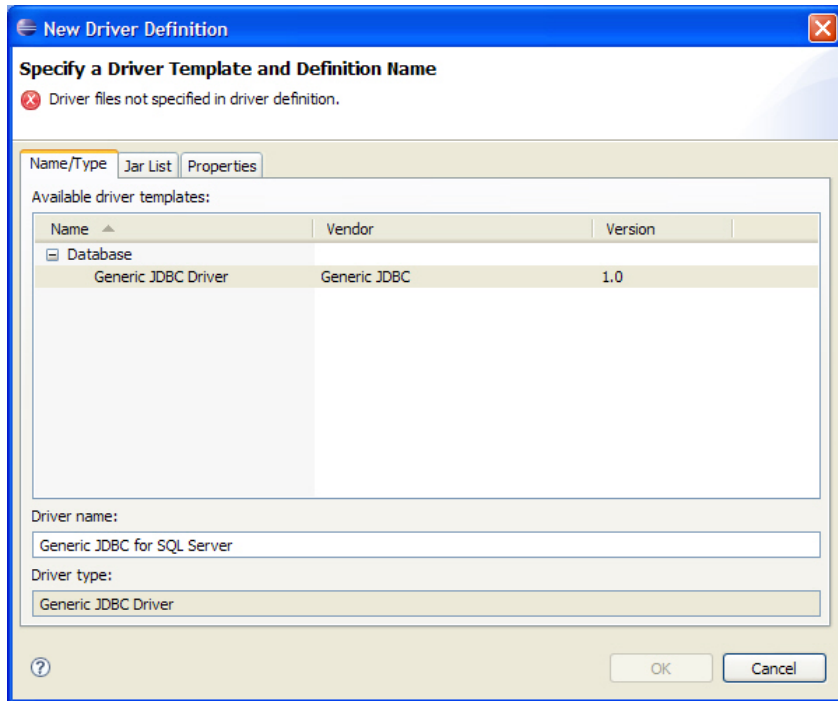
☒ Connect when the wizard completes

☐ Connect every time the workbench is started

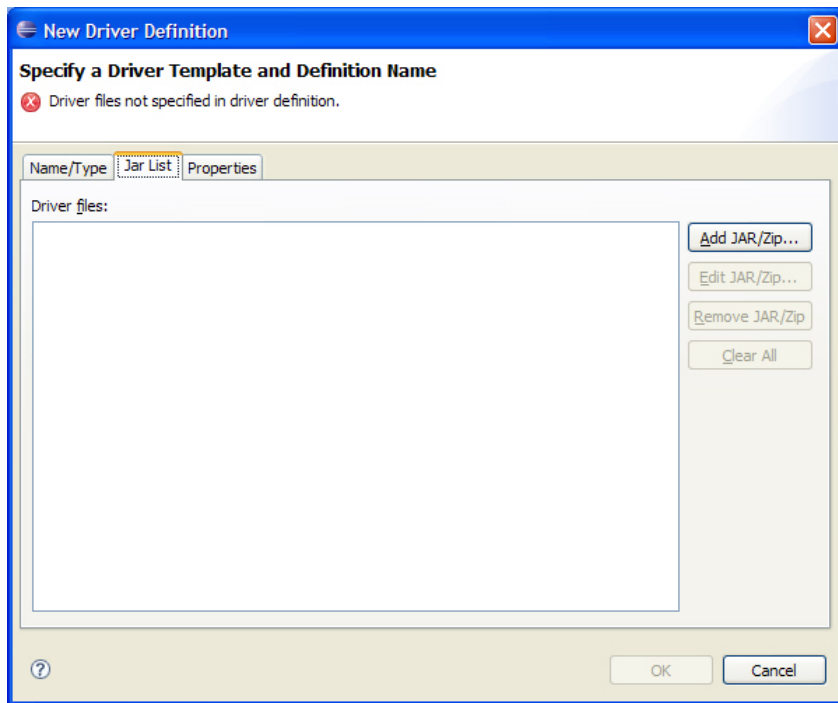
Test Connection

< Back Next > Finish Cancel

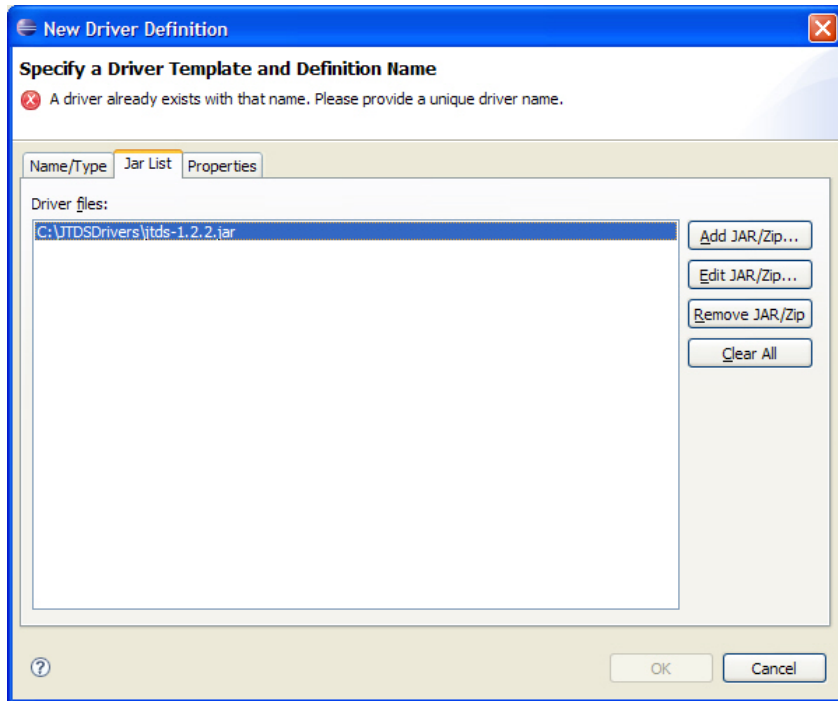
4. In this next screen the driver to be used for the connection is selected. If a driver profile exists that is suitable for the connection, select it from the Drivers list. In this case you can skip the next instructions on creating a new Driver profile.
5. To create a new Driver Profile, click the new Driver Profile Button  to the right of the Drivers list. This will display the New Driver Profile wizard.



6. In this first tab you can select the driver template. This list will vary depending on the type of Connection Profile you are creating. In this example there is only one option, Generic JDBC Driver. Enter a name for the Driver Definition.
7. Next select the Jar List tab. This list contains the jar files for the JDBC driver package to be used for this Driver Definition. For this example the list is empty initially.

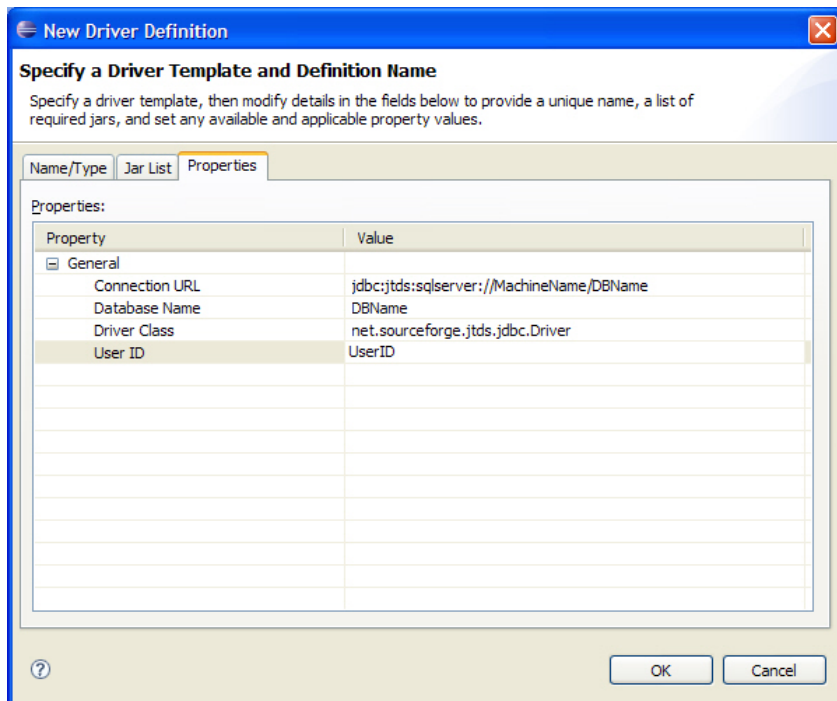


8. Add the proper jar file(s) to this list by selecting those provided with the selected JDBC driver package. This is done by clicking the Add Jar/Zip button and then browsing to the jar file on the file system.



Here the selected item is the jar file for the jTDS 1.2.2 JDBC driver package provided by SourceForge.net. For other driver packages the selected jar file(s) will be different.

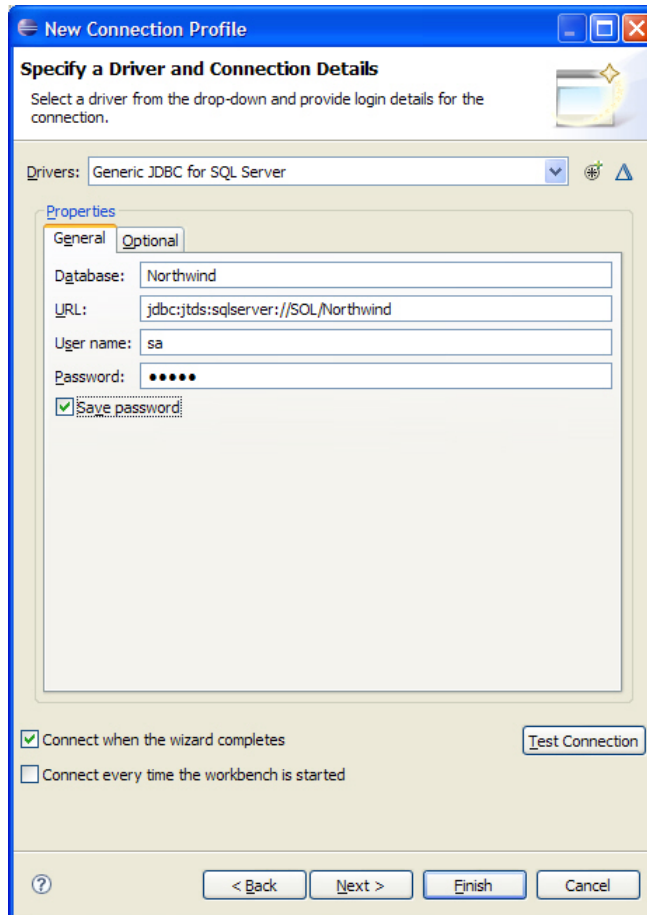
9. Next select the Properties tab. The items listed in this tab will be dependent on the selected jar file and are those items the JDBC driver requires to establish connections.



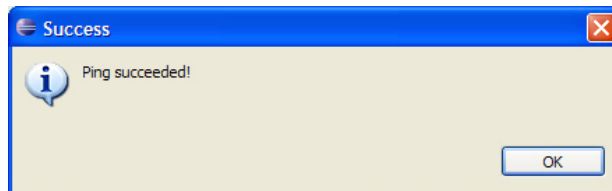
The values set here should be left to generic or template types of values. This driver definition, once created, can be reused for multiple actual connections to a database of the same type. The values entered here will be presented going forward when this driver definition is selected for a new Connection Profile and can be set at that time to the database specific values needed for a connection. Once these items are set, click the **[OK]** button.

The New Connection Profile Wizard will now be displayed again. The values set in the fields on this screen will match those entered for the selected JDBC driver properties.

10. Now alter these settings by entering the proper values for the specific database connection to be created. In this example, the database name, URL, User name and Password are altered for a connection to a Northwind database in MS SQL Server.

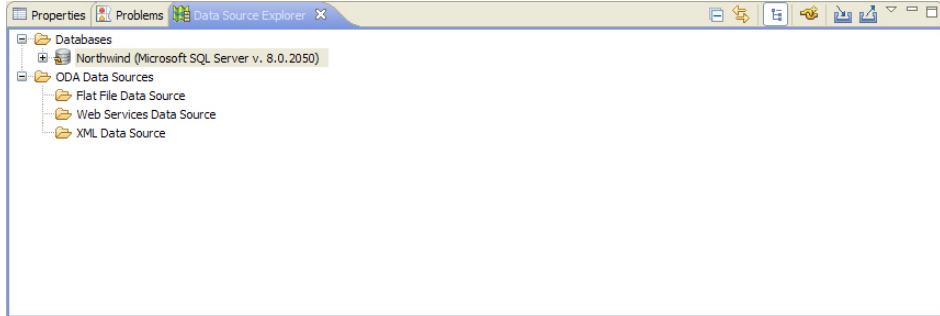


11. If the target database is available you can now test this connection by clicking the **[Test Connection]** button. If the connection is successful you will see the following pop-up screen.



12. Click OK on this prompt. Finally, choose whether to connect when this wizard closes and also whether or not to connect automatically when the Data Source Explorer view is displayed in Eclipse. Then click the **[Next >]** button to view a summary of this configuration, or the **[Finish]** button to close this wizard.

The Data Source Explorer View will now list the new Connection Profile under the Databases node with the name you entered.



The database connection created will allow for use of the schema information provided by that database in the object definition process. The developer can use the Agentry Connector Studio, accessed through the Data Source Explorer view, to select a table and to create or modify an object definition, as well as its related transactions and SQL Step definitions, based on the structure of the database table.

Agentry OpenUI SDK for iOS Setup Overview

The Agentry OpenUI for iOS clients is provided within the SAP Mobile Platform SDK. It is provided in a single TGZ file, referred to generally as the Agentry Client framework, and can be used to create a development project within the Xcode IDE on Mac systems. It is provided in the native Objective C programming language for iOS devices. Also included in this framework is a sample project with basic overrides of each detail screen field edit type provided as examples.

When creating the project within Xcode, the developer has two options. The first is to manually create the project in Xcode, pulling in the various resources provided in the Agentry Client framework and setting the various configuration options for the project. Instructions are provided here to support this method if desired.

The second, and recommended option in most situations, is to use the prebuilt empty project file provided in the Agentry Client framework, named `SMPAgentryClientFrameworkSetup`. This project contains all the same settings and options as described in the manual process. It can be used as is within Xcode to support a quicker and easier project setup. In general it is only necessary to create the project manually in situations where there is a need for significant differences from the recommended setup. Such situations are extremely rare and should not be necessary in most use cases.

Framework Contents

The resources for the OpenUI SDK are provided in the Agentry Client framework, which is found in the file `SMPAgentryClientFramework-iOS-7.0.x.x.tgz`. This library and the resources it contains includes all functionality of the Agentry Client for iOS and includes the OpenUI SDK to allow for the detail screen fields to be overridden with custom

controls; as well as exposing the necessary resources to rebrand the Agentry Client. Due to limitations with Apple's current support of dynamic frameworks, this framework is intended to be statically linked to the application.

To support testing on both iOS devices and within the simulator, the framework standard library is provided with slices for instruction set architectures armv7, armv7s, and i386.

The framework contains the following directory structure after it has been expanded:

```
//SMPAgentryClientFramework/iOS/
```

```
./SMPAgentryClient.Framework/Versions/Current/Headers
```

Headers needed for adapters and models of SMPOpenUI

```
./SMPAgentryClient.Framework/Versions/Current/Resources
```

SMPAgentryClientResources.bundle - Image, sound, and other resources

```
./SMPAgentryClient.Framework/Versions/Current/SMPAgentryClient
```

SMPAgentryClient - The static library containing the iOS Agentry Clientcode

```
./Samples/SMPAgentryClientFrameworkDemo/
```

```
./Samples/SMPAgentryClientFrameworkSetup/
```

```
./SampleApp/
```

./OpenUI-app.agx - Agentry application project corresponding to the Xcode demo project provided in the Samples directory

./OpenUIPlayersDB.txt - Sample data in SQL script for back end DB

OpenUI for iOS Manual Xcode Project Setup

Prerequisites

The following items must be addressed prior to performing this procedure:

- The SAP Mobile Platform SDK 3.0 contains the necessary resources for this procedure and must be installed. Note the installer is only available for Windows systems and the iOS

framework must be manually copied to the Mac system after installation. This framework file can be found in the directory `AgentryToolkit/OpenUISDK/iOS/SMPAgentryClientFramework-iOS-7.0.x.x.tgz`

- Xcode version 4.6 must be installed to the Mac system where the project is to be created.

Task

This procedure provides the steps necessary to create a project in Xcode that makes use of the Agentry Client framework for iOS. Creating this project is necessary to make use of the OpenUI SDK as well as to rebrand the Agentry Client for iOS. This project is created on a Mac system with the Xcode IDE version 4.6 installed.

Note: This procedure provides the manual steps to create this project. In most use cases this is not necessary and the project provided with the `SMPAgentryClientFramework-iOS-7.0.x.x.tgz` file can be used. This file is located in the directory `./Samples/SMPAgentryClientFrameworkSetup` after the TGZ file has been expanded.

1. Expand the framework archive `SMPAgentryClientFramework-iOS-7.0.x.x.tgz` into a directory on the system. In these instructions this base directory is represented by the value `<Framework_BaseDir>`.
2. Within Xcode, create a new project, selecting iOS Application | Empty Application.
3. In the Build Settings for the main target of the project, set the Framework Search Path to: `<Framework_BaseDir>/SMPAgentryClientFramework/iOS/**`
4. In the Build Settings for the main target of the project, set the Header Search Path to: `<Framework_BaseDir>/SMPAgentryClientFramework/iOS/SMPAgentryClient.framework/Headers/**`
5. In the Build Settings for the main target of the project, set the Other Linker Flags to: `-ObjC -framework -SMPAgentryClient`
6. In the Build Phases section for the main target of the project, add the following resources to the Link With Dynamic Libraries section:

- `libc++.dylib`
- `libiconv.dylib`
- `libcucore.dylib`
- `libsqlite3.dylib`
- `libxml2.dylib`
- `AudioToolbox.framework`
- `AVFoundation.framework`
- `CFNetwork.framework`
- `CoreLocation.framework`
- `CoreMedia.framework`
- `CoreText.framework`

- `CoreVideo.framework`
 - `QuartzCore.framework`
 - `Security.framework`
 - `CoreGraphics.framework`
 - `Foundation.framework`
 - `UIKit.framework`
7. In the Build Settings for the Xcode project set the Strip Linked Product option to No.
 8. Add the resource bundle to the project by selecting the menu item File | Add Files to Project and add the location: `<Framework_BaseDir>/SMPAgentryClientFramework/iOS/SMPAgentryClient.framework/Resources/SMPAgentryClientResource.bundle`
 9. Modify the main.m file within the project by replacing the app delegate class name and return statement:
 - New app delegate class name: `@“SMPAgentryClientAppDelegate`
 - New return statement: `return UIApplicationMain(argc, argv, nil, @"SMPAgentryClientAppDelegate");`
 10. After building and launching the project, the standard Agentry Client should run in either the simulator or on the client device.

With the completion of this procedure a project has been created in Xcode using the resources provided in the Agentry Client framework for iOS.

Next

You can begin developing custom controls using the OpenUI SDK and/or rebranding the Agentry Client for iOS devices by modifying the resource bundle.

Agentry OpenUI SDK for Android Setup Overview

The Agentry OpenUI for Android clients is provided within the SAP Mobile Platform SDK. It is provided in a single ZIP archive, referred to generally as the Agentry Client framework, and can be used to create a development project within a Java IDE. Eclipse was used to create these resources by SAP. It is provided in the Java programming language. Also included in this framework is a sample project with basic overrides of each detail screen field edit type provided as examples.

In order to create the project, it is necessary to have the following tools installed

- Eclipse
- Android SDK - API versions 10, 15, and 17
- Java SDK
- ANT *If building with ANT - This is provided as a part of the Eclipse installation, but may also be installed separately to support rebranding-only builds*

In order to set up an Android development environment, there are different options available and the proper option is up to the developer and the environment currently in place. Both include the usage of the Android Development Toolkit (ADT), an Eclipse plug-in. A summary of each option is provided below, and step by step instructions are available for each in subsequent sections, which include URL's to the Android developer site where both the downloads and installation instructions are provided.

Once the Android development environment is created, the projects provided in the Agentry Client framework archive for OpenUI can be directly imported into the Eclipse workspace. Once this process is complete, you can begin developing the custom controls needed for your mobile application. Instructions are also provided on creating the projects within the Eclipse workspace and information on items such as build order and similar details are included.

Setup for Android Development - Installing the ADT Bundle

To create the Android development environment you can download and install the Android Development Toolkit (ADT) Bundle. This bundle includes both the Eclipse IDE as well as the ADT plug-in for Eclipse already bundled together and configured. As a part of the ADT bundle the Android SDK is included and installed. After installing this bundle it is necessary to use the SDK Manager within the Android SDK to verify and/or install the required API versions of the SDK.

Setup for Android Development - Installing the ADT Plug-in Manually

If you have an existing installation of Eclipse, it is possible to install the ADT plug-in to it, rather than installing the full Eclipse IDE and plug-in. The primary difference in this process from the full ADT bundle option is that you will need to manually configure the preferences within Eclipse for the ADT after it is installed to point it your Android SDK installation.

Setup for Android Rebranding Only

If you will only be rebranding the Agentry Client for Android and will not be developing customer controls using the OpenUI SDK, you do not need to setup an Android development environment. Rather, you can simply install the Ant build scripting tool and make changes to certain files extracted from the framework archive. You can then run an Ant build command to create a rebranded and/or resigned Agentry Client for Android.

Framework Contents

The resources for the OpenUI SDK are provided in the Agentry Client framework, which is found in the file `SMPAgentryClientFramework-Android-7.0.x.x.zip`. This archive and the resources it contains includes all functionality of the Agentry Client for Android and also includes the OpenUI SDK to allow for the detail screen fields to be overridden with custom controls; as well as exposing the necessary resources to rebrand the Agentry Client.

The framework contains the following directory structure, relative to the base directory into which it is extracted:

```
./Sample/
```

```
OpenUI-app.agz
```

Agentry application project corresponding to the Java sample project

```
./com/
```

Beginning of the folder structure in which the sample project and extensions are contained.

```
./SAP/Mobile/Platform/android/AgentryAndroidClientSolution
```

Location of the project containing the buildable resources that produce the standard Agentry Client for Android

```
./vendor/
```

Installing the ADT Bundle With Eclipse

Prerequisites

The following items must be addressed prior to performing this procedure:

- This procedure includes both an Eclipse installation as well as updates to the Android SDK, both of which are a part of the ADT bundle. Instructions for this installation are provided on the Android developer site, as referenced in the main procedure. Access to the bundle itself as well as to the instructions requires internet access.
- The Java Development Kit installer should be downloaded and available. Information and the installer can be found by [Clicking Here](http://www.oracle.com/technetwork/java/javase/downloads/index.html). Alternately, you can enter the following URL into your web browser: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Task

This procedure provides information on installing the Android Development Toolkit (ADT) Bundle. This bundle includes both Eclipse and the ADT plug-in, as well as a basic installation of the Android SDK, all of which are necessary components for development of custom controls using the SAP Agentry OpenUI.

Additionally, it is necessary to install the Java SDK version 1.7, and to configure the system upon which it is installed by making changes to the Environment variables.

1. Install the Java Development Kit (JDK). Note the installation location, as this will be needed later in this procedure.
2. Create a new Environment Variable for Windows by right clicking on My Computer, selecting Advanced system settings, and then clicking Environment Variables. Create a System Variable named JAVA_HOME and set its value equal to the location in which the JDK was installed in the previous step.
3. Next the Android Development Toolkit Bundle must be downloaded and installed. Instructions for this procedure are provided on the Android developer site. In a web browser, navigate to the download page for the ADT Bundle by [Clicking Here](http://developer.android.com/sdk/installing/bundle.html). Alternately enter the following URL in your web browser: `http://developer.android.com/sdk/installing/bundle.html`
4. After installing the ADT Bundle, it is necessary to update the Android SDK that was a part of the installation. Open the folder under the Eclipse directory create in the previous step containing the Android SDK and start the SDK Manager. Here, add the required API versions from the SDK, which include versions 10, 15, and 17. For information on the SDK Manager, [Click Here](http://developer.android.com/sdk/installing/adding-packages.html). Alternately, enter the following URL in your web browser: `http://developer.android.com/sdk/installing/adding-packages.html`

With the completion of this procedure, the Android development environment needed for working with the OpenUI SDK for Android is in place.

Next

See the procedure “SAP Mobile Platform Agentry OpenUI for Android Project Setup” for instructions on creating and importing the projects within the OpenUI SDK.

Installing the ADT Plug-in to an Existing Eclipse Instance

Prerequisites

The following items must be addressed prior to performing this procedure:

- An existing Eclipse installation must already exist. If you wish to install both the ADT plug-in and Eclipse, see the procedure “Installing the ADT Bundle With Eclipse.”
- The Java Development Kit 7 (a.k.a. JDK version 1.7) installer should be downloaded and available. Information and the installer can be found by [Clicking Here](http://www.oracle.com/technetwork/java/javase/downloads/index.html). Alternately, you can enter the following URL into your web browser: `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

Task

This procedure provides information on installing the Android Development Toolkit (ADT) Plug-in to an existing Eclipse instance. This is a necessary component for development of custom controls using the SAP Agentry OpenUI.

Additionally, it is necessary to install the Java SDK version 1.7, and to configure the system upon which it is installed by making changes to the Environment variables.

1. Install the Java Development Kit (JDK) version 1.7. Note the installation location, as this will be needed later in this procedure.
2. Create a new Environment Variable for Windows by right clicking on My Computer, selecting Advanced system settings, and then clicking Environment Variables. Create a System Variable named JAVA_HOME and set its value equal to the location in which the JDK was installed in the previous step.
3. Next the ADT Plug-in can be installed to the Eclipse instance. This process can be handled either through Eclipse itself, or by downloading the PPlug-in and then installing to Eclipse locally. Instructions for this procedure are provided on the Android developer site. In a web browser, navigate to the page for the ADT Plug-in by [Clicking Here](http://developer.android.com/sdk/installing/installing-adt.html). Alternately, enter the following URL into your web browser: `http://developer.android.com/sdk/installing/installing-adt.html` This page contains information on both installing and configuring the plug-in within Eclipse.
4. After installing the ADT plug-in, it is necessary to update the Android SDK that was a part of the installation. Open the folder under the Eclipse directory create in the previous step containing the Android SDK and start the SDK Manager. Here, add the required API versions from the SDK, which include versions 10, 15, and 17. For information on the SDK Manager, [Click Here](http://developer.android.com/sdk/installing/adding-packages.html). Alternately, enter the following URL in your web browser: `http://developer.android.com/sdk/installing/adding-packages.html`

With the completion of this procedure, the Android development environment needed for working with the OpenUI SDK for Android is in place.

Next

See the procedure “SAP Mobile Platform Agency OpenUI for Android Project Setup” for instructions on creating and importing the projects within the OpenUI SDK.

SAP Mobile Platform Agency OpenUI for Android Project Setup

Prerequisites

The following items must be addressed prior to performing this procedure:

- The SAP Mobile Platform SDK must be installed, with the Agency Toolkit being one of the selected components during installation.
- An Android development environment must have been created, per the instructions provided. This includes Eclipse with the ADT plug-in installed, as well as the installation of the Android SDK with the proper API versions installed.

Task

This procedure describes the necessary steps to setup a development environment within Eclipse that can be used for development of customer controls using the SAP Agentry OpenUI SDK for Android clients, as well as for rebranding of the Agentry Client for Android.

When this procedure is complete, a series of development projects will exist in the Eclipse workspace that will allow you to both development customer controls as well as rebrand the Agentry Client for Android and ultimately produce a new application package (.apk file) for distribution to Android devices.

1. Within the directory where the SAP Mobile Platform SDK was installed, locate the file at `AgentryToolkit\AgentryOpenUISDK\SMPAgentryClientFramework-Android-7.0.x.x.zip`
2. Open the archive and extract the contents to an accessible location.
3. Launch Eclipse if not already running. Within Eclipse, you should have the ADT perspective open and have the desired workspace for the projects to be created selected.
4. Within Eclipse, select the menu item **File | Import | General | Existing Projects into Workspace**.
5. Navigate to the location of the files extracted from the Agentry Client for Android framework. There are three directories here: `Sample` `SAP` `vendor`
6. Select each for import into the Eclipse workspace.

This imports the projects within the OpenUI SDK to allow for custom controls and rebranding of the Agentry Client.

7. Perform a clean of all projects once imported.
8. Once this is complete, you can configure the preferences of the projects. For the project `AgentryAndroidClientSolution` you should specify the following build order in the project preferences:

- `AndroidOpenSource`
- `Codeus`
- `com_actionbarsherlock`
- `com.google.zxing.client.android.CaptureActivity`
- `MAF_Controls`
- `OI Distribution Library`
- `OI File Manager`
- `polidea_treeview`
- `AgentryAndroidClientResources`
- `AgentryAndroidClientSolution`

Once this is complete, the development environment for the Agentry Client for Android has been created. To develop custom controls using the OpenUI SDK, all extension classes should

be created in the `AgentryAndroidClientSolution` project, within the package `com.sap.mobile.platform.client.openui.extensions`

For rebranding and/or resigning of the Agentry Client for Android devices, changes should be made to the resources in the `AgentryAndroidClientResources` project.

Agentry OpenUI for Windows Setup Information

in order to use OpenUI SDK for the Windows clients, the setup is straight forward. All methods, object, and types within the SDK for Windows are exposed via Reflection in the .NET Framework.

To create a project, you must first install the standard .NET Agentry Client for Windows. Note that if installing the client to Program Files, it is necessary to have administrator privileges. Alternately you can install the client to a directory to which you have privileges, such as your user directory.

Within the installation folder for the client, there is the `AgentryClientSDK.dll` assembly. In this same folder, create a sub-folder named “CustomControls.”

Within Visual Studio (2012 or later) create a project targeting the .NET 4.5 Framework. You must add a reference to the `AgentryControls.dll` assembly.

To create your custom control, it must implement the `AgentryClientSDK.ICustomerAgentryControl` interface. The Agentry Client will set the `Data Context` property of your custom control to an object that implements the Agentry ADK interface and that corresponds to the type of control.

Once you’ve built your customer control, copy the assembly into the CustomControls folder of the Agentry Client installation for testing. To deploy, you can bundle the Agentry Client installer to include the custom controls. This is accomplished using the Agentry Client Branding SDK.

Installing the Agentry Test Environment

Prerequisites

Address the following items prior to performing this procedure:

- The SAP Mobile Platform SDK 3.0 installer must already have been run and access to the items it installed must be available from the system to which the Agentry Test Environment is being installed.
- Determine the installation location of the Agentry Test Environment on the intended host system. By default this location is `C:\Agentry\Test Environment`. You can specify a different location during the installation if desired.
- Log into the host system as an administrator, with privileges to install software to the desired location.

Task

This procedure describes the steps necessary to install the Agentry Test Environment software component. This component is used as a client test tool during development of applications using the Agentry archetype. It is not intended for end users. It can only be installed to Windows desktop systems and cannot be installed to mobile devices.

Perform this procedure to:

- Test modifications of the mobile application, when those modifications affect client-side behavior or client-server communications or synchronization.
- Test or debug run time issues in a Production environment. The Agentry Test Environment can be connected to the Agentry Production Server just as a normal client can.

When this procedure is complete, the Agentry Test Environment is available for use to test the client-side behaviors of any modifications made to the mobile application.

1. Start the Agentry Test Environment installer on the host system where you are installing this component by running the program `Agentry_7_0_x_x_TestEnvironment.exe` (NOTE: The value x represents the latest service pack or patch release of the system, depending on the version downloaded). This displays the **Welcome to the Agentry Test Environment Setup Wizard** screen.
2. Click the **[Next]** button to advance the wizard. This displays the **License Agreement** screen.
3. Click the **[Yes]** button to accept this agreement and advance the wizard. The **Choose Install Location** screen is displayed.
4. Enter the desired directory path in the Destination Folder field. This can be a new path, an existing location selected by clicking the **[Browse]** button, or the default location. Click the **[Next]** button to install the Agentry Test Environment to the specified directory. This displays the **Shortcuts for Agentry Test Environment** screen.
5. Select the desired shortcut locations by selecting or deselecting the listed shortcut options. Once the shortcuts are configured, click the **[Install]** button to begin the installation of the **Agentry Test Environment** software component.

The selected shortcuts are created by the installer. The **Installation Status** screen is displayed next indicating the installation progress. When the installation is complete, the **Completing the Agentry Test Environment Setup** wizard screen is displayed.

6. To start the Agentry Test Environment now, leave the box checked on this screen. Otherwise, deselect this box. Click the **[Finish]** button to close the wizard.

With the completion of this procedure, the Agentry Test Environment is installed to the selected location and is available for use in testing mobile applications in development, configuration, implementation, or production issue resolution scenarios.

Next

Just as with a standard client installation, the Agentry Test Environment needs to connect to an Agentry Server to perform an initial transmit. Prior to performing this transmit, the proper client platform to be mimicked, enabling or disabling scan and GPS simulations, and other similar selections should be made.

Agentry SAP Framework Foundation Installation Overview

The Agentry SAP Framework Foundation is implemented as an ABAP Add-On and is installed to the SAP system as a part of the implementation of the SAP Mobile Platform only when building or supporting mobile applications built in the Agentry archetype, and only when those mobile applications synchronize data with SAP systems. If both of these are not a part of your environment, this component is not installed. The framework includes both the Administration and Monitoring Portal, and the Integration Framework.

The Integration Framework provides the integration point to the SAP system for the Agentry Server. This then allows the Agentry Server to synchronize data with the SAP system for the mobile application. Included in the Integration Framework is the Configuration Portal used during implementation to create synchronization components of the mobile application.

The Administration and Monitoring Portal provides the tools and user interface to allow for the administration, configuration, and care and feeding tasks for the Agentry SAP Framework Foundation. These include security settings, log settings, user monitoring, and other similar tools.

There are also two additional resources needed in both the development and runtime environments when using the Agentry SAP Framework Foundation component. The first is an API contained in the Java file `SAPCommon-130881.jar`. The second is a configuration file, which must be created manually, named `JavaBe.ini`. Both of the files must reside within the folder where the Agentry Server has been installed. Additionally, the `SAPCommon-130881.jar` must be an available resource for any Java development projects related to the mobile application being developed.

The installation of the Agentry SAP Framework Foundation includes the following main tasks:

1. Verify the system requirements.
2. Install the Agentry SAP Framework Foundation software and support packages and place the `SAPCommon-130881.jar` file in the proper Agentry Server directory.
3. Create the `JavaBe.ini` configuration file and place it in the Agentry Server's installation folder.

Agentry SAP Framework Foundation Component

The following table lists the minimum version of the items required for the Agentry SAP Framework Foundation Component installation to the SAP system system. The software component for Agentry SAP Framework Foundation is SMFND release 600_700.

Software Requirements

Software Component	Release	Service Pack
SAP_ABA	700	SP14
SAP_BASIS	700	SP14

Installing Agentry SAP Framework Foundation

Prerequisites

The following items must be addressed prior to performing this procedure:

- Ensure you have the current versions of kernel, TP and R3trans.
- Current SPAM / SAINT Update - Compare the short text of the last SPAM / SAINT update you imported with that of the SPAM / SAINT Update in the SAP Service Marketplace. If the version of the SPAM / SAINT Update in the SAP Service Marketplace is more recent, import it.
- Verify the required software components have been installed, as listed in the system requirements for the Agentry SAP Framework Foundation.
- Verify the required support packages have been installed, as listed in the system requirements for the Agentry SAP Framework Foundation.
- No SAP password is required.

Task

Additional Information

- Space required in the transport directory: approximately 10 MB
- Total runtime: approximately 0.5 hour
- **SAP ABAP Add-Ons:**
 - Integration Foundation Add-On:
 - SMFND_600700_NW700.SAR

Language Support

The Agentry SAP Framework Foundation supports the following languages:

- English
- German
- Brazilian Portuguese
- French
- Spanish
- Korean

- Japanese
- Simplified Chinese

1. Log on to your SAP system as client **000** and as a user that has system administrative privileges. Do **not** use the SAP* or DDIC users.
2. Import the required user language(s) for all components that have already been installed. You must perform this language import prior to the installation of the Agentry SAP Framework Foundation.
3. Load the software package into your system via the Add-On Manager, using the transaction code **SAINT**.

For more information about this, see the online documentation for the Add-On Installation Tool. Select the help function in the application toolbar and navigate to **Online Documentation | Loading Installation Packages**.

4. Start the installation of the Agentry SAP Framework Foundation using the Add-On Installation Tool, accessed from the transaction SAINT.

For more information about this, see the online documentation for the Add-On Installation Tool, selecting the help function on the toolbar.

5. Activate the Services for the Web Dynpro ABAP Applications

- a) Start transaction SICF.
- b) Activate all services under the node `default_host/sap/bc/webdynpro/syclo`

6. Define Intervals for Number Range Objects /SYCLO/C_1 and /SYCLO/C_2

- a) Start transaction SNRO.
- b) Enter the number range object /SYCLO/C_1.
- c) Select **Number Ranges | Intervals**.
- d) Maintain or create the intervals 01 and 02 .
- e) Repeat these steps for the number range object /SYCLO/C_2 and maintain the intervals 01, 02, 03, 04, and 05. For example:

Maintain Number Range Intervals

Interval

NR Object: /SYCLO/CORE Counter

No.	From number	To number	Current number	Ext
01	0000000001	0099999999	0	<input type="checkbox"/>
02	0100000000	0199999999	0	<input type="checkbox"/>
03	0200000000	0299999999	0	<input type="checkbox"/>
04	0300000000	0399999999	0	<input type="checkbox"/>
05	0400000000	0499999999	0	<input type="checkbox"/>

- Next, copy the file `SAPCommon-130881.jar` from the SAP Mobile SDK package to the installation location of the Agency Server.

With the completion of this procedure the Agency SAP Framework Foundation has been installed to the SAP system and is available for developers to begin creating the necessary components for a mobile application built using the Agency archetype.

Next

The final item to address is the creation of a configuration file for the Agency Server. See the instructions “Creating the JavaBE.ini File for SAP Systems” for details.

Creating the JavaBE.ini File for SAP Systems

Prerequisites

The following items must be addressed prior to performing this procedure:

- The `JavaBE.ini` file is needed only when using the Agency SAP Framework Foundation.
- This file is needed by the Agency Server. This component should be installed, per instructions found in the SAP Mobile SDK installation guide.

- This file should NOT be manually placed within the SAP Mobile Platform Runtime environment. Rather it should be packaged and deployed per documented procedures for deploying Agentry applications.

Task

This procedure documents the steps necessary to create the JavaBE.ini configuration file for use by the Agentry Server. The SAPCommon-130881.jar processes this file at runtime and expects certain sections and settings to be present. These revolve primarily around connectivity and authentication to and with the SAP System.

In addition to these required sections, in some implementations developers choose to add additional configuration sections. These additional sections are valid, but are outside the scope of this document. Typically these settings are consumed by the synchronization logic built for a specific application.

1. Create a new plain text file named JavaBE.ini and save it in the installation folder of the Agentry Server.
2. Using a plain text editor, open the JavaBE.ini file for editing.
3. Create the [HOST] section with the following settings:

```
[HOST]
```

```
server=<your sap host system's network name>
```

```
APPNAME=<name of your mobile application>
```

4. Create the [CONFIG] section exactly as shown below. Note there are other options for the source setting listed in this section, but the default is sufficient for initial configuration. Further information on this is provided in development guides.

```
[CONFIG]
```

```
source=SAP
```

5. Create the [CLIENT_NUM] section with the following setting:

```
[CLIENT_NUM]
```

```
CLIENT=<client number for Agentry Server to communicate with SAP  
Application server>
```

6. Create the [SYSTEM_NUM] section with the following setting:

```
[SYSTEM_NUM]
```



```
SYSNUM=<system number for Agentry Server to communicate with the
SAP Application Server>
```

7. Create the [LANGUAGE] section with the LANG option set to the two character SAP language key matching the language of the SAP system:

```
[LANGUAGE]
```

```
LANG=<two chracter SAP language key>
```

8. Create the [LOGGING] section, with the logging level set based on the information provided after the example:

```
[LOGGING]
```

```
Level=<logging level>
```

- 1 - Fatal Errors only
- 2 - Errors and above
- 3 - Warnings and above
- 4 - Informational messages and above
- 5 - Debugging messages and above
- 6 - Trace level debugging

9. Create the [LOGON_METHOD] section with the following settings, using the descriptions of settings provided below the example. The option set here will specify whether the [GLOBAL_LOGON] or [GROUP_LOGON] section is needed as well:

```
[LOGON_METHOD]
```

```
LOGON_METHOD=<logon method to SAP system>
```

- USER _AUTH - Standard User ID and password authentication is used.
- USER_AUTH_GLOBAL - Pooled connections using a single user ID and password; requires the section [GLOBAL_LOGON] to also be created.
- USER_AUTH_GROUP - User ID and password authentication with the SAP Message Server (load balancing) is used; requires the section [GROUP_LOGON] to also be created.

10. If the LOGON_METHOD is set to USER_AUTH_GLOBAL, create a new section named [GLOBAL_LOGON] as shown next:

```
[GLOBAL_LOGON]
```

```
UID=<User ID shared by all users>
```

```
UPASSWORD=<Password shared by all users>
```

```
SHAREDCONNECTIONS=<number of connections created for the shared pool>
```

- 11. If the LOGON_METHOD is set to USER_AUTH_GROUP, create a new section named [GROUP_LOGON] as shown next:**

```
[GROUP_LOGON]
```

```
MESSAGE_SERVER=<host name or IP of the SAP Message Server>
```

```
GROUP_NAME=<name of the SAP Application Server group>
```

```
SYSTEM_ID=<name or ID of the SAP system>
```

```
CLIENT=<client number to be used by the Agentry Server to connect to the SAP system>
```

- 12. Create the section [REQUIRED_BAPI_WRAPPER] with the settings and values listed exactly as shown next:**

```
[REQUIRED_BAPI_WRAPPER]com.syclo.sap.bapi.LoginCheckBAPI=/SYCLO/  
CORE_SUSR_LOGIN_CHECK
```

```
com.syclo.sap.bapi.RemoteUserCreateBAPI=/SYCLO/  
CORE_MDW_SESSION1_CRT
```

```
com.syclo.sap.bapi.RemoteParameterGetBAPI=/SYCLO/  
CORE_MDW_PARAMETER_GET
```

```
com.syclo.sap.bapi.SystemInfoBAPI=/SYCLO/CORE_SYSTINFO_GET
```

```
com.syclo.sap.bapi.ChangePasswordBAPI=/SYCLO/  
CORE_SUSR_CHANGE_PASSWD
```

```
com.syclo.sap.bapi.CTConfirmationBAPI=/SYCLO/  
CORE_OUTB_MSG_STAT_UPD
```

```
com.syclo.sap.bapi.DTBAPI=/SYCLO/CORE_DT_GET
```

```
com.syclo.sap.bapi.GetEmployeeDataBAPI=/SYCLO/  
HR_EMPLOYEE_DATA_GET
```

```
com.synclo.sap.bapi.GetUserDetailBAPI=/SYCLO/CORE_USER_GET_DETAIL
```

```
com.synclo.sap.bapi.GetUserProfileDataBAPI=/SYCLO/  
CORE_USER_PROFILE_GET
```

```
com.synclo.sap.bapi.PushStatusUpdateBAPI=/SYCLO/CORE_PUSH_STAT_UPD
```

```
com.synclo.sap.bapi.RemoteObjectCreateBAPI=/SYCLO/  
CORE_MDW_USR_OBJ_CRT
```

```
com.synclo.sap.bapi.RemoteObjectDeleteBAPI=/SYCLO/  
CORE_MDW_USR_OBJ_DEL
```

```
com.synclo.sap.bapi.RemoteObjectGetBAPI=/SYCLO/  
CORE_MDW_SESSION_GET
```

```
com.synclo.sap.bapi.RemoteObjectUpdateBAPI=/SYCLO/  
CORE_MDW_SESSION_UPD
```

```
com.synclo.sap.bapi.RemoteReferenceCreateBAPI=/SYCLO/  
CORE_MDW_USR_KEYMAP_CRT
```

```
com.synclo.sap.bapi.RemoteReferenceDeleteBAPI=/SYCLO/  
CORE_MDW_USR_KEYMAP_DEL
```

```
com.synclo.sap.bapi.RemoteReferenceGetBAPI=/SYCLO/  
CORE_MDW_SESSION_GET
```

```
com.synclo.sap.bapi.RemoteReferenceUpdateBAPI=/SYCLO/  
CORE_MDW_SESSION_UPD
```

```
com.synclo.sap.bapi.RemoteSessionDeleteBAPI=/SYCLO/  
CORE_MDW_SESSION1_DEL
```

```
com.synclo.sap.bapi.RemoteUserDeleteBAPI=/SYCLO/  
CORE_MDW_SESSION1_DEL
```

```
com.synclo.sap.bapi.RemoteUserUpdateBAPI=/SYCLO/  
CORE_MDW_SESSION_UPD
```

```
com.synclo.sap.bapi.TransactionCommitBAPI=WFD_TRANSACTION_COMMIT
```

```
com.syclo.sap.bapi.SignatureCaptureBAPI=/SYCLO/  
CS_DOBDSDOCUMENT_CRT
```

13. Save and close the file to the installation location of the Agentry Server.

With the completion of this procedure the `JavaBe.ini` configuration file for the Agentry Server is created, with settings to connect to the SAP system for which the mobile application will be developed.

Installing the Agentry SDK

Prerequisites

The following items must be addressed prior to performing this procedure:

- This SDK is provided for development purposes and it is assumed the person installing these components understands the development of ActiveX controls, and/or inter process communications in Windows Mobile environments. If such items are not needed for your implementation, then this SDK need not be installed and is not needed for any other mobile application development using the Agentry Editor or other components of the SAP Mobile SDK.
- This SDK should be installed to the location where needed for development work; i.e. onto the same host system as the IDE or other development toolset to be used for development; or to a file share or network drive accessible to these tools.

Note: The Agentry ActiveX API should be considered a deprecated resource. It is provided for backwards compatibility and will not be functionally changed in the future. Existing implementations making use of ActiveX can be migrated to the latest SAP Mobile Platform version without modification to the ActiveX controls in use. For new implementations in which custom controls are to be created, see the information provided on the OpenUI API for the Agentry Client.

Task

This procedure provides the steps for installing the Agentry SDK, which can be found in the SAP Mobile SDK package provided on the SAP Service Marketplace. This is a simple installation procedure, consisting of executing the installation program and selecting the location for the files within the Agentry SDK.

Once the SDK has been installed, the resulting files are found in two distinct folders within the base installation location you will select:

- Agentry ActiveX
- Agentry External Process

For details on the contents of this SDK, and how to develop applications or controls using this SDK, see the *Developer Guide: Agentry Applications*, specifically the information beginning in the section “The Agentry SDK,” and the subsequent sections “Technical Overview -

ActiveX Controls and the Agentry Client” and “Agentry Client API for External Processes Technical Overview.”

1. Launch the installer program by executing `Agentry_7.0.x.x_SDK.exe`. (NOTE: ‘x’ will vary depending on the service pack version currently available).
2. Click the **Next** button on the first screen **Welcome to the Agentry Software Development Kit Setup Wizard**.
3. Click the **Yes** button to accept the license agreement presented on the **License Agreement** screen.
4. Next specify the location where you would like the resources of the SDK to be installed in the **Choose Install Location** screen.
5. Click **Install** to install the Agentry SDK. Finish the wizard when prompted by the last screen,

With the completion of this procedure the resources of the Agentry SDK have been installed. These resources can be moved or copied to other locations as needed and can be included in development projects.

Next

If you have not done so already, review the information provided on the Agentry SDK found in the *Developer Guide: Agentry Applications*.

Developing Agentry Apps

Use the Agentry Toolkit to develop metadata-driven applications.

Eclipse Preferences for the Agentry Editor Plug-In

With the release of version 5.2 of the Agentry Mobile Platform, additions have been made to the Eclipse preference pages by the Agentry Editor plug-in. These preferences affect the behavior of the Editor as a whole, or the appearance and behavior of the various views within the Agentry Perspective.

The following sections detail each of these new preference pages. To access these pages, select the menu item **Window | Preferences** in Eclipse. In the Preference screen, select the **Agentry** root node on the left side of the screen.

Compare Preferences

The Compare Preferences page displays the preferences for the Comparison View’s appearance. the settings provided allow for the personalization of the colors used to denote the definitions when in different states within this view. There is also a preference for the number of previous revisions to display when the local project is connected to a share repository.

For each of the color preferences, the foreground and background color can be set by selecting the desired color from a color palette. The foreground color setting specifies the text color and

the background color specifies the field behind the item. The following is a list of each of these preferences:

- **Selected Definitions:** This preference specifies the color scheme for the currently selected definition in the Comparison View.
- **Matching Definitions:** This preference specifies the color scheme for definitions when the two definitions match exactly in both projects being compared in the Comparison View.
- **Non-Matching Definitions:** This preference specifies the color scheme for definitions that do not match between the two projects being compared in the Comparison View.
- **Non-Matching (Unimportant) Definitions:** This preference specifies the color scheme for definitions that do not match between the two projects being compared in the Comparison View when the difference between the two is an unimportant difference, such as a difference in the descriptions of each.
- **Source-Only and Editor-Only Definitions:** This preference specifies the color scheme for definitions that exist in only one of the projects being compared in the Comparison View.
- **External Compare Tool:** This preference allows for the selection of the external comparison tool used to compare two text files from the Comparison View. By default the tool Windiff is used. However, another tool can be used provided it can accept two arguments, each representing the files to be compared. This preference should be set to the full path and file name of the executable file of the comparison tool.

Compression Settings

The Compression Settings page displays the preference settings for compressing Agentry definition files during export, export differences, and publish operations. The default is to compress the definition files generated during these operations.

- **Compress Export:** This preference, when selected, results in the creation of compressed export files when the Agentry application project is exported. The file extension for the export file is set to `.agxz`. If the file is not compressed the resulting export file has a file extension of `.agx`.
- **Compress Export Differences:** This preference, when selected, results in the creation of compressed export files when the Agentry application project is exported via an Export Differences operation. The file extension for the export file is set to `.agxz`. If the file is not compressed the resulting export file has a file extension of `.agx`.
- **Compress Publishes:** This preference, when selected, results in the published definitions, stored on the Agentry Server, being compressed. The Server recognizes this compression and automatically decompresses it when loading the definitions. If this option is not selected, definitions are not compressed when published to the Server and the Server will recognize this as well. No change is needed on the Agentry Server to reflect the compression setting for publish.

New Definitions

The New Definitions preference page contains preferences related to the creation of new definitions within the application project. This includes setting reminders on definitions,

default names for certain definition types, and the default selected data type for complex table fields. These settings only specify default values initially selected and all values can be changed when actually creating the definitions referenced here. The following list describes each of the preferences found on this page:

- **New Definition Reminders:** Each definition type within the project supports reminders, or text values displayed as informational messages in the Problems View. By default reminders are set manually by the developer. Using this preference setting it is possible to set a default reminder message for all new definitions created in the project, including a default reminder message.
- **Transaction Naming:** The default name of all new transactions can be specified to be either Noun-Verb (default) or Verb-Noun. Regardless of the default set here, names can always be changed when adding a new transaction definition.
- **Rule Naming:** The default name of all new rule definitions can be specified to be either Name-Usage (default) or Usage-Name. Usage refers to the attribute type referencing the rule, e.g. Enable Rule, Initial Value Rule, etc. Regardless of the default set here, names can always be changed when adding a new rule definition.
- **Complex Table Field Type:** This preference allows you to specify the default data type selected when creating a new complex table field definition. This is only the default selection when a new field is first created and can be changed during the creation process to a different data type if needed.

Perspectives

This section specifies the behavior of Eclipse related to the Agentry perspectives when opening an Agentry application project when there is no Agentry perspective currently open. These preferences specify whether to always open an Agentry perspective, never open one, or to prompt you to choose whether or not to open the perspective. Included in these preferences is a list of the available Agentry perspectives where the specific one to open by default is selected.

Project Explorer

The Project Explorer preferences page contains the preferences that affect the appearance and behavior of the Project Explorer View, as well as the Properties View interaction with it. The following list describes each of the preferences found on this page:

- **Link Properties View breadcrumb navigations with the Project Explorer View:** When this preference is set to true, navigating in the Properties View using the navigation buttons for back and forward will automatically select the definitions in the Project Explorer View.
- **Link Properties View parent navigations with the Project Explorer View:** When this preference is set to true, using the parent arrow in the Properties View to navigate to the parent or ancestor definition of the one currently displayed will automatically select that definition in the Project Explorer.

- **Link Properties View hyperlink navigations with the Project Explorer View:** When this preference is set to true, clicking a hyperlink label in the Properties View to display the referenced definition will automatically select that definition in the Project Explorer View.
- **Link Trash Bin definition restores with the Project Explorer View:** When this preference is set to true, the restoration of a definition from the Trash Bin View will automatically select that definition in the Project Explorer View.
- **Link Visual Screen Editor selections with the Project Explorer View:** When this preference is set to true, the selection of a screen control (detail screen field, list column, button, etc.) in the Layout View of a screen definition will result in that same definition being selected in the Project Explorer View.
- **Sort Order:** This preference specifies how the definition type nodes in the Project Explorer View should be sorted. The options are Traditional, which sorts the type nodes as in releases of Agentry prior to 5.2; and Ascending, which sorts the type nodes in alphabetical order.

Tagging Configuration

The Tagging Configuration preferences page contains the preferences for the definition tagging behavior of the project. It is also possible to add new tags to the project within this preference page. Note the preferences on this page are specific to the currently open Agentry application project. The following list describes the preferences displayed on this page:

- **Show tags bar in the Properties View:** This option shows or hides the tags bar in the Properties View. If the current open Agentry application project is connected to a share, this preference is always true and cannot be changed.
- **Auto-Tagging (Public):** Within this section each of the public tags for the project are listed, represented as a button. One or more of these buttons can be selected to specify that tag should be applied to any definition changed, added, or deleted in the current project. Clicking the tag configuration button within this section will display the tags dialog where additional tags can be added to the project.

Team Configuration

The Team Configuration preference page contains preferences for the team configuration behavior. These preferences are specific to each Agentry application project. These preferences are not available when the current project is not connected to a share. The following list describes these preferences:

- **Auto-Tagging (Private):** Private auto-tagging describes the behavior where the Editor automatically applies a private tag to any definition when it is modified, added, or deleted in the current project. This tag is used in comparisons with the share revisions, and is also displayed in certain operations to allow for the selection of definitions with this tag. The preference setting on the Team Configuration preference page allows for a customized value to be used as the name of this tag. By default, if this preference is blank, the developer's Windows user ID is the tag name.
- **External File Handling:** This preference specifies whether it should be required that the Agentry Development Server is specified before any share operations are executed that

involve definitions with an external script normally stored on the Development Server. As examples, when this preference is true, a new project cannot be imported from a share without specifying the Development Server for the new project; a SQL step cannot be updated from the share unless the Development Server has been specified.

- **Share Revisions Menu Count:** This preference specifies the maximum number of revisions to display at one time in the revision menu of the Comparison View. This preference setting only affects the Comparison View's behavior when the open Agentry application project is connected to a share repository.

Agentry Editor and Eclipse Platform Overview

The primary development tool for an Agentry mobile application is the Agentry Editor. The Editor is provided as a plug-in component to the Eclipse development platform. Eclipse is an open source project freely available for download from the project's website, and can also be installed with the the Agentry Editor plug-in. The Agentry Editor is a 4th generation language (4GL) development tool providing point-and-click development of applications. Much of the behavior of the Editor is dictated by the overall behavior of the Eclipse platform.

The following is general information covering the Eclipse platform and the Agentry Perspective within Eclipse. Developers unfamiliar with Eclipse will find this information useful as a starting point. It is also recommended that developers review the help information provided with Eclipse by selecting **Help | Help Contents** and reviewing the *Workbench User Guide* as well as other information provided in the help content.

Specifics on procedures, navigation, and other topics related specifically to the Agentry Editor are covered in detail in subsequent sections.

The Agentry Editor Perspective and Views

The Agentry Editor plug-in includes an Eclipse perspective consisting of several Views. A view is a pane or tab within Eclipse that presents information about and access to various development components. Perspectives are a collection of multiple views. Opening a perspective within Eclipse opens all of that perspective's views.

The views within the Agentry perspective provided by the Agentry Editor plug-in are the primary views for application development within Agentry, but they are not the only views that will be used in most projects. Other views and other perspectives within Eclipse provide features and functionality that match the tasks and development work performed for a given application project. The ability to use the core views within the Agentry perspective in conjunction with views from other tools and plug-ins within Eclipse is one of the main advantages to the Agentry Editor plug-in architecture.

A common perspective used during the development of a mobile application is the Java perspective. This perspective is provided with Eclipse and includes various views and other tools to support Java development work. When working with a mobile application that will synchronize data with a back end system using Java, this perspective is used regularly for development, implementation and configuration tasks.

Agentry Application Project

Development work related to a mobile application's Agentry definitions is organized within the Agentry Editor in an Agentry Application Project. This project contains the definitions that are the mobile application, structured according to the Agentry Application Hierarchy. Within the Agentry Editor perspective this project is presented in the Project Explorer View according to the application hierarchy. Within this view the developer navigates through the application definitions. The project itself is stored as an Agentry application project file with the extension `.apj`.

Eclipse Workspace

The Eclipse platform organizes development projects, interface settings, and most other aspects of the environment into workspaces. This includes the Agentry application project created and managed within the Agentry Editor plug-in. Other related projects are also stored within the same workspace. In most cases it is recommended that a workspace exists for each mobile application. Within the workspace for a given application there should then reside the Agentry application project, as well as other related items such as Java or Ant projects, where necessary.

An alternative is to create a single workspace for multiple mobile applications that are developed for the same back end system and that have significant overlap in back end processing requirements and methodology. In environments where multiple applications are developed for the same back end system, developers may structure the back end synchronization components into layers that include those common to all mobile applications for the same back end, and those specific to a given mobile application. In such environments it may make sense to have a single workspace containing all components and projects, so that the individual application projects can share the common synchronization resources, as well as contain the individual synchronization projects for each application. This would also include one Agentry application project for each mobile application.

In general, the structure of the workspace is up to the developer or development team, and the best approach for all involved can be used. There is no hard and fast rule concerning the proper structure for a workspace within Eclipse as it relates to the mobile applications developed or modified in the Agentry Editor.

Other information concerning the Eclipse interface is also stored within the workspace. These include the items set via Eclipse's preference settings. These settings can affect the behavior of numerous different views within Eclipse. Changes made to the preferences will be stored in the workspace. This means each workspace within Eclipse can have its own preferences tailored to the needs of that workspace, and switching from one to another will load the new workspace's settings.

Third Party Views and Tools

In addition to the functionality contained within the Agentry Editor plug-in, there are additional tools and views provided by third parties as contributions to the Eclipse Platform.

Many of these may be useful in different development environments that involve Agentry application projects. Certain contributions are expected and used by the Agentry Editor, specifically the Data Tools Platform project and the Java Perspective. The Java Perspective is provided with all implementations of Eclipse and is used by the Agentry Editor when the application project includes synchronization with a Java Virtual Machine system connection.

The Data Tools Platform is included in Eclipse as well and is used when the application project includes synchronization with a SQL Database system connection. It is through this package that the Agentry Connector Studio provides its functionality.

The Agentry Perspective in Eclipse

The various views within the Agentry Perspective in Eclipse work in conjunction with one another. The navigation of an application project involves selecting a definition in one perspective and then viewing it in one or more of the other perspectives.

The following are the views found in the Agentry Perspectives provided with the Agentry Editor plug-in for Eclipse:

- Project Explorer
- Properties
- Diagram Views
- Dependency
- Trash Bin
- Problems

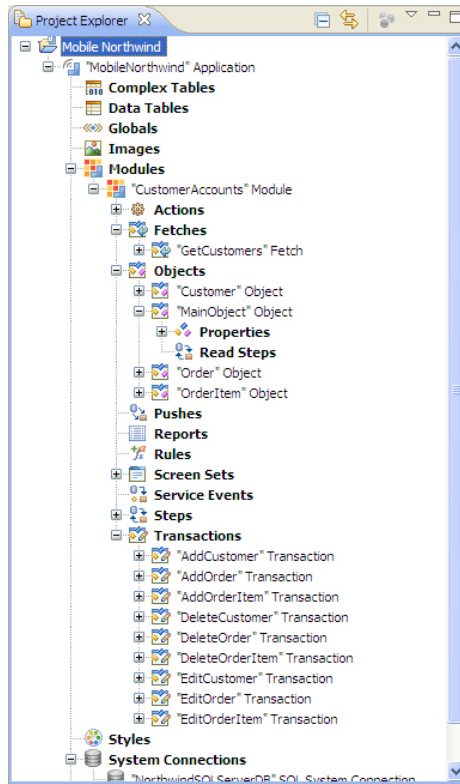
Project Explorer View

The Project Explorer View is a part of the core Eclipse platform that is used by the Agentry Editor plug-in. In general, this view displays projects saved within the current Eclipse workspace. These projects are displayed in a tree control, with each root node representing one of the projects. For the Agentry Editor, the Agentry application projects within the current workspace are listed here as well.

An Agentry project is opened or closed within this view. An Agentry project can be navigated in the project explorer view only when it is open, and only one Agentry application project can be open at a given time. Opening one project will close any Agentry project that is currently opened.

The definitions within an Agentry project are displayed in a tree control that matches the Agentry Application Hierarchy. For the open project, any definition listed in the tree control can be selected. This displays the definition in the Properties View, lists its dependency items in the Dependency View, and may display a diagram for the definition in the Diagram View. This last depends on the type of definition selected, as not all definitions have a corresponding diagram.

The following is an example of the Project Explorer View with a sample Agentry application project open within it:



Each of the nodes in this view represents a definition within the application project. Bold nodes represent a set of definitions of the same type, contained within a parent definition. Selecting a bold node will display a list of all definitions of that type within the same parent definition. Selecting a definition node will display information about that definition in the other views within the Agentry Perspective.

Right-clicking a node within the view displays a context menu with options to affect the application project or the selected definition. Which specific options are available depends on which node is selected, or which definition that node represents. Items that may be displayed include displaying the definition in its diagram view, adding a new definition of the selected type, copying or deleting the selected definition, or setting a reminder for the selected definition.

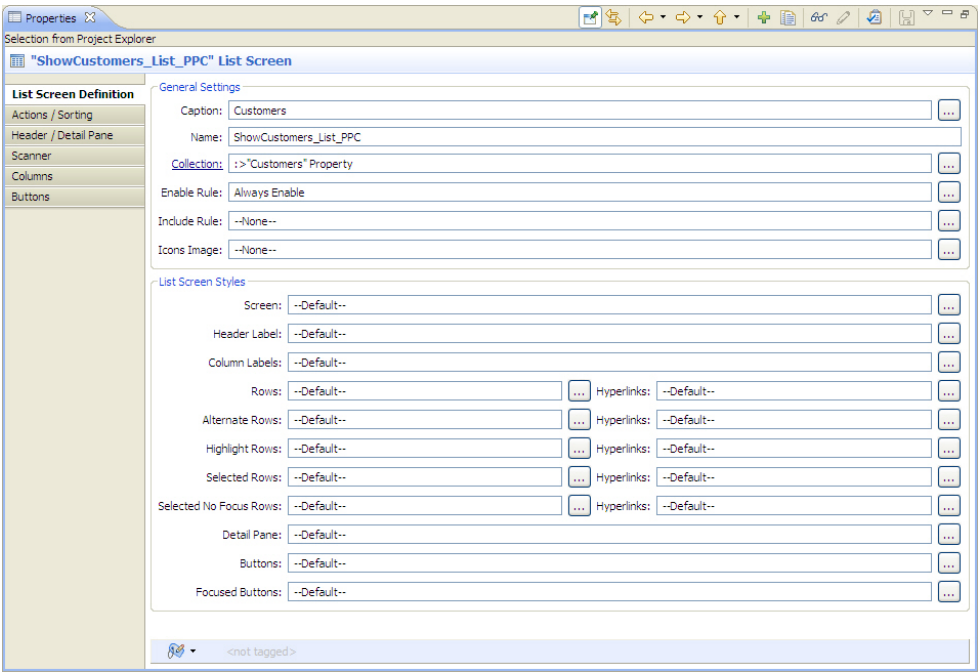
Properties View

The Properties View displays the attributes and child definitions for the currently selected definition within the project. The definition's attributes are edited within this view, and its child definitions can be modified as well, including adding new definitions, deleting a child definition, or selecting a child definition. Selecting a child definition displays it in the Properties View.

Within the Properties view there can be one or more tabs, listed to the left of the view. Each tab displays a set of attributes for the definition, or a list of child definitions. When a tab that displays attributes is selected, there is a single row of toolbar buttons displayed for the view. These buttons pertain to the definition, or to navigation within the application project based on the currently displayed definition.


When the selected tab within the Properties View displays a list of child definitions, a second row of toolbar buttons is displayed that allow for the modification of that list and the child definitions each item in the list represents. This includes adding, editing, deleting, copying, navigation, and other similar options.









The following is an example of the Properties View for a list screen definition, with the List Screen Definition tab selected:



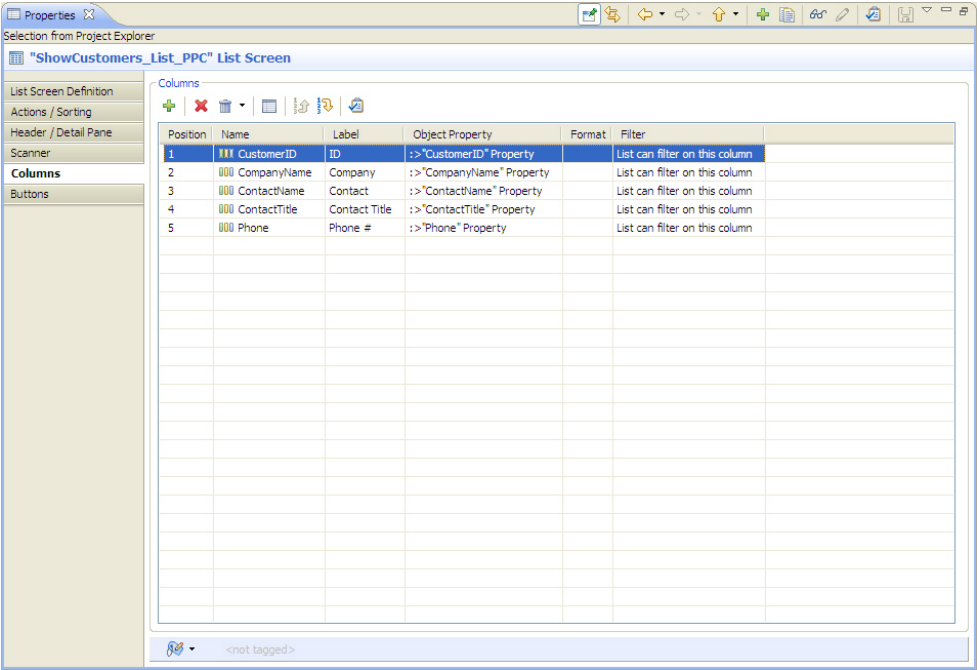
As noted, at the top of this view are the toolbar buttons for working with the definition and its currently displayed attributes. The following is a list of these buttons and their purpose:

Button	Button Icon	Description
--------	-------------	-------------

Pin Properties View		Pins the Properties View so that it always displays the current definition, regardless of any selections made in any other view. Note that this prevents navigating away from this definition until the button is deselected.
---------------------	---	---

Button	Button Icon	Description
Link With Project Explorer		Selects and highlights the node for the currently displayed definition in the Project Explorer View.
Navigation Buttons		These buttons provide browser-like navigation of the application project, with back, next, and parent navigation possible.
Add Definition		This button adds a definition of the same type, and to the same parent definition, as the one currently displayed in the Properties View; e.g., if currently viewing a screen, this will add a new screen to the same parent screen set.
Copy Definition		This button creates a new definition that is a copy of the currently displayed definition, including all descendent definitions. The new definition is added to the same parent as the original definition.
Diagram View		This button opens the Diagram View for the currently selected definition. This button is disabled for definitions that do not have associated diagram views.
Edit Definition		This button opens the editor specific to the currently displayed definition in the Properties View. Most definitions do not have a type-specific editor, and their attributes are edited within the Properties View. This button is disabled in these cases. The Rule definition is the only definition for which this button is enabled, displaying the Rule Editor when selected.
Set Reminder		This button sets a reminder for the definition displayed in the Properties View. Reminders are listed in the Tasks View within Eclipse, and also result in informational messages in the Problems View during publish and check on publish operations. All definitions support reminders. Setting a reminder does not affect application behavior and serves only to provide notes to developers within the application project on tasks remaining to be accomplished.
Save		This button saves any changes made to the attribute settings currently displayed in the Properties View. Note that the main Eclipse toolbar also contains a save button. Only the save button in the Properties View saves changes made to the definition's attributes. The Eclipse toolbar button is disabled when the Properties View has the current focus.

Selecting a tab listing child definitions for the current definition in the Properties View displays a second toolbar of buttons that affect the list of child definitions. Following is an example of the Properties View for a list screen with the Columns tab selected, which displays a list of all columns that are child definitions to the list screen:



Note the row of buttons directly above the list of child definitions. Following is a description of each of these buttons:

Button	Button Icon	Description
Add Definition		This button adds a new definition of the type currently listed as a child definition to the one currently displayed in the Properties View.
Delete Definition		This button deletes the currently selected child definition in the list.
Trash Bin		This button displays the all definitions that have been previously deleted from the parent definition of the type currently listed; e.g. all columns deleted from the list screen. From this list a deleted child definition can be recovered from the trash bin and placed back in the parent definition.
Display Definition		This button displays the currently selected child definition in the Properties View.



Button	Button Icon	Description
Change Positions		These buttons move the currently selected definition in the list up or down one position in relation to the other child definitions listed. These buttons are only available in child definition lists where the position of the child definitions affect application behavior; e.g., the position of list screen columns in the list dictates the initial display order of those columns on the client at run time, whereas the order in which object properties are listed has no effect on application behavior.
Set Reminder		This button sets a reminder for the currently selected child definition. Reminders are listed in the Tasks View within Eclipse, and also result in informational messages in the Problems View during publish and check on publish operations. All definitions support reminders. Setting a reminder does not affect application behavior and serves only to provide notes to developers within the application project on tasks remaining to be accomplished.

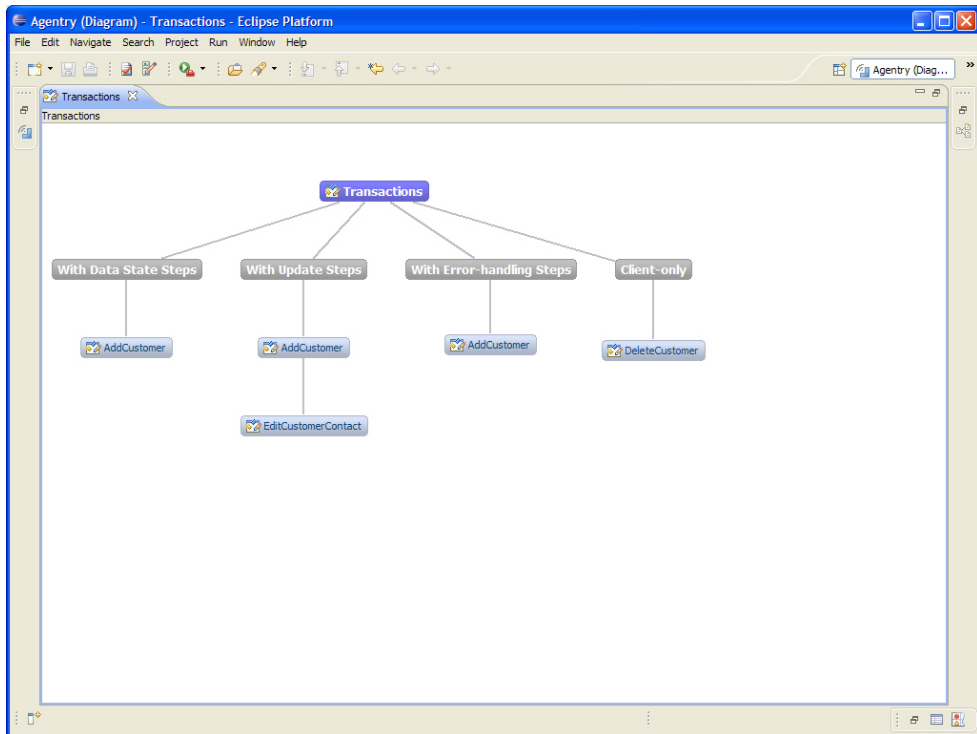
Diagram View

The Diagram View provides a graphical representation of a definition. The layout, organization and appearance of this view is specific to the type of definition being displayed. Many of these are self-explanatory in nature. Not all definitions are displayed in the diagram view, as there is no benefit in doing so. The general rule of thumb is that definitions without child definitions are not displayed in the diagram view. The exception to this is the Rule definition type. Rules do not have child definitions. However they are displayed in the Diagram View, with the structure of the rule logic displayed in a read-only format.

For other definition types, the items in the Diagram View are organized according to the definition type selected. For many definition types this is a graphical representation of the parent-child relationship between the selected definitions and any child definitions. For others, certain organizational information is provided. It is this last group that is explained in more detail here.

Diagram View: Module Transactions

The Diagram View for a module's transactions displays all transactions within a given module. This Diagram View is displayed when the bold Transactions node in the Project Explorer View is double-clicked, or when it is right-clicked and the View Transactions Diagram menu item is selected. The Diagram View displays a module's transactions in a manner organized according to their back end processing definitions, i.e. the step usage definitions within the transaction.

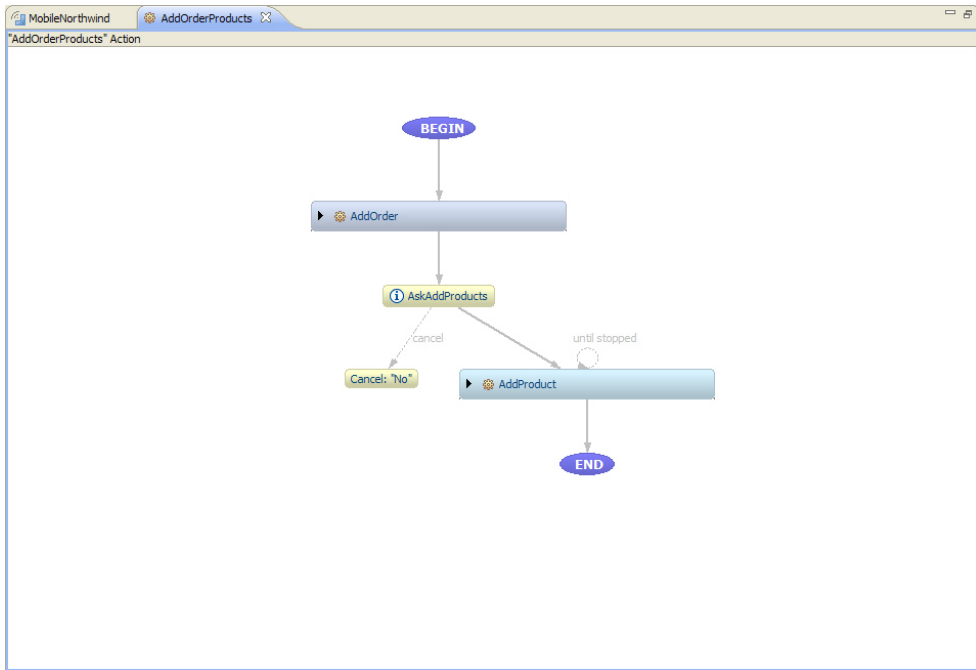


In this simple example there are three transactions displayed. Note that the AddCustomer transaction is listed three times, as it contains server data state steps, server update steps, and error handling steps. EditCustomerContact contains only update steps and is therefore listed only under that node in the diagram. Finally, DeleteCustomer does not have either data state or update steps. Therefore, it is listed under the Client-only node meaning it has no server-side processing.

Diagram View: Actions

The Diagram View for an action definition displays that action in a flow chart view representing the execution flow of the action. This can be a useful view for complex actions, especially those with subaction steps and looping behaviors.

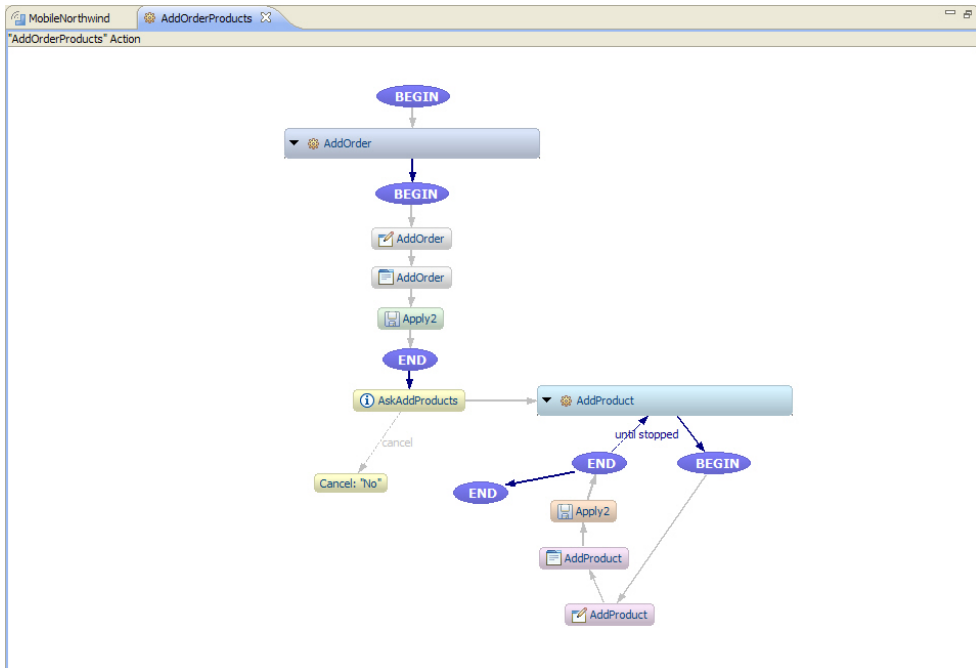
Following is an example of an action with two subaction steps. The first subaction step is defined to execute once and the second is defined to execute iteratively, also known as a looping subaction step:



The first subaction step is **AddOrder**, which executes a second action for an Add transaction. This step is defined to execute once. It is followed by a message step named **AskAddProducts**, which prompts the user to add products to the order created by the **AddOrder** step before it. If the user clicks the **No** button in the message, the Action ends execution, which is represented by the node "Cancel: 'No'" in the above example.

If the user clicks the **Yes** button in the message prompt the subaction step **AddProduct** is executed. As shown in the Diagram View, this step is defined to loop until stopped, meaning the users indicate they are finished. Once this loop ends, the action completes, as represented by the node **End** in the Diagram View.

The two nodes for the subaction steps can each be expanded to display the steps of the actions they execute. Following is an example of this same action with the subaction steps expanded:



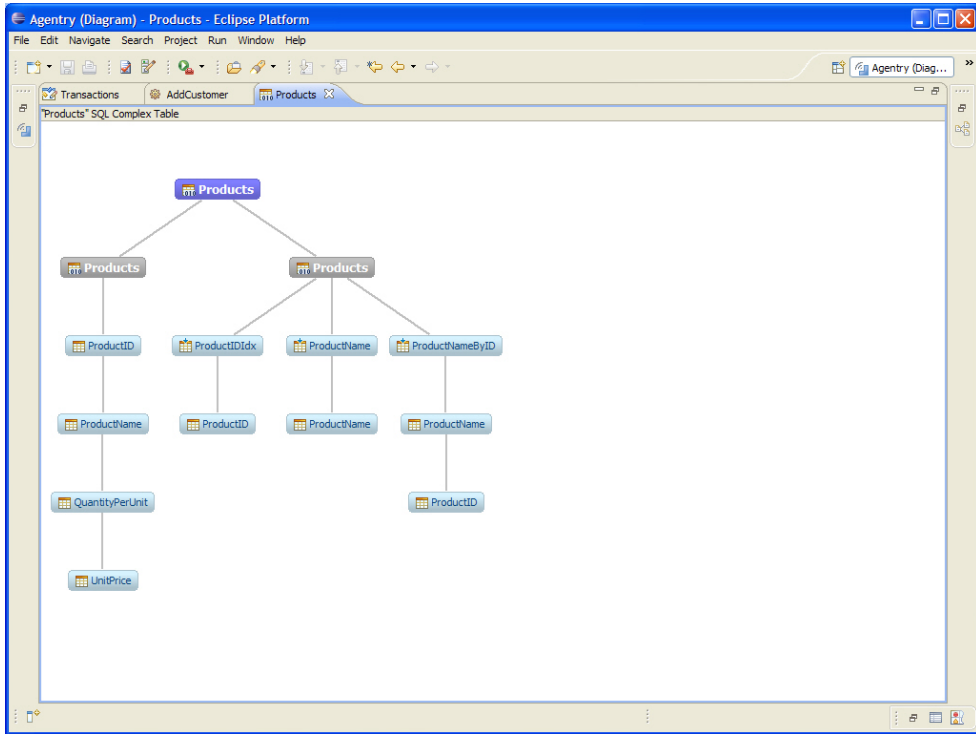
With these subaction steps expanded, the steps of each subaction are displayed. The AddOrder action contains the AddOrder transaction step that displays the transaction in the AddOrder screen set. This is then followed by the Apply step.

Next the Add Products prompt is displayed, which introduces a conditional execution to the action's execution flow. A positive response from the user executes the AddProduct subaction step. Since this step is defined to loop, the steps are presented in the diagram view to represent this behavior. The loop condition is displayed in the flow of this action, with the text "until stopped."

A negative, or "Cancel" response from the user to the Add Products prompt ends the action's execution.

Diagram View: Complex Tables

When displaying a complex table in the Diagram View, the indexes and fields are each displayed as child definitions to the complex table. In addition, the index nodes have one or more child nodes of their own. These child nodes are, first, the field for which the index was created and, second, if it is a child index the field or fields for which the parent indexes were created.



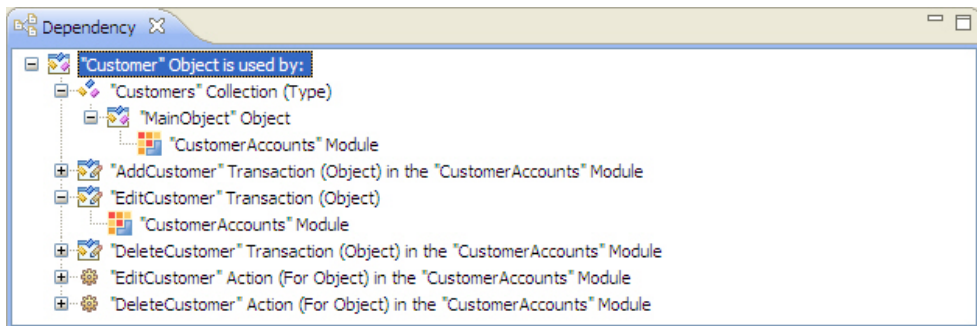
In this example, the index `ProductNameByID` is a child index to the `ProductIDIdx` index. This is represented by the Diagram View.

Dependency View

The Dependency View displays the definitions within the application project that reference the currently selected definition in the Project Explorer. This is useful information when editing or removing the selected definition, as you can readily see the definitions that can or will be affected by the change.

The list of definitions in the Dependency View also includes the hierarchy information. A definition can be expanded in the Dependency View to display its parent definition. This information provides the path up to the module level. Any definition in the Dependency View can be selected and displayed in the Properties View.

The following is an example of the Dependency View for an object definition:



Trash Bin View

The Trash bin View displays a list of all definitions that have been deleted from the Agentry application project. This view provides a holding pen of deleted definitions to allow for their recovery in the event the deletion was in error.

Following is an example of the Trash Bin View:

Name	Type	Original Location	Date Deleted
EditCustomerContact	Action	CustomerAccounts* Module	2010/08/10 12:53:35
Fax	Property	Customer* Object in the CustomerAccounts* Module	2010/08/10 12:52:16
Fax	List Screen Column	ShowCustomers_List_PPC List Screen in the ShowCustomers Screen Set in the CustomerA	2010/08/10 12:51:54

Each of the items in this list represent a definition that has been deleted from the Agentry application project. Within this view a definition can be selected and either restored to its previous location within the project, or permanently deleted from the project. Once a definition is removed from the Trash Bin it cannot be recovered.

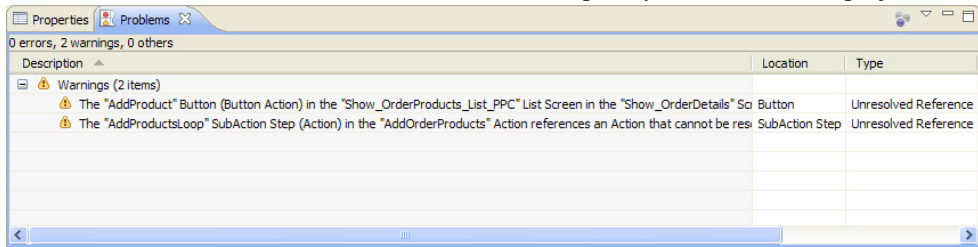
Problems View

The Problems View provides informational, warning and error messages related to the current Agentry application project. Following are the three classes of messages displayed:

- **Informational:** These messages are informational in nature and are innocuous in that they do not indicate any issue that may have a detrimental affect on the application project. Examples include information concerning the date and time of the application project's last publish, and a list of the definitions for which reminders have been set.
- **Warning:** These messages indicate a potential issue with the application project. Items resulting in a warning will not prevent an application from being published but may result in issues arising at run-time.
- **Error:** These messages indicate a validity issue with the application project and will prevent the project from being published. Such issues must be rectified prior to publishing the project.

The Problems View is always updated during a publish or a check on publish operation. Additionally, certain items are displayed in the Problems View as soon as a change occurs that indicates an issue.

As an example, if an action definition is currently defined and is used by a control or other definitions, and that action is then deleted from the project, the Problems View will immediately display a list of warning messages indicating those definitions. Following is an example of this messaging for an action that was executed by both a sub-action step in another action and a button on a screen. The action was subsequently deleted from the project:



This view is interactive. Double-clicking an item in this list displays in the Properties View the definition with the reported issue so that changes can be made to the definition. Once such changes are made the corresponding message will be removed from the Problems View.

Agency Views Outside of the Agency Perspective

The Agency Editor includes views displayed in the Agency Perspective as well as views displayed when certain operations are performed or displayed on demand by the developer. The views within Eclipse which may be displayed by the Agency Editor plug-in, but which are not a part of the default Agency Perspective, are explained briefly. The information provided here is introductory in nature, providing an overview of the behavior of these views. Details on each are provided in the content discussing the operations to which the views pertain.

Comparison Editor

The Comparison Editor is displayed during import, export, and share repository operations. This view displays the current Agency application project and a second source project side by side. Definitions within each are compared and those with differences between the two projects, or those that exist in one project but not the other, are highlighted.

For import operations the Comparison Editor is used to select the definitions in the import source to be imported into the current Agency project. For export operations this view is displayed when the export differences tasks is performed, displaying the differences between the application project and the comparison source, indicating what definitions are to be imported based on differences between the two. For share repository operations, the comparison view is displayed during updates from the repository to the current application project, indicating the differences between the two and the definitions affected by the update.

History View

The History View is displayed when the developer selects the **Team | Show History** menu item in the menu for the Agentry application project. This view is solely used for the team configuration functionality when the current application project is connected to a share repository. This view displays a list of all revisions within the repository. The revision from which the current application project was last updated is highlighted in this list.

The Data Tools Platform: SQL Development Tools

The Data Tools Platform is an open source project for the Eclipse platform that encompasses three separate but related Eclipse projects. The Agentry Editor plug-in for Eclipse make use of two of these projects: the Connectivity Project, and the SQL Development Tools project. Each of these projects provides perspectives and views to the Eclipse platform that can be useful to Agentry development, implementation, or configuration projects involving a SQL Database back end system.

Though these tools are provided by contributors to the Eclipse project and not Syclo, they are used by Syclo's Agentry Editor Eclipse plug-in to facilitate and support common development tasks. Information and instructions for working with and configuring these various tools as it relates to mobile application development and configuration is provided in the document set for the SAP® Mobile Platform. For extensive information on the Data Tools Platform project and its child projects, see the Eclipse help site at:

<http://help.eclipse.org/galileo/index.jsp>

Of note for the uninitiated developer is the *Data Tools Platform User Guide*, which is available in electronic form on the Eclipse help site listed above.

Connectivity Project and the Agentry Connector Studio

From the *Data Tools Platform User Guide*:

The connection-management functionality provided in the Connectivity project includes...components for defining, connecting to, and working with data sources.

One of the features provided by the Agentry Editor Eclipse plug-in is the Agentry Connector Studio. Using this tool an Object Wizard is displayed that allows for the definition of various data-related definitions within a module, with the attributes of those definitions set in part based on available information about the back end system with which the mobile application will synchronize data. The Connector Studio itself can be used with SQL Database, Java Virtual Machine, or HTTP-XML system connections. When working with a SQL Database system connection, the connector studio requires the use of the Connectivity Project tools within the Data Tools Platform.

Within this set of tools, there are specific items used by or required for the Agentry Connector Studio functionality when working with a database system. These include Connection Profiles, Driver Definitions and the related Driver Management Framework, and the Data Source Explorer View. To use the Agentry Connector Studio to create the module data definitions, a Connection Profile must exist for a connection to the database from which the data definitions will be defined. A Connection Profile makes this connection using a Driver Definition that encapsulate the method in which the connection to the profile is made. The Connection Profile then represents the connection to the specific database instance or database server. Connection Profiles created in Eclipse are exposed to the developer in the Data Source Explorer View.

The Agentry Connector Studio can then be accessed from within the Data Source Explorer View, when using the Connector Studio to access a database system. Within the Data Source Explorer, the developer navigates to the specific database table for which an object definition is to be created. Right-clicking on this table displays a context menu, which includes the menu item **Agentry Connector Studio**. Selecting this item will display the Connector Studio Object Wizard, which will walk the developer through the definition of the object and its properties, based on the schema information provided for the database table. It also provides the option for defining transactions for the new object type and SQL step definitions, including basic SQL statements based on the database table. These SQL statements are intended for use by the transaction server update steps, and the object read steps or fetch server exchange steps.

Instructions for creating a Connection Profile and Driver Definition within the Connectivity Project tools can be found in the *Agentry Implementation and Administration Guide*.

SQL Steps and SQL Synchronization Definitions: The SQL Editor View

The Agentry Editor plug-in makes use of the SQL Development Tools, another component of the Data Tools Platform. Specifically the SQL Editor View is the view used to display and edit SQL statements within the Agentry application project. The SQL Editor is a view that supports the authoring, editing, and testing of a SQL statement. When a definition that includes a SQL statement for synchronization is defined, the default behavior of Eclipse is to display that statement in the SQL Editor. This editor provides several configurable aides in authoring well-formed SQL logic, including helpers such as adding quotes around values that require them automatically, indentation/tab and other “pretty print” functionality, and other similar behaviors, all of which have default behaviors that can be configured and customized to the needs of the developer and the project. In addition, since the SQL Editor is a part of the SQL Development Tools project, and since this project is a part of the larger Data Tools Platform project, the SQL Editor can make use of the features provided by its sibling Connectivity project.

Specifically, if a connection profile has been created within the Connectivity Project tools, the SQL Editor can use that connection profile to open a connection to the database and execute the query within the SQL Editor against that database. This is a useful testing feature that can help developers verify the validity of their SQL statements within the application.

The Java Perspective

The Java Perspective provided with Eclipse is the main interface to the JDT project for Eclipse. This perspective, as well as other tools within the JDT, are used in development of an Agentry application that connects with a Java Virtual Machine system connection to synchronize data.

The Java Perspective is opened by default when Eclipse is started. The views for this perspective include the Package explorer. Within this view are the Java packages for the current project. Also listed in this tab are any Agentry application projects created and saved within the current Eclipse workspace. Selecting and opening this project in this view will open the Agentry perspective within Eclipse, displaying that project.

The Java components to the mobile application should be organized within a Java project. Organized within this same project the packages of the Java Interface through which data will be synchronized, and the Agentry Java API packages provided in the `Agentry-v5.jar` file should also be included.

The Java components of the Agentry application project are created and maintained using the tools and wizards provided with the JDT project. These include the Java Perspective, as well as other tools within the JDT to build and maintain the Java logic. When a definition is created in the Agentry Perspective that contains a Java synchronization component, the option as to the source of the logic for that component is presented. Depending on the selected source, the Java class wizard is displayed, which allows for the selection of the package and parent class for the new class, as well as the package and project placement of the class.

Information covering the Java Perspective, as well as the JDT of which it is a member, provided in the Agentry document set is limited to those areas of functionality in which the two directly relate. This includes guidelines for creating Java projects and packages for a mobile application development project, as well as use of the Java class wizard and some other tools within the JDT. The JDT itself is a robust Java IDE with far more features and functionality than will be covered in this document set. Extensive information can be found on the JDT at the web address:

```
http://help.eclipse.org/galileo/index.jsp
```

The item of note at the above URL for the uninitiated JDT user or Eclipse developer is the *Java Development User Guide*.

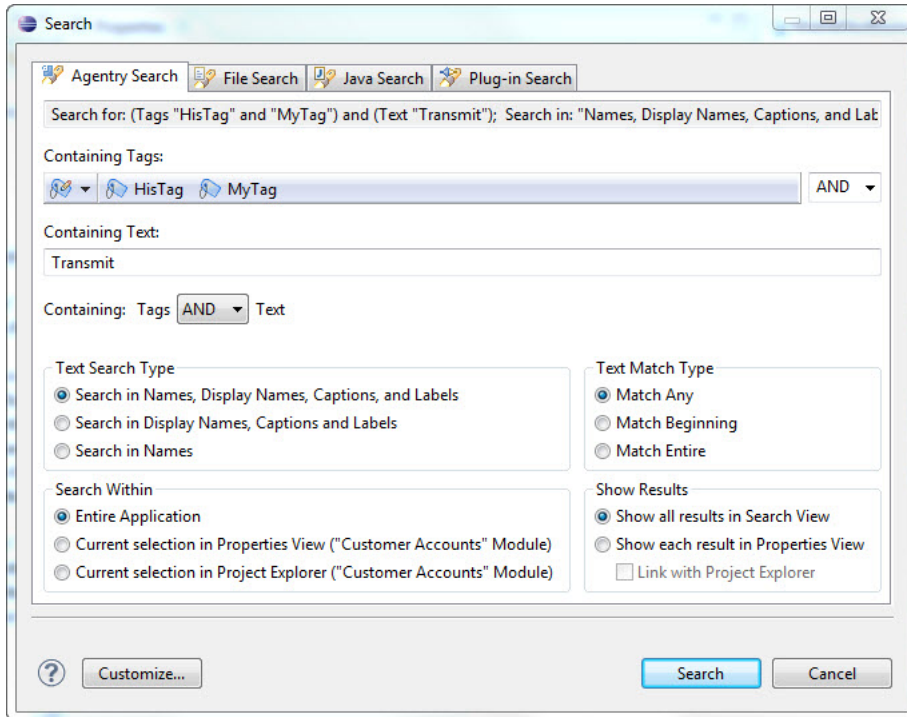
Searching Agentry Application Projects

The Agentry Editor to Eclipse includes search functionality. This functionality is supported through the standard search wizard within Eclipse, with the addition of an Agentry Search tab displayed in the Eclipse search wizard.

Within the Agentry Search tab there are numerous options available specific to an Agentry application project. Project definitions can be searched by tags, text contained within the

definition, and searches can be performed within the entire project or within the currently selected definition and its descendent definitions only.

To perform a search of the Agentry project, select the Eclipse menu item Search | Search... This displays the Search wizard, with the Agentry Search tab selected:



The following items can be selected to perform a search of the project:

- **Containing Tags:** One or more tags from the current project can be selected here, along with the option of definitions with no tag applied. The drop down list at the end of the list of selected tags allows for the selection of an AND or an OR search. AND requires the definition to contain all selected tags, OR returns definitions with any of the selected tags.
- **Containing Text:** Search text contained in the name, display name, caption or label of the definition. This can be further refined with the Text Search Type settings. Selection of the Containing: Tags AND/OR Text option will also impact the search results. AND will require the definition to contain both the selected tags and entered text. OR will return definitions with either the tags or the text or both.
- **Text Search Type:** Allows for the restriction of the search text to be found in one of the three groups of text values of the definitions.
- **Search Within:** The options here allow for the scope of the search to be set. This can include the entire project, the selected definition in the Project Explorer View and its descendents, or the definition displayed in the Properties View and it's descendents.

- **Text Match Type:** These options allow for search behaviors such as matching the whole word only, case sensitivity, and similar options.
- **Show Results:** The option to display all matching definitions in the Search Results View, or to display each definition in turn within the Properties View is set in this section.

Once the search options and criteria have been selected, click the **[Search]** button to search the Agentry application project. Matching definitions will be displayed according to the Search Results settings selected in the wizard. Only the definitions of the currently opened Agentry project are searched.

Agentry Application Projects: Creating, Managing, and Publishing

The Agentry Editor provides several features for creating, managing, and publishing the Agentry application project. An application project can be added to the current Eclipse workspace via an import or a new project created from scratch. Importing can be performed using one of a number of different sources as discussed in detail in the sections on importing. Creating a new project from scratch is performed using the New Application Wizard within the Agentry Perspective in Eclipse.

Project management features include the ability to export definitions from the project to a single file, as well as support for multiple developers through a common repository, or “Team Development,” a concept new to the Agentry 5.2 release. Exports can be performed for the entire project, manually selected definitions within the project, or automatically selected definitions based on differences between two different versions of the same project. Additional features include the ability to compare two projects or a project and export file and to selectively import components from a source to the current project.

Publishing is the task performed when modifications within the application project are in a stable state and can then be either tested or deployed to end users. The process of publishing can include development publishes, production publishes to a single Agentry Server instance, or production publishes to a cluster of Agentry Servers. The process of publishing to production for deployment can be performed directly to the Agentry Server(s), or, alternately, may involve an intermediary Agentry Production Server. This depends on the network environment and policies in the implementation environment and is discussed in detail in the sections on publishing to production.

Creating a New Agentry Application Project

Prerequisites

The following items must be addressed prior to performing this procedure:

- The Eclipse environment including the Agentry Editor must be installed.
- The Agentry Perspective must be open within Eclipse.
- The Eclipse workspace to which the new Agentry application project will be added must be open.

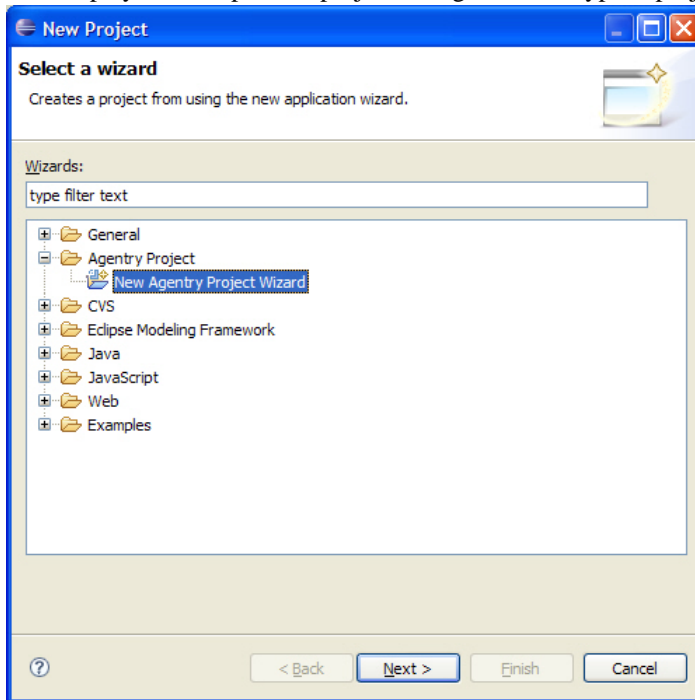
- It is recommended, though not required, that the Agency Development Server is installed to which development publishes will be made.

Task

The following procedure provides instructions on creating a new Agency application project. Perform this procedure when a new project is needed and that contains no existing business logic. If creating a new project within the current Eclipse workspace based on an existing Agency application project, export file, or published version residing on an Agency Server, see the information on importing Agency application projects.

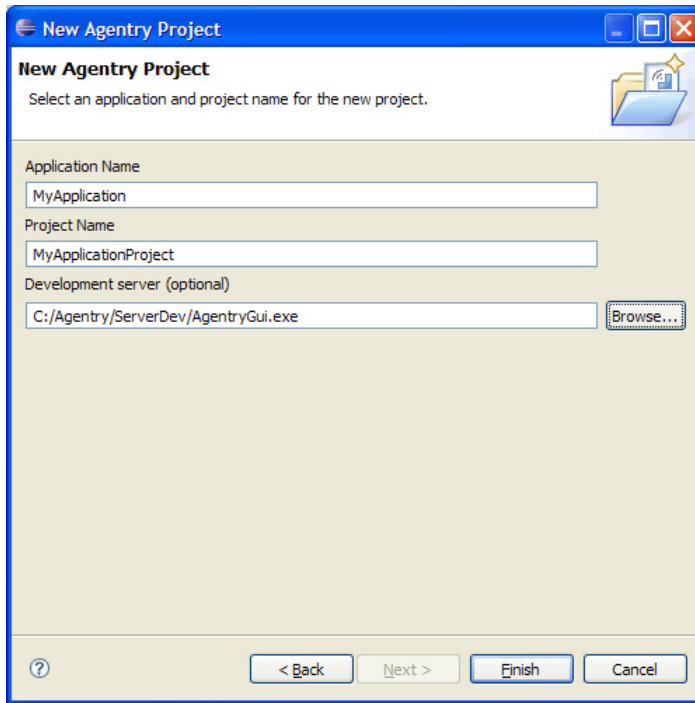
1. Start the New Application Wizard for Agency application projects by selecting the menu item **File | New | Project...**

This displays the Eclipse new project dialog where the type of project to create is selected:



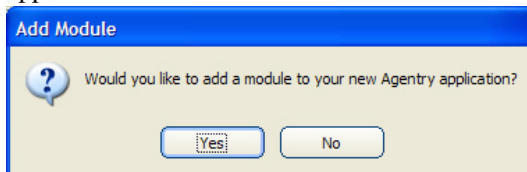
2. Select the item **Agency Project | New Agency Project** in the tree control displayed. Click the **[Next >]** button.

The first screen of the New Agency Project wizard is displayed:



3. In this screen enter the name for the mobile application, the name of the project by which it will be identified in the Eclipse workspace, and optionally the location of the Agentry Development Server that will be used in the development of the project.

The next prompt displayed concerns adding a module to the application. All mobile applications have at least one module:



4. To add a module to the new application project, click the [Yes] button.

A new module is created with the name NewModule1, and which should be edited to a more meaningful value.

The new application project has been created and stored in the Eclipse workspace. Depending on the selections made during this process, the following definitions will now exist within this project:

- Application
- A set of default transmit configurations:

- Dialin
- Network
- WirelessLan
- WirelessWAN
- A module definition
- The module MainObject, which in turn contains a collection property defined for the BusinessObject object definition
- A second object definition named BusinessObject, which should be renamed to a more meaningful value
- The module main screen set, defined to display the MainObject
- The module main fetch, defined to target the collection property within the MainObject.
- An action named Transmit that includes a single action step of type Transmit.

Next

With the completion of this process the mobile application project is created and development work can begin. In addition, and likely before the development work, additional configuration may be needed of the overall development environment. This can include the following, depending on the nature of the project:

- If synchronization with a Java Virtual Machine system connection is a part of the mobile application's behavior, create and configure a Java project and include the Agentry Java API packages, and other packages related to the back end system.
- When the development project is ready for its initial publish, configuration of the Agentry Development Server will begin with the publish, and will then be completed through the Management Cockpit and possibly through modifications to the Server's configuration files.
- Configuration of the Agentry Development Server to be used in the development process. This can include system connection configuration, logging behaviors, and other similar items.

Agentry Application Export, Import, and Comparison Introduction

A feature of the Agentry Editor is the ability to import and export application project definitions. Exporting from a project can include the entire project or any selected definitions within it down to the module level and will create a single Agentry Export File (.agx) containing all exported definitions. The source for comparing and importing to an existing application project can be an Agentry Export File, another application project, or a mobile application as published to the Agentry Server. An additional option for the source of an import is an Agentry 3.x Editor, which will also contain the application project it manages.

The uses for importing and exporting Agentry application projects include:

- Exporting to a single file in support of archiving an application project version control and backup.

- Exporting components of an application that contain differences from a base-line project. This is common when an product application has been configured or customized, and it is desirable to archive those changes for future reuse.
- Importing from an application project source to create a new project in the current Eclipse workspace. This may be done to create an application project from an archived export, or to upgrade an application project to a newer version of the Agentry Mobile Platform. Importing a project, Server published application, or export file created by a previous version will automatically upgrade the application project to the version of the Editor performing the import. Additionally, all products provided by Syclo are delivered with the Server and therefore must be imported to an instance of the Agentry Editor prior to extending or modifying the core functionality of the application.
- Comparing and importing components of another application project or export file to a current project to make use of common customizations or configurations, or to take advantage of previous development work in the current application project.

Import Functionality Overview

The import tools provided by the Agentry Editor provide functionality to support importing application definitions from other projects, published versions from Agentry Servers, and export files. Reasons for importing can include:

- Creating a new application project in the current workspace from an archived project or Agentry Server.
- Upgrading application projects to the current Agentry Mobile Platform version.
- Adding previously developed application definitions and components to the current project.
- Merging separate development work from multiple developers or a share repository created using the Team Development functionality.

Import Source

Import Source is the general term used to describe any item used as source for an import operation. Valid import sources within Agentry include:

- **Agentry Application Project Files (.apj):** This may be any Agentry project file, including its related definition files, created by the Agentry Editor. This project can be stored in a different Eclipse workspace, or elsewhere on the file system if created by a version of the Agentry Editor from 4.0 through all 5.0.x releases. Note that the .apj file is the main project file, but it is not the entire application project. The definition files that are a part of that project must exist in the proper Agentry Editor created file structure in order for this source to be imported. Typically this is a non-issue as there is no valid reason for the definition files within a project to be moved or modified manually.
- **Agentry Export Files (.agx, .agxz):** This can be any Agentry export file created using the Application Export functionality provided within the Agentry Editor. Note that as of

version 6.0 either standard or compressed export files can be imported. However, compressed export files cannot be imported by versions prior to 6.0.

- **Published Agentry Server Application:** An application published to the Agentry Server, either Development or Production, can be selected as an import source. The selection made for this source is the server's executable file. The application published to that server instance is then used as the import source. For Agentry Production Servers, any published version residing on that server can be an import source. Agentry Development Servers have only a single application version that may be an import source.
- **Agentry 3.x Editor Project:** In the 3.x versions of Agentry, application definitions were not stored in projects, but rather as a part of the data within the Agentry Editor instance. A given Editor contained the definitions for a single mobile application, meaning each mobile application included its own dedicated Agentry Editor. These definitions can be an import source, with the selection of the `AgentryEditor.exe` containing the desired mobile application to be imported.

As a part of the import process it is first necessary to select the type of import source. This information is required at the beginning of the screen flow for the import operation within the Agentry Editor. The above sources are valid import sources for import operations creating new projects within the Eclipse workspace, and for imports that add or replace definitions, known as a “compare and import” operation, within an existing project.

For each of these, the import source must have been created with a version of the Agentry Mobile Platform matching or prior to the version to which the definitions will be imported. There is no “downgrade” functionality provided by the import tools. As an example, it is not valid to select an Agentry export file created by version 5.2 as an import source for an instance of the Agentry Editor delivered in version 5.1. However, the reverse is allowed, importing from version 5.1 to version 5.2. This is, in fact, the proper method for upgrading an application project to a newer release of the Agentry Mobile Platform.

Adding a New Project to the Workspace

When importing to create a new project within the Eclipse workspace, the developer will first select the application source type and the specific source project. The import tool will then read in the source definitions, creating a new application project within the workspace. All definitions from the source are imported.

If the source for the import is not a full application project, or is missing definitions within the application hierarchy, changes will be necessary to the project. As a basic example, if an export file contains only the object definitions from a project, creating a new project by importing the export file will create a new Agentry application project containing an application and module definition. However, these parent definitions to the objects will contain minimal attribute settings with default values. Typically this is not a situation encountered often in real-world development environments.

Adding Definitions to an Existing Project

When performing an import to an existing application project, the current project and the source application for the import will be displayed in the Comparison View within the

Agentry Perspective. This view displays the current project and import source in side-by-side panes. Both are presented according the application hierarchy and are aligned based on the definition type and name. Each definition alignment is denoted as one of the following within this view:

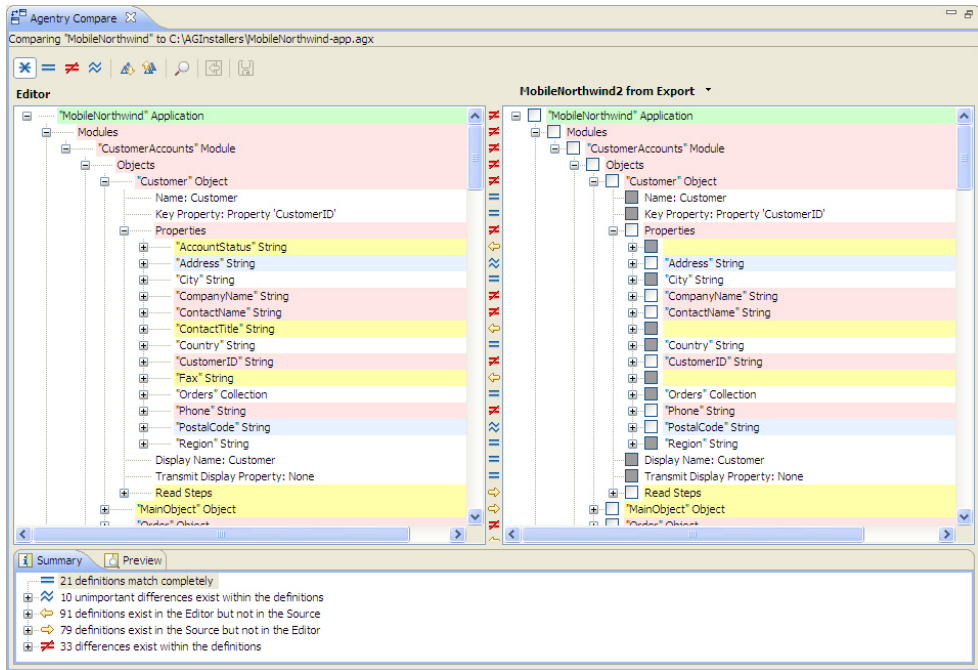
- **No Difference:** The definition in both the project and import source are identical, including child definitions and attribute settings.
- **Unimportant Differences Only:** The definitions in both the project and import source are the same in all areas that would affect run time behavior. Differences were found, but were limited to comments or descriptions only.
- **Exists Only in Source:** The definition exists only in the import source. It is not found in the current project. Such a definition can be selected in the import source pane and imported into the current project. Alternately this definition can be aligned with an existing definition of the same type and with the same parent definition. The source definition will overwrite the existing definition if it is then imported.
- **Exists Only in the Project:** The definition exists in the project but not the source. An import operation will have no effect on such a definition. Alternately this definition can be aligned with a definition in the import source. The source definition will overwrite the existing definition if it is then imported.
- **Differences Exists Between the Definitions:** The definition exists in both the project and the import source, but there are differences between the two definitions. This can include differences in attribute settings or differences in the child definitions. Child definition differences can include attribute differences, or a different set of child definitions.

Within the Comparison View the developer selects the items in the import source to be imported into the current project. This consists of checking and unchecking boxes within the import source indicating the specific definitions and their child definitions should or should not be imported.

The Comparison View

The Comparison View has been added in Agentry version 5.2. Its functionality and behavior is similar to the Comparison Screen that served the same general purpose in prior versions of Agentry. The new Comparison View includes additional functionality in support of the Team Development feature set. It also includes more information concerning the comparison and import, with lists now displayed summarizing the differences found between the project and import source, and a Preview tab listing the potential results of performing the import based on the current selections within the view.

The following is an example of the Comparison View within the Agentry Perspective:



In the pane on the left side of the View is the Agentry application project currently open in the Agentry Editor. On the right are the application definitions in the import source. Between the two are the icons for each definition indicating the comparison status between the project and import source. Following is a list of these icons and their descriptions:

Difference Icon	Description
	No Differences Found
	Differences in Definitions
	Unimportant Differences Found
	Exists Only in Project
	Exists Only in Import Source

Importing a New Agentry Project Into the Eclipse Workspace

Prerequisites

Address the following items prior to performing this procedure:

- Determine the source of the application project you are importing in this procedure and that you have access to that source.
- Verify that the application project source was created with the same or earlier version of the Agentry Mobile Platform. You cannot import projects, export files, or published applications created with a later version into an earlier version.
- Verify the workspace in which you are importing the project is the currently opened workspace in Eclipse.
- Determine a name for the project as it will be listed in the Eclipse workspace, as this is required information entered in the import process.
- Determine a value for the **Name** attribute of the application definition. This is information required during the import process.
- Though not required, it is strongly recommended that the Agentry Development Server to which development publishes are performed is installed. While the location of this Server instance is optional during the import, it is necessary information when defining any of the synchronization logic within the project after it is created.

Task

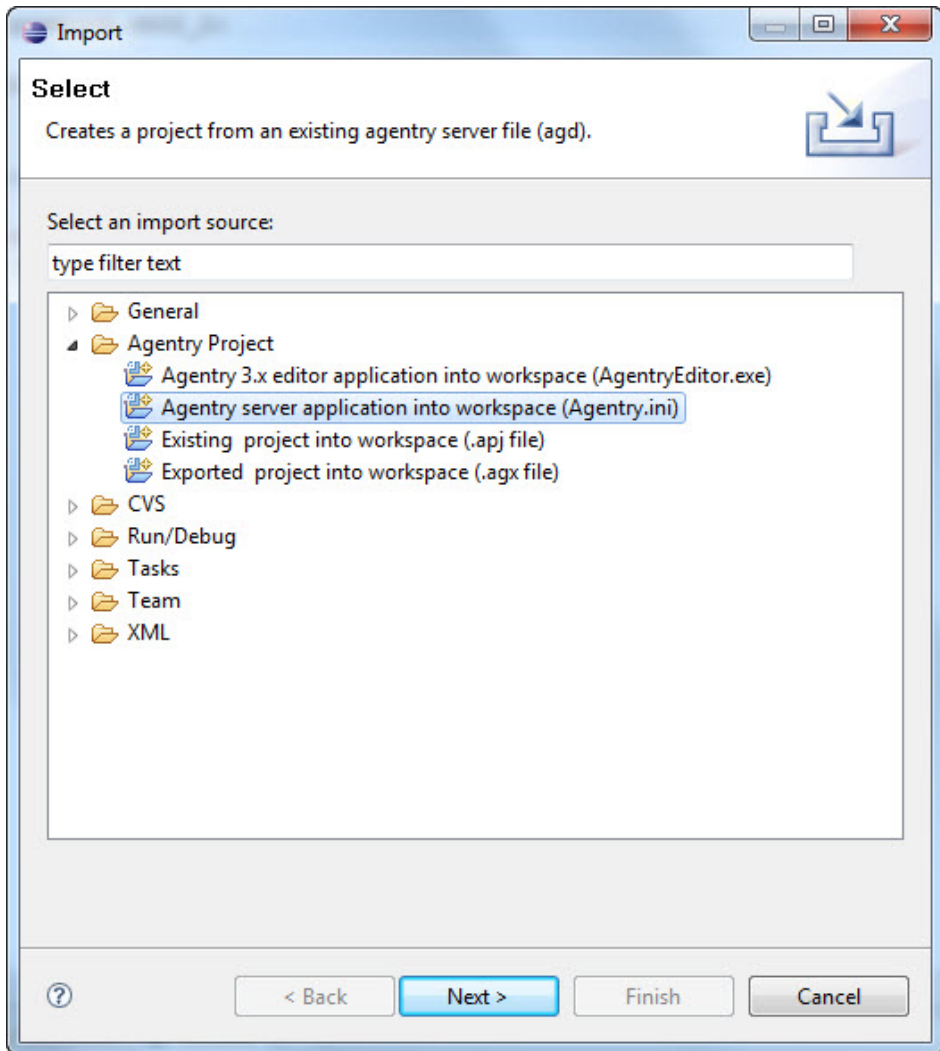
This procedure describes the steps involved in importing an application project into the current Eclipse workspace. When this procedure is complete, a new Agentry application project is created in the current Eclipse workspace. This project contains the definitions and application components found in the import source.

Note that this process excludes any related projects for the source application that may reside in that source project's workspace, such as Java development projects and related packages. Import these related projects and components according to the process that matches that project type, using tools found in Eclipse. Whether the Agentry application project is imported before or after other related projects is unimportant. However, all items must be imported and/or configured before modifications are made.

This procedure accomplishes the following:

- Checks out an Agentry application project from an Agentry share repository and creates a new local project based on the top revision within that repository.
 - Migrates an application project from one workspace to another.
 - Upgrades an application project or application export file created in a previous release of the Agentry Mobile Platform.
 - Restores or recovers an application project from an archived project or export file.
 - Creates, restores, or recovers an application project from a mobile application published to the Agentry Server.
1. If not already open, open or create the Eclipse workspace in which to import the new project. Opening or creating a workspace in Eclipse begins by selecting the menu item **File | Switch Workspace** and following the on-screen instructions.
 2. Right-click an empty area in the Project Explorer View and select the menu item **Import....** Alternately, select the menu item **File | Import...** in the Eclipse main menu.

The Select Import Source screen displays.



3. On this screen are the different import sources for Eclipse. Two of these pertain to Agentry application projects: **Agentry Project** and **Agentry Share**. Select the desired source by expanding one of these nodes and selecting the appropriate item under it. Click **[Next]** to continue.

The Select Source screen displays. This screen will be slightly different depending on the selected import source type. The following example is for a source type of Agentry Server application:

Import Agentry Server Application

Select Agentry Server
Select a directory to search for an existing Agentry server.

Agentry Server
C:/Agentry5.2/ServerDev/Agentry.ini Browse...

Source Application
Development MobileNorthwind

Application Name
MobileNorthwind

Project Name
MobileNorthwind

Development Server (optional)
C:/Agentry5.2/ServerDev Browse...

? < Back Next > Finish Cancel

4. In this screen the information entered is dependent on the source type selected in the previous step. Enter the information according to the following:
 - a) The specific item selected for the source is different based on the type. Select the source by clicking the **[Browse]** button.
 - b) The **Source Application** box is only displayed when the source type is an Agentry Server. This lists the name of the application published to that Server instance. If the selected server instance is a production Server, each published version currently residing on that Server is listed and you can import any one of those versions.

- c) The **Application Name** is the name given to the mobile application that is created by the import. This is set as the value of the Name attribute within the application definition and can contain no white space. This field is read-only if the selected import source is an Agentry share repository.
 - d) The **Project Name** is the name for the project within the Eclipse workspace. This must be a unique project name for the workspace and white space is allowed.
 - e) The **Development Server (optional)** is the Agentry Development Server for the application project being created. Any script files contained in the source project are copied to this Server. By default, this field is set to the Server from which the project is imported, if that Server instance is a development Server. Leave the option set as-is if this is the development Server for the new project, or change to a different development Server if necessary.
5. Verify the information entered is accurate and complete. Click **[Finish]** to perform the project import.

A new project is created by importing the definitions from the selected import source. The project is listed in the Project Explorer View and is automatically opened.

After this process is complete, the new project is added to the Eclipse workspace. The project is opened and displayed in the Agentry Perspective within Eclipse. The application name and project name match those values entered in the Import wizard. If the application project source was created using a previous version of the Agentry Mobile Platform, the new project was upgraded during the import to the version of the current Agentry Editor.

If the selected import source was an Agentry share repository, the new project contains the definitions found in the top revision of that repository. The revision is checked out to the current user. Subsequent changes made to the application project are tagged with that user ID. The project is connected to the selected share repository and you can update from it. Commit changes made locally to this repository.

Compare and Import Into an Agentry Application Project

Prerequisites

Before importing an application project source into an existing application project, the following items must be addressed:

- Determine the import source and verify access to that source.
- Verify the import source was created with the same or earlier version of the Agentry Mobile Platform. Export files or Server published applications created with a later version cannot be imported into an earlier version.
- Verify the correct Agentry application project to which definitions will be imported is the one currently open in the Agentry Perspective within Eclipse.
- Back up the current project by exporting the entire project, or by committing any changes to the Agentry share repository prior to beginning the import.

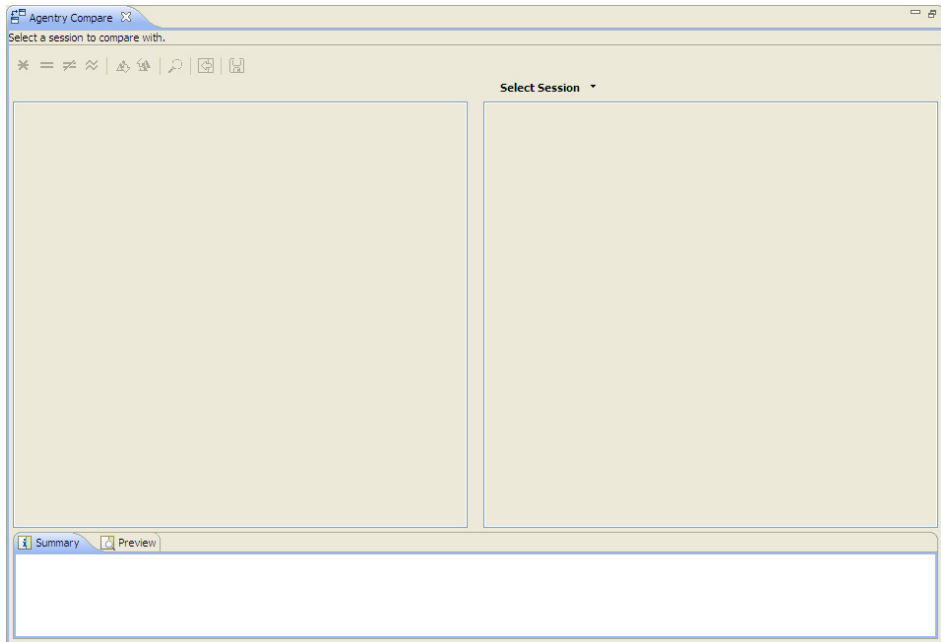
Task

This procedure describes the steps involved in importing definitions from an import source into the current Agentry application project. This process includes comparing the two projects and selecting those components to import to the current project. When completed, any definitions selected in the import source will be added to the current application project. The source application will be unaffected by the procedure. This process may be cancelled at any time.

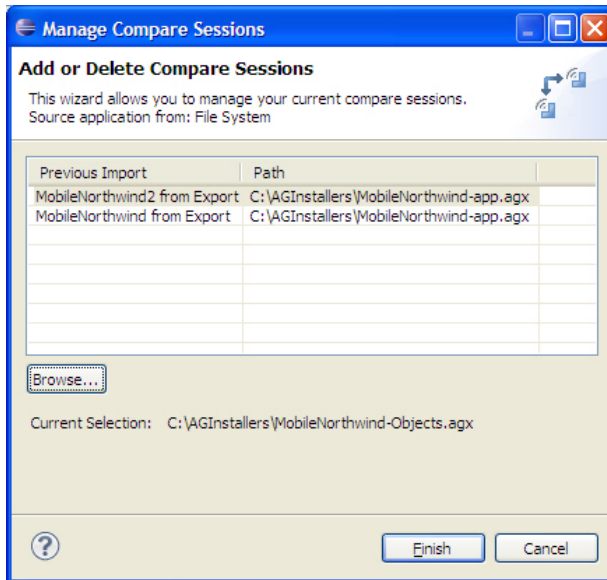
This procedure should be performed to:

- Compare and import from the currently connected share repository. This is a feature included in the Team Development support added to the system with the release of Agentry version 5.2. This includes manual compare operations where the share repository is selected, as well as updates from the share that result in conflicts.
 - Make use of archived customizations or components in the current application project.
 - Merge development work performed by multiple developers into a single master project. Note that this is *not* a part of the Team Development support and feature set. It is recommended that the Team Development features be used when coordinating work among multiple developers for the same application. Importing from an export file of another developer's work can still be performed, but should be limited to the scenarios of handing off responsibility for development, or when making use of modifications made to one project that are needed in another application project.
1. If not already open, select and open the Agentry application project to which definitions are to be imported in the Project Explorer View.
 2. Right click the root folder for the application project in the Project Explorer View and select the menu item **Compare With | Compare with other Agentry project**.

This will open the Comparison View in the Agentry Perspective, with no project items yet displayed. It will remain blank until the import source is selected for comparison:

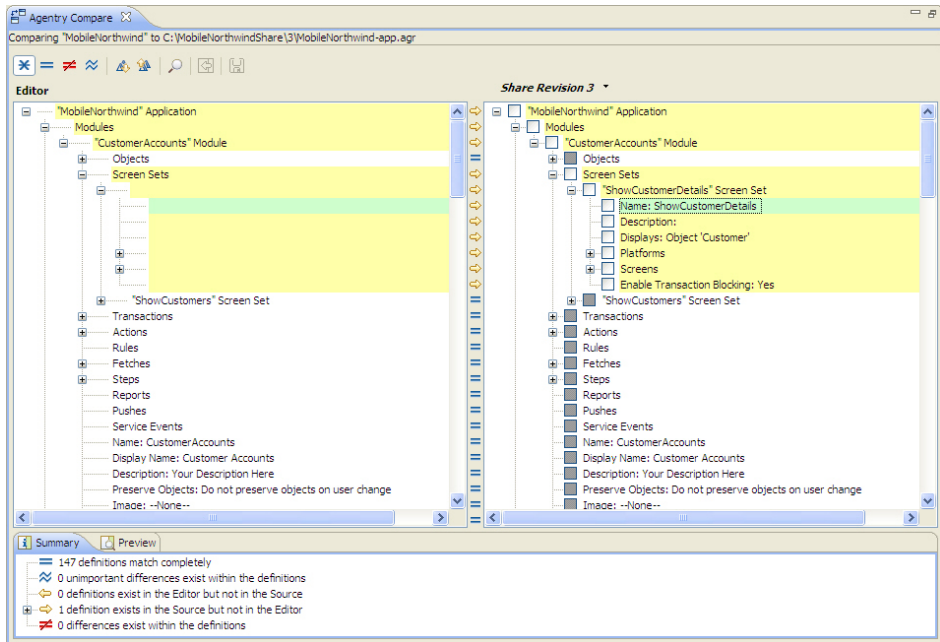


3. Within this view, above the pane on the right side, is the button **[Select Session]**. This button allows for the selection of the import source. Its behavior and the proper selection to make depend on whether or not the open project is currently connected to a share repository:
 - If connected to a share repository, clicking the button will automatically open the share revision matching the last revision to which the project was updated. The share is then set as the import source, and other revisions can be selected within the share to compare and import from within the share.
 - If not connected to a share, clicking this button opens the Manage Share Sessions screen. Within this screen either a previous import session can be reopened, or a new import source can be selected by clicking the **[Browse]** button below the list of save sessions:



- Alternately, whether connected to a share or not, clicking the drop down arrow for this button lists the import sources appropriate to the current project's connection status. Always listed are any previous import sessions, as well as a menu item to display the Manage Compare Sessions screen. If connected to a share, an additional menu item is displayed that opens a sub-menu listing all revisions of the share to which the project is connected.

Once the import source is selected, that source and the current project are opened in the Comparison View and displayed side-by-side:



4. Within the comparison screen now displayed, the pane on the left is currently opened application project. The pane on the right is the import source. Differences between the two are highlighted. Items can be selected in the import source to import to the current project by checking the boxes for those item's nodes. Definitions are aligned by default in the two panes based on definition name and type. To force two definitions of the same type but with different names to be aligned, right click either one and select the menu item **Align With | definition name**, where *definition name* is one of the possible definitions with which the selection can be aligned. Once all of the definitions to be imported from the import source have been selected, click the button Import in the view's toolbar. To save the changes, click the apply button in the toolbar.

The definitions will be moved to the left side pane indicating where they will be placed in the application project once imported.

5. Close the Comparison View by clicking the **X** button on the view's tab.

Once this procedure is complete, the selected definitions in the import source are now a part of the open application in the Agency Perspective. They may be modified further within the open project or otherwise used as needed. If the session was cancelled at any point, any applied imports will remain. Any imports that were not applied will be rolled back.

Next

Whenever importing into an existing application project, it is always recommended that a check is performed of the resulting project to verify the new definitions are defined as needed. Any modifications can be made according to normal processes. All imports, as with any other

application change, should be thoroughly tested before being published to a live production environment.

Export Functionality Overview

Export operations within the Agentry Editor are performed to store multiple application definitions, including a complete application project, in a single file known as an Agentry Export File (.agx, .agxz). When performing an export operation it is possible to manually select the definitions to be exported from a given application project, or to export the differences between a project and some comparison source project. Available with Agentry version 5.2 it is also possible to select the definitions to export based on one or more tags having been previously applied to those definitions.

When a definition is exported, the definition's attributes and child definitions are included by default. It is possible to select a definition for export and then deselect one or more of its child definitions. The resulting definition in the export file will only contain the selected child definitions.

Exporting Definitions - Manual Selection

When exporting definitions from an Agentry application project, it is possible to manually select the individual definitions to be saved in the export file. This can include selecting the entire application project, or any individual definitions within it. Also, it is possible to select to export definitions based on the tags that have been applied to them within the application. During the export process, one or more tags can be selected and those definitions that have been tagged accordingly are selected for export.

It is a common practice to export an entire application project to a single export file as a backup prior to making significant changes to a stable release of the application. Note that this same result can be accomplished using the Team Development feature set available in Agentry version 5.2 by committing a project to the share repository as a new revision prior to making modifications to the project. Either method of backup is acceptable and the one best suited to a given developer's environment or preference should be used.

Exporting Definitions - Exporting Application Differences

Exporting application differences is functionality that can be useful when making modifications to a core application for implementation-specific needs. Once such changes are complete, exporting the definitions involved in just those modifications can be accomplished by comparing the modified version of the application project with the original version. It is then possible to store such changes for later uses, including importing the modifications into the same core application at a different implementation with similar requirements.

Exporting differences involves comparing the Agentry application project with either an Agentry export file or an application as published to an instance of the Agentry Server. This comparison will determine which definitions exist in one version but not the other, and which definitions exist in both but are different from each. During the export process the source for the export is then selected, either the project or the comparison source. The resulting Agentry

export file will contain only those definitions found to be different as they exist in the selected export source. Adhering to this practice whenever implementation-specific modifications are made to a standard product application can result in a robust library of common customizations that can be imported into future implementations for less labor and time intensive implementation projects.

When performing an export of differences, a comparison is always made between the open Agentry application project within the Agentry Editor and a comparison source, which cannot be an Agentry application project. Valid comparison sources include:

- **Published Agentry Server Application:** An application published to the Agentry Development Server can be selected as a comparison source. Production server applications cannot be selected. The selection made for this source is the server's `Agentry.ini` configuration file. The application published to that Server instance is then used as the comparison source.
- **Agentry Export Files (.agx, .agxz):** This can be any Agentry export file created using the Application Export functionality provided within the Agentry Editor. Note that compression of export files is available in the 6.0 version and later. Compressed export files (.agxz) cannot be used as a comparison source in versions prior to 6.0.

Once a valid comparison source is selected, that source is compared to the open Agentry application project, with differences between the two highlighted in a comparison screen. Within this screen the export source, i.e., either the project or comparison source, is selected. When the export file is created, it contains the definitions found to be different within the selected export source.

Exporting Agentry Application Project Definitions

Prerequisites

The following items must be addressed prior to performing this procedure:

- The application project to be exported must be open in the Agentry Perspective.
- Identify the location of the export file to be created by this process and verify read-write and network access to that location.
- Determine the desired file name and a brief description for the export file that will be useful for later reference and identification. This information is entered during the export process.
- If exporting a subset of the definitions within the application project, identify and take note of those definitions before proceeding.

Task

This procedure describes the steps necessary to export definitions from an Agentry application project. During this process it is possible to select definitions with the application project to be exported, or to export the entire application. When selecting individual

definitions, all child definitions to the one selected will also be selected by default. It is possible to export definitions down to the module-level.

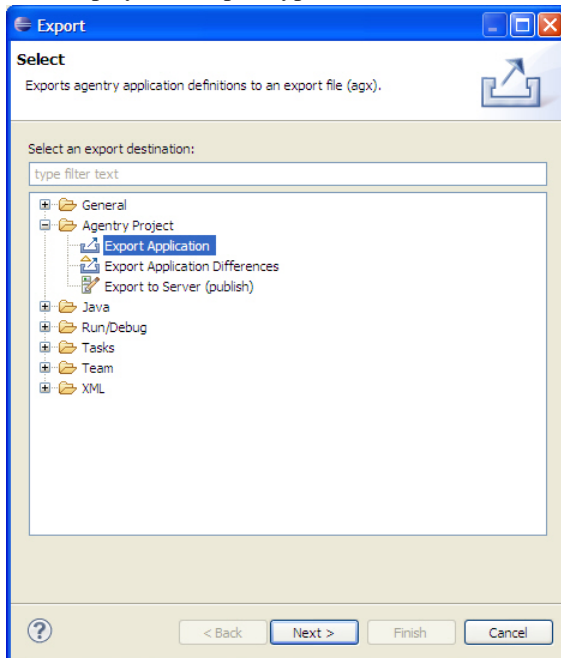
This procedure should be performed to:

- Archive an application project for version control and back up purposes.
- Export and archive components of a project for version control and back up purposes, or to make available for merge into a master application in a multi-developer effort (see information provided on Team Configuration for an alternative to this manual procedure).

With the release of the Agentry Mobile Platform 6.0 the default behavior is to create a compressed export file (.agxz). The preference pages in Eclipse for the Agentry Editor plug-in provide the ability to change this default behavior to create standard export files. The process for creating an export file is the same regardless of whether or not the file is compressed.

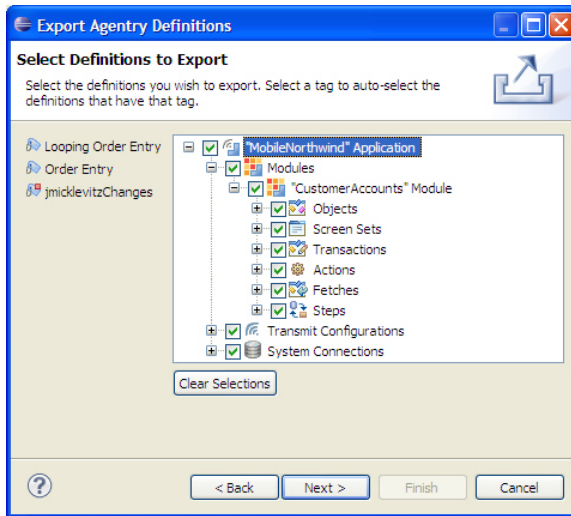
1. To begin the export process, right click the root project node in the Project Explorer view of the open Agentry application project. In the context menu select the item **Export...**

This displays the Export type selection screen:



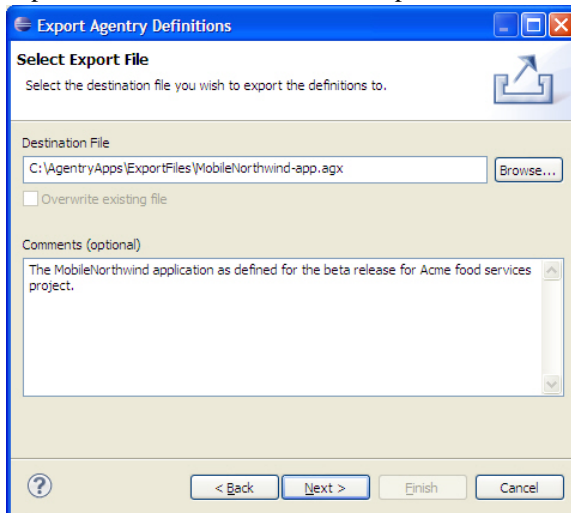
2. Select the **Agentry Project | Export Application** item in this screen. Click the [Next >] button.

This will display the first screen of the Export Wizard where the application definitions to export can be selected:



3. Within this screen the definitions of the application project are displayed and can be selected for export by checking the associated box for each. Checking a given definition will automatically select all child definitions. Alternately, one or more of the tag buttons to the left of the project can be selected, which will then automatically select all definitions with the associated tag to be exported. To export an entire application project, simply check the root Application node in this screen. Click the **[Next >]** button to proceed.

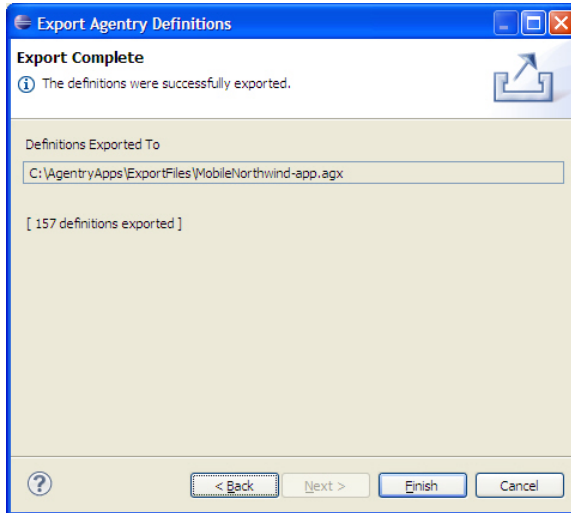
This will display the next screen of the Export Wizard where the name and location for the export file is entered, as well as an optional comment:



4. In this screen select the location and file name for the export file to be created. Ensure the selected location is one to which the Windows user has read-write privileges and network access where applicable. A comment can also be entered at this time. The contents of the

comment field are displayed as tool tip for the export file once it is created. Information concerning the date and time of creation and the version of Agentry are always a part of the tool tip for the file and need not be a part of the comments. Click the **[Next >]** button to proceed.

The export will now begin. When completed, the following summary screen is displayed:



5. Click the **[Finish]** button to close the Export Wizard.

Completion of this procedure results in the creation of an Agentry Export File containing the selected definitions from the Agentry application project. This file can now be archived in a version control system, made available to other developers for import, or moved or copied to any desired location. It can be used as an import source to create new Agentry application projects or to import definitions into another project where needed.

Exporting Agentry Application Project Differences

Prerequisites

The following items must be addressed prior to performing this procedure:

- The Agentry application project to be compared against a comparison source must be open in the Agentry Perspective within Eclipse.
- The comparison source must be accessible to the Windows user and Eclipse.
- The name and location for the export file to be created should be determined and read-write and network access to this location should be confirmed.
- A comment for the export file should be determined that will be useful for later reference.

Task

This procedure describes the steps necessary to export the differences between an Agentry application project and a comparison source. This process will create an Agentry Export File

containing the definitions from either the open application project or comparison source deemed different from the other. This includes definitions found in one but not the other, or definitions found in both but that contain attribute differences. Before the export proceeds, those definitions to be exported are highlighted in the Comparison View.

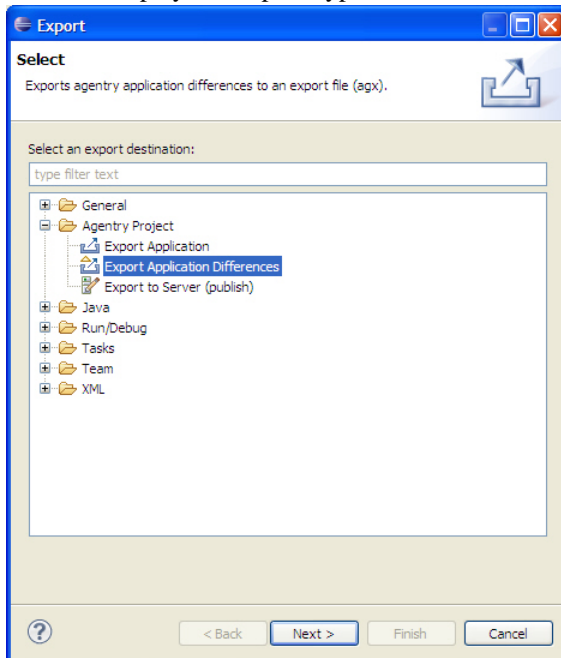
This procedure should be performed to:

- Capture the differences between one application version and another for archive purposes.
- Capture differences made for implementation-specific configuration or customization. Such changes can be archived for later import into other implementations with similar functionality requirements.
- Other use cases where it is desired to export the differences between two Agency application projects.

With the release of the Agency Mobile Platform 6.0 the default behavior is to create a compressed export file (.agxz). The preference pages in Eclipse for the Agency Editor plug-in provide the ability to change this default behavior to create standard export files. The process for creating an export file is the same regardless of whether or not the file is compressed.

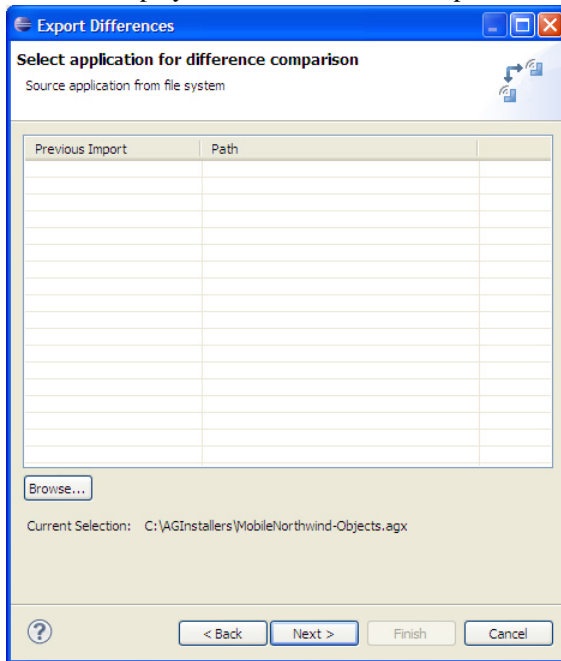
1. To begin the export process, right click the root node of the open Agency application project in the Project Explorer View and select the item **Export...** in the context menu. Alternately, select the menu item **File | Export...** in the Eclipse menus.

This will display the Export Type Selection screen, where the type of export is selected:



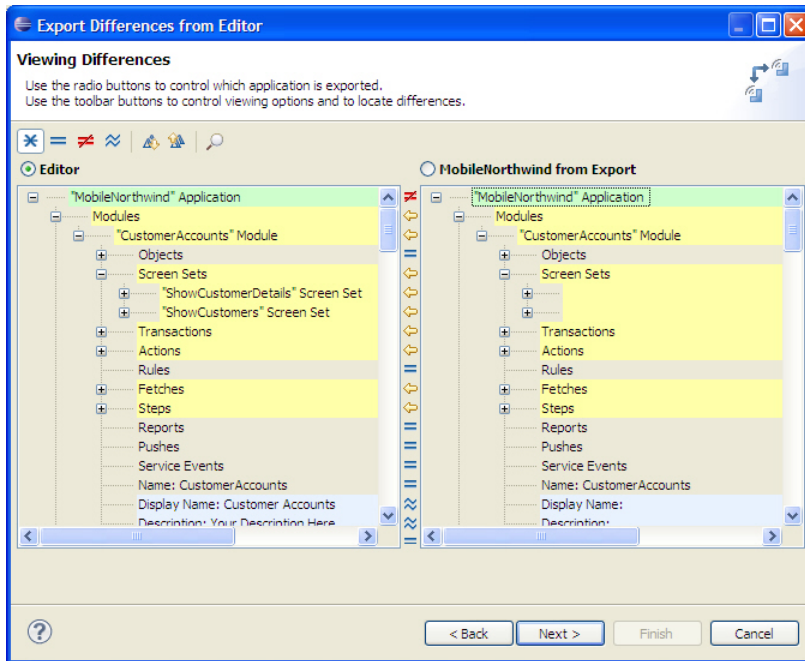
2. To export the differences between two projects, select the item **Agentry Projects | Export Application Differences**. Click the **[Next >]** button to proceed.

This will display the first screen of the Export Differences wizard:



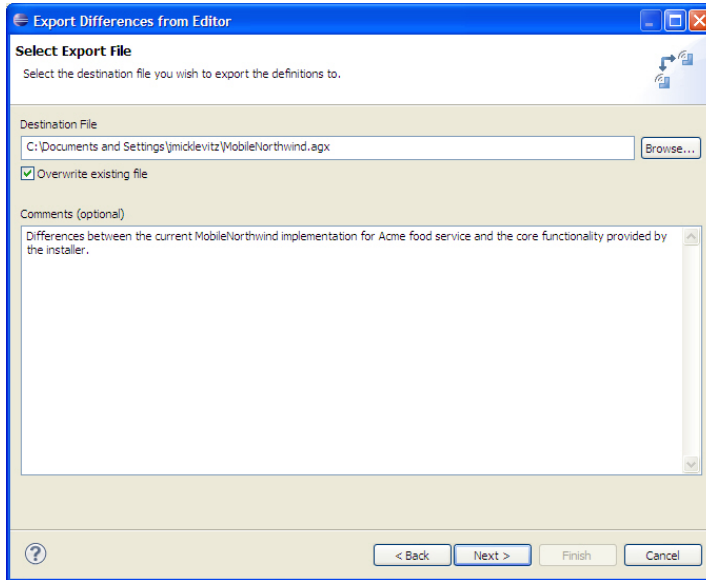
3. The list control on this screen displays the previous export differences sessions. Items can be selected in this list as a comparison source. Alternately, to select a difference source click the **[Browse]** button below the history list to display a Windows file dialog where the comparison source can be selected. Valid options are either an Agentry export file or an instance of the Agentry Server. Once the comparison source is selected, click the **[Next >]** button.

If a new source is selected, a prompt is displayed to enter a name for the history list. Enter this now if necessary. The next screen displayed is the comparison screen:



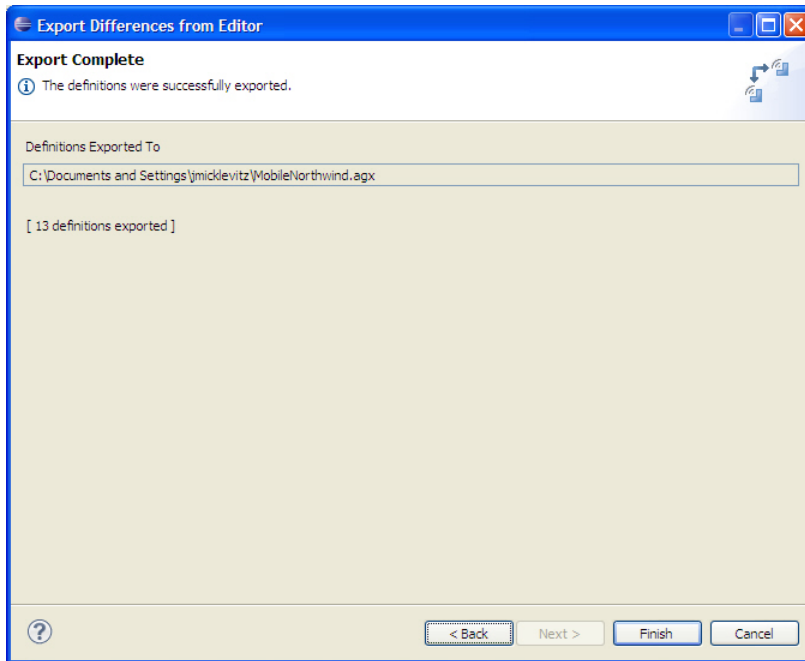
4. This screen displays the Agency application project on the left, and the comparison source on the right. Differences between these are highlighted. Above each is a radio button that can be selected to indicate that the definitions from that source, either the Application project in the workspace, or the comparison source, will be exported. The definitions exported from the selected source will be those found to be different from, or that do not exist in the other project. Once the selection has been made, review the definitions to be exported and then click the **[Next >]** button.

This displays the destination and comment screen:



5. Within this screen select the location and file name for the export file to be created by this process. If the file exists, check the **Overwrite existing file** check box to replace the existing export file. Optionally, enter a comment for the export file. This comment is displayed for the file as a tool tip in any Windows Explorer or File Dialog. Information concerning the date and time the file is created and the version of Agentry that created it are automatically a part of this tool tip and need not be a part of the comments. Click the **[Next >]** button to proceed with the export.

The export is performed and the export file is created. The following summary screen is displayed:



6. Click the **[Finish]** button to close the Export Differences Wizard and return to the Agentry Perspective in Eclipse.

Completion of this procedure results in the creation of an Agentry Export File containing the definitions from the selected source, either the open project or comparison source, found to be different. This file can now be archived in a version control system or other repository, made available to other developers for import, or moved or copied to any desired location. It can be used as an import source to add these definitions to another Agentry application project where similar functionality is needed.

Publishing Applications Overview

In order to test application projects or to deploy applications to users, the application project must be published from the Agentry Editor to the Agentry Server. If you are ready to distribute the application to the end users, it must then be deployed to the SAP Mobile Platform Server, after it has been published. At this point it will then be made available to and downloaded by those Agentry Clients that synchronize.

There are two types of publishes that can be performed, Development and Production. When publishing an application from the Agentry Editor, there are various aspects that will affect the type of publish performed and which options to select during the process. Additionally, the first time an application project is published there is some initial configuration performed by the publish process that will affect the Agentry Server.

Other options exist for the publish process, including the creation of files used for localization of the application. When performing a production publish, the publish version of the application must be set. Also during a production publish the option exists to delay the deployment from the Agentry Server to Agentry Clients until a specified date and time.

For all publish operations the application data is copied to the Agentry Server, with transformation of the application data into the format for the Agentry Server. This format will vary depending on whether the target Agentry Server is configured for development or production. The location of the application data will be in the Agentry Server's configuration directory, in either the subdirectory `Application\Development` for a development publish, or `Application\Production\Version` for a production publish. To perform a publish it is necessary to have read-write access to this location.

Publishing During the Development Cycle

When developing a new application or when customizing an existing one, it will be necessary to perform several publishes to an instance of the Agentry Server provided within the SAP Mobile Platform Server. This Agentry Server instance should be installed as a development server; that is, the Configuration property `Development Server` should be set to true. Note that it is false by default and must be modified after the SAP Mobile Platform Server has been installed.

Publishing to Production for End Users

When development and testing are complete, the application can be published to the SAP Mobile Platform Server. This is accomplished by performing a production publish to the Agentry Server configuration directory within the SAP Mobile Platform Server installed for end users. A production publish includes specification of a publish version number, which dictates certain application update behaviors performed by during the next transmit, and as described in the information provided specific to production publishing.

Compression of Published Files

The files generated during a publish are compressed by default. This behavior can be altered using the preference settings in Eclipse for Agentry. There is no difference in the process of publishing the application based on whether or not the files are compressed. The Agentry Server automatically determines whether or not the files are compressed and will process them accordingly.

Development Publish Description and Overview

When performing development work, customizations, or configuration to an application project, the development and unit testing is normally performed in the Agentry Development Environment with components installed from the SAP Mobile SDK. This includes the Agentry Development Server to which the application will be published, and the Agentry Test Environment, used for development and other testing of client functionality. The Agentry Server is installed within the SAP Mobile Platform Server and must be configured to be a development server. See the administration content on the SAP Server for details on changing

this configuration. When publishing to a Development Server, the publish performed is a development publish. A development publish should only be performed to an Agentry Development Server.

A development publish will transfer the definition files in the application project from the Agentry Editor to the Agentry Development Server. The files are stored on the file system at the installation location of the SAP Mobile Server; specifically within the configuration directory of the Agentry Server in the sub-directory `Application\Development`. A development server has no concept of application versions. When a development publish is performed, the previous version of the application on the Server is overwritten.

In a development publish, SQL scripts, file system scripts or batch files are written to the file system as separate, editable files. This allows for modification to these synchronization components on the Server without the need for a publish. Any other application modifications made require a publish to the Server to be available to the Agentry Clients or Agentry Test Environment.

The options for a development publish are limited to selecting the Agentry Development Server to which the application will be published, and whether or not to create the localization base files. Delayed deployment and version information cannot be set.

The main reasons for performing a development publish are to perform development and unit testing of an application during the development process. Additional testing should also be performed in a Production environment established for the purpose of quality assurance and user acceptance testing.

Publishing to Development


Prerequisites

Before publishing an application project to the Agentry Development Server, the following items must be addressed:

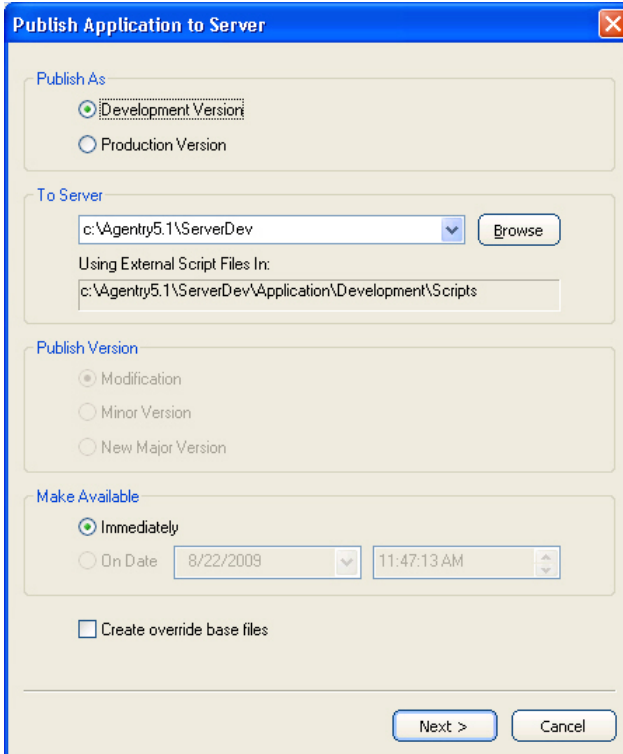
- Verify the Windows user for the Agentry Editor host system has read-write and network access (where applicable) to the installation location of the Agentry Development Server.
- Confirm the application project is in a state for which a publish is reasonable and ready for testing.
- During the publish process checks will be made as to the overall integrity of the application project. Errors or warnings will be listed in the Problems View in Eclipse. Prepare for the possibility of needing to correct these issues in order to perform a successful publish. An application can be published with warnings, but not with errors.
- Verify the version number of the Agentry Development Server and the Agentry Editor are the same, excluding any patch or point releases. Publishing from an Editor from one major or minor release to a Server from another release is not supported.
- The Agentry application project must be open in the Agentry Perspective.

Task

This procedure describes the steps necessary to perform a development publish from the Agentry Editor to the Agentry Development Server. When this process is complete, the application project will exist on the Server and will be deployed to any Agentry Clients that perform a transmit with that Server. This process is performed whenever it is desired to publish an application project for the purposes of development or unit testing in advance of eventual deployment to a production environment.

1. Begin the publish by clicking the toolbar Publish  button in Eclipse.

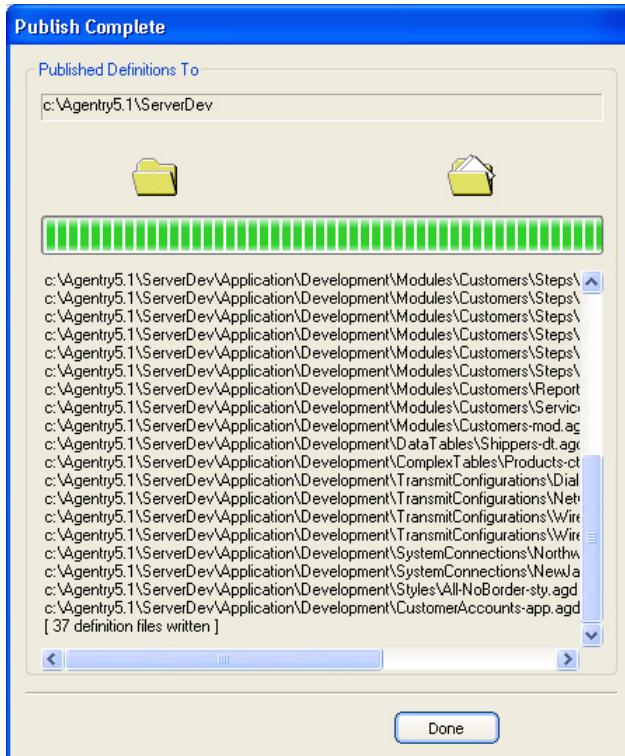
This displays the first screen of the Publish Wizard:



2. All options affecting the publish, including the type of publish, are set on this screen. For a development publish the following options are set:
 - The **Publish As** option is set to Development Version.
 - The **To Server** option lists the most recently selected Development Server, or the Development Server specified when the application project was created. Any other Development Servers to which a publish has been performed will be listed in this drop down. Select the Development Server in this field by selecting it from the drop down or by navigating to the Server's installation location using the **[Browse]** button.

- The **Create override base files** option can be set to generate the files used in support of localization. These files will contain all the display strings within the application project, with each display string including an identifier. The contents of these files can be translated and reincorporated in the Server to localize the application. See available information on localizing an Agentry application on the next steps after these override base files have been generated.

Once these options are set click the **[Next >]** button to begin the publish. The publish status screen will be displayed:



3. The total count of definitions published to the Server is now displayed. Click the **[Done]** button to close the Publish Wizard and return to the Agentry Perspective in Eclipse.

Performing a development publish results in the business logic encapsulated in the definitions being transferred to the Agentry Development Server where they can be deployed to clients connecting to that server instance. Any previous version of the application published to the same server instance is overwritten by subsequent publishes.

Production Publish Description and Overview

A production publish is performed to an instance of the Agentry Production Server within the SAP Mobile Platform Server, typically when it is time to deploy the application to users. Additionally, production environments are normally established for the purposes of quality

assurance testing prior to deployment. This can include publishing a new application project or publishing modifications to an existing project to create a new application version.

A production publish will transfer the application project from the Agentry Editor to the Agentry Server. The location of the project on the file system for the Agentry Server is the Agentry Server configuration directory in the sub-directory `Application\Production`. The entire project from the Agentry Editor is written to a single file at this location with an extension of `.agpz`. This file encapsulates the application for the version published. Note that in addition to this file, it may be necessary to manually transfer additional resources, where needed, depending on the nature of the application's synchronization logic. Examples include `.jar` files containing Java resources, `.dll` files, or other similar resources. Such files are completely dependent on the mobile application's architecture.

During the production publish the version number and deployment date and time can be specified. The specified version number can affect the behavior of the application during client-server synchronization of the new version, with a change to the major version number having one affect and a change to the minor or modification number another.

Production Publish Version Number Selection

As a part of the process of performing a production publish, the version number for the application must be specified. The first time a production publish is performed to the Agentry Server, the version number will be 1.0. During subsequent production publishes to the same Agentry Server instance, the version number is changed based on the selection made in the Publish Wizard.

An Agentry project's publish version contains the three components major, minor, and modification number. In the publish version 3.2 mod 1, the 3 is the major version, 2 is the minor version, and mod 1 is the modification number. The Publish Wizard will contain a section where the developer will specify which of these should be changed. This selection impacts the behavior of the application during the next synchronization by the Agentry Clients.

The specific difference is related to the processing of transactions sent from the Agentry Client to the Agentry Server. Depending on the nature of the change made to the application, it is possible that changes captured on the Agentry Client in transactions using the previous application version will be incompatible with the new version of those same transactions. When the synchronization definitions of the transaction (server data state or update steps) perform processing that will not work with the previous version of the application, it is necessary to process the pending transactions with the previous application version. To force this behavior, change the production publish version by incrementing the major version number.

When a new major version is specified, the Agentry Server will process each transaction sent by a Client using the old version of the application. Once all transactions have been processed, the new version of the application will be sent to the Agentry Client and dictate any subsequent synchronization behaviors. When either the minor or modification numbers are changed, the

new version of the application takes effect immediately, before the pending transactions on the Agentry Client are processed.

Because of this behavior, it is the responsibility of the developer performing the publish to determine which behavior is desired, and to then select the proper change to the publish version of the application.

Publishing to Production

Prerequisites


The following items must be addressed prior to performing this procedure:

- Confirm the application project is in a state for which a publish is reasonable and ready for production use or user acceptance and quality assurance testing.
- Obtain the directory within the SAP Mobile Server to which the project should be published from the system administrator.
- Verify the Windows user for the Agentry Editor host system has read-write and network access to the configuration directory of the Agentry Server defined within the SAP Mobile Platform Server.
- During the publish process checks will be made as to the overall integrity of the application project. Errors or warnings are listed in the Problems View in Eclipse. Prepare for the possibility of needing to correct these issues in order to perform a successful publish. An application can be published with warnings, but not when any errors are found. It is recommended that any warnings are corrected prior to performing a production publish, especially when deploying to end users.
- Verify the version number of the Agentry Server and the Agentry Editor are the same, excluding any patch or point releases. Publishing from an Editor from one major or minor release to a Server from another release is not supported.
- The Agentry Application Project must be open in the Agentry Perspective.
- Determine which of the major, minor, or modification number of the publish version should be changed, based on the desired transaction processing behavior related to Agentry Client transmits.
- Determine if the application should be immediately available, or if deployment to the Agentry Client should be delayed to a future date and time.

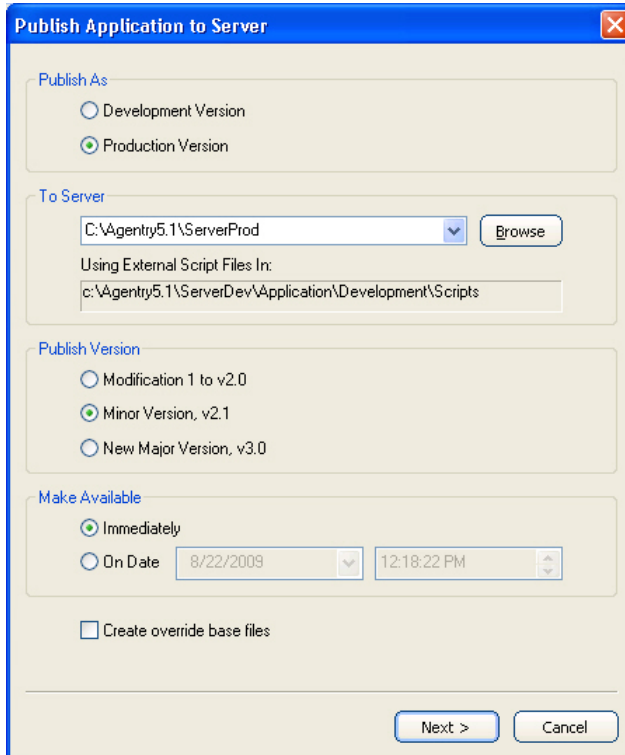
Task

This procedure describes the steps necessary to perform a production publish from the Agentry Editor to the Agentry Server. When this process is complete, the application project will exist on the Agentry Server. From here it can be packaged, along with other resources, for deployment to the SAP Mobile Platform Server.

In addition, Agentry Clients can also synchronize with this server instance for testing purposes, if desired.

1. Begin by clicking the toolbar Publish  button in Eclipse.

This displays the first screen of the Publish Wizard:



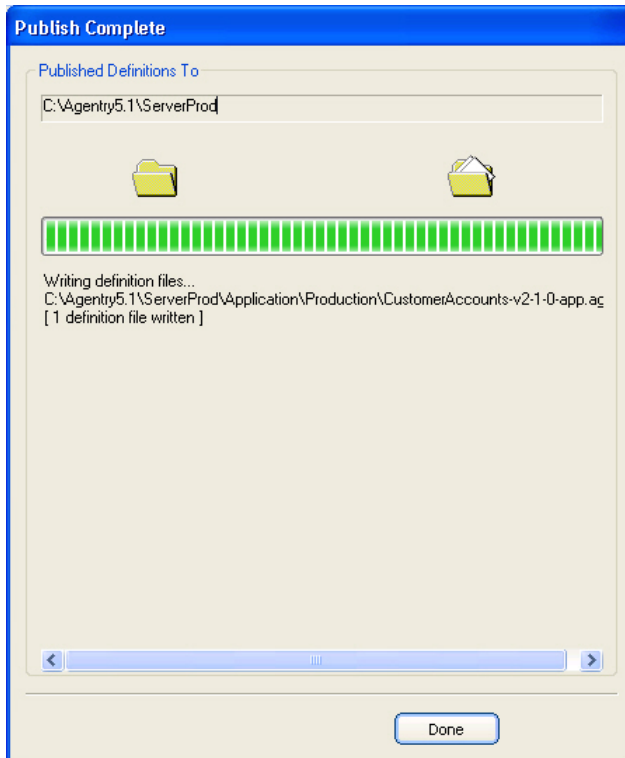
2. All options affecting the publish, including the type of publish, are set on this screen. For a production publish the following options are set:
 - The **Publish As** option is set to Production Version.
 - The **To Server** option lists the most recently selected production server. Any other production servers to which a publish has been performed are listed in this drop down. Select the production server in this field from this drop down, or by navigating to the Server's installation location using the **[Browse]** button. The proper location is the configuration directory of the Agentry Server defined within the SAP Mobile Platform Server for the mobile application.
 - The **Publish Version** is specific to a production publish. The option selected here increments one of the Major, Minor, or Modification numbers of the publish version on the Server. Increment the Major version when it is necessary to process transactions using the previous published version before providing Clients with the new version. Increment the Minor or Modification number when this processing is unnecessary.
 - The **Make Available** option is specific to a production publish. This option can specify whether to make this published version of the application available to Clients immediately, or at a future date and time. When a date and time is specified, the

application is published immediately, but will not be deployed to Clients until after the specified date and time.

- The **Create override base files** is set to generate the files used in support of localization. These files contain all the display strings within the application project. The contents of these files can be translated and reincorporated in the Server to localize the application.

If this is the first time an application has been published to the Agentry Server a prompt will be displayed when advancing the wizard indicating changes will be made to the Agentry.ini file. These changes include the addition of configuration sections related to the specific system connections defined within your project. Accept this message to proceed as these sections are needed.

Once these options are set click the **[Next >]** button to begin the publish. The publish status screen is displayed:



3. The total count of definitions published to the Server is displayed at the end of the publish. For a production publish this is always one file, regardless of the number of changes made. Click the **[Done]** button to close the Publish Wizard and return to the Agentry Perspective in Eclipse.

Performing a production publish results in the transfer of the application logic to the Agentry Production Server. The publish version of the application on that server instance is changed

according to the selection made in the Publish wizard. The business logic contained in the `.agp` or `.agpz` file, along with the accompanying `.ini` file, can now be bundled with the ZIP archive for deployment to the SAP Mobile Platform Server.

Introduction to Definition Tags

As of version 5.2 of the Agentry Mobile Platform the concept of Definition Tags is available. Definition tags, or simply tags, are a way to mark definitions of any type with a consistent tag for organizational purposes. Tags can be public or private. Public tags are associated with definitions and remain a part of those definitions during export, import, and share repository operations. Private tags are primarily for use with team configuration functionality, applied to definitions whenever they are changed, and are not included in any export, import, or share operations. Private tags are stripped from the definitions in the local project before they are committed to a share revision or exported to an Agentry export file.

Public Tags

Public tags are created and maintained by the developer within the Agentry application project. The developer can create as many tags as needed for the project, and a given definition can have multiple tags applied to it. When an application project is exported, committed to a share repository, or when a share repository is created from the local project, the public tags are included in the information written to those destinations. Imports from the export files, or updates from the share repository retrieve the tags for the definitions along with the definitions themselves and are displayed in the local project.

By default, public tags are manually applied to a definition by the developer. When a tag is applied to a definition, it can be applied to just that definition, or recursively applied to the selected definition and its descendents. Public tags can be removed from any definition that currently has a tag, and can be recursively removed from the definition and all its descendents that have the same tag.

As an optional behavior it is possible to set preferences in Eclipse to automatically apply one or more tags to any definition modified by the developer. Within the preference page “Tagging Configuration” for Agentry, one or more public tags can be selected for auto-tagging. This results in the selected tags being applied to any definition modified by the developer in any way. This continues until the auto-tagging is disabled for the previously selected tags.

Auto-tagging can be a useful feature when implementing a feature set or custom functionality in an existing product or previously deployed application. A tag can be created to mark those definitions that have been modified or added specifically in support of the new functionality. During subsequent export operations it is then possible to select these definitions by their tags, creating an export file containing just the definitions with the selected tag.

Private Tags

Within a local Agentry application project, there can be one designated private tag. The private tag is stripped from the definitions before they are committed to the share or before they are

exported to an Agentry export file. Private tags are primarily intended for use with the Team Configuration functionality.

The project's private tag is automatically applied to definitions when they are modified and only when the project is connected to a share repository. The private tag is used by the Agentry Editor during commit operations, with the definitions containing the private tag being those compared to the share revision to determine if there are differences.

Note that a definition with a private tag does not guarantee it will be committed to the share repository. If a definition is modified in such a way that at the time of commit it exactly matches the same definition in the share's tip revision, the local definition will not be committed to the share.

As an example, if a developer modifies the minimum length of a the string property City in the Customer object to a value of 5 and then commits, a new tip revision is created in the share. If a second developer that has not yet updated the local project to this new tip revision makes the same change to the City property in his or her local project, it will have the private tag applied to it. When the second developer then commits, however, the City property will not be sent to the share as the Agentry Editor recognizes that the two definitions are the same. If this is the only change made to the project, the commit will not proceed. Other changes will be committed if present.

The name of the private tag can be edited within the "Team Configuration" Agentry preference page. By default, if no name is specified, the default private tag name is *usernameChanges*, where *username* is the Windows user ID of the developer.

The private tag cannot be manually added to or deleted from definitions. If the Agentry application project is not currently connected to a share repository, there is no private tag available.

Tagging: Creating New Public Tags


Prerequisites

The following items must be addressed prior to performing this procedure:

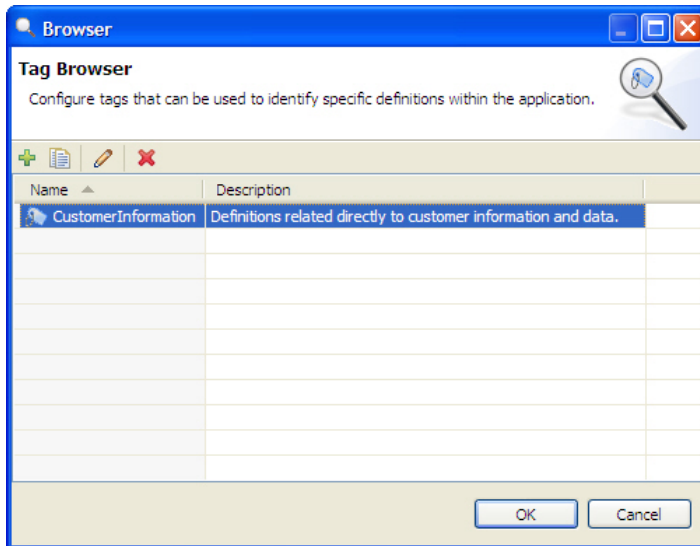
- The Agentry application project must be open in the Agentry Perspective.

Task

This procedure describes how to create new public tags within an Agentry application project. When complete a new public tag will exist within the project and be applied to that project's definitions.

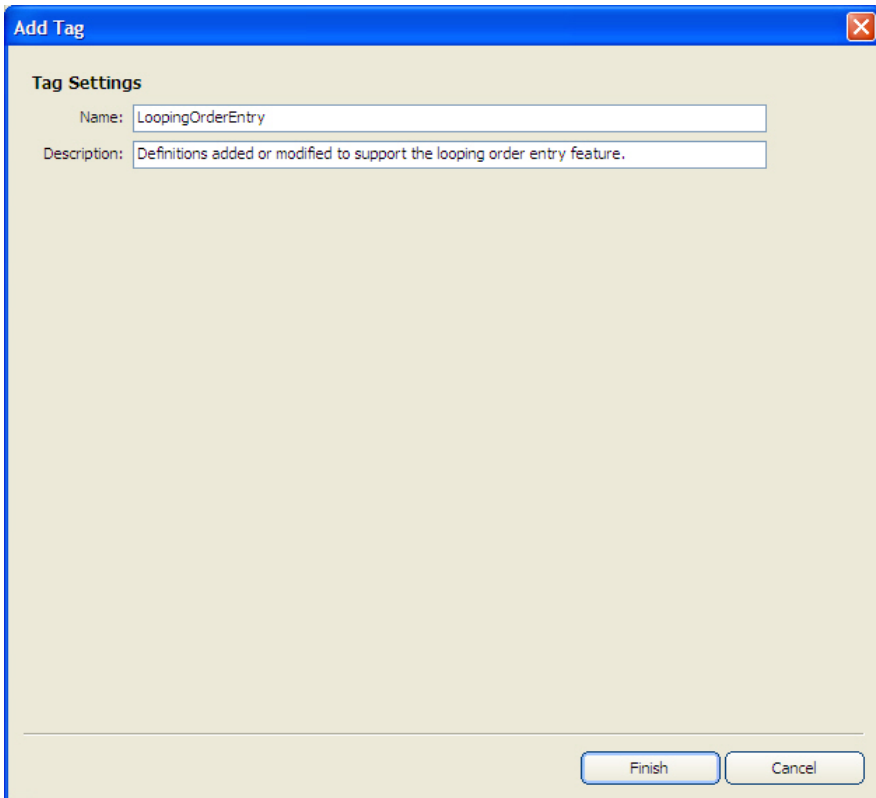
1. Begin by clicking the tag button  in the Properties View for any definition.

This displays the Tag Browser screen listing all current public tags for the project. This can also be used to edit an existing tag or delete a tag from the project:



2. Click the add button  above the list of public tags.

This displays the Add Tag wizard screen:



The image shows a dialog box titled "Add Tag" with a blue header bar and a close button in the top right corner. The main area is light beige and contains a section titled "Tag Settings". Below this title are two text input fields. The first field is labeled "Name:" and contains the text "LoopingOrderEntry". The second field is labeled "Description:" and contains the text "Definitions added or modified to support the looping order entry feature." At the bottom right of the dialog box, there are two buttons: "Finish" and "Cancel".

Add Tag

Tag Settings

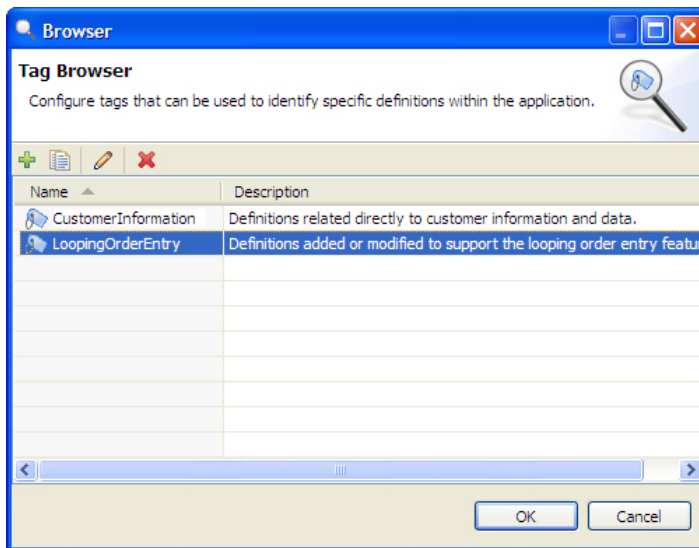
Name: LoopingOrderEntry

Description: Definitions added or modified to support the looping order entry feature.

Finish Cancel

3. Enter a name and description for the new tag. Click the **[Finish]** button when complete to create the new tag for the project.

The tag is added to the project and listed in the Tag Browser:



4. To immediately add this tag to the currently selected definition in the project, double-click it in this list. To close the Tag Browser screen click the [OK] button.

A new public tag has been added to the Agentry application project. This tag can be applied to definitions manually or via the auto-tagging feature.

Next

The tags name and description can be edited by returning to the Tag Browser at any time and editing the selected tag in the list. Edits update all definitions to which the tag has been previously applied. The tag can be deleted from the project in the tag browser, removing it from all definitions to which it was previously applied.

Tagging: Applying Public Tags to Definitions

Prerequisites

The following items must be addressed prior to performing this procedure:

- The Agentry application project containing the definitions to be tagged must be open in the Agentry Perspective.
- The tag to apply must exist within the Agentry application project.
- The tag cannot currently be applied to the definition.

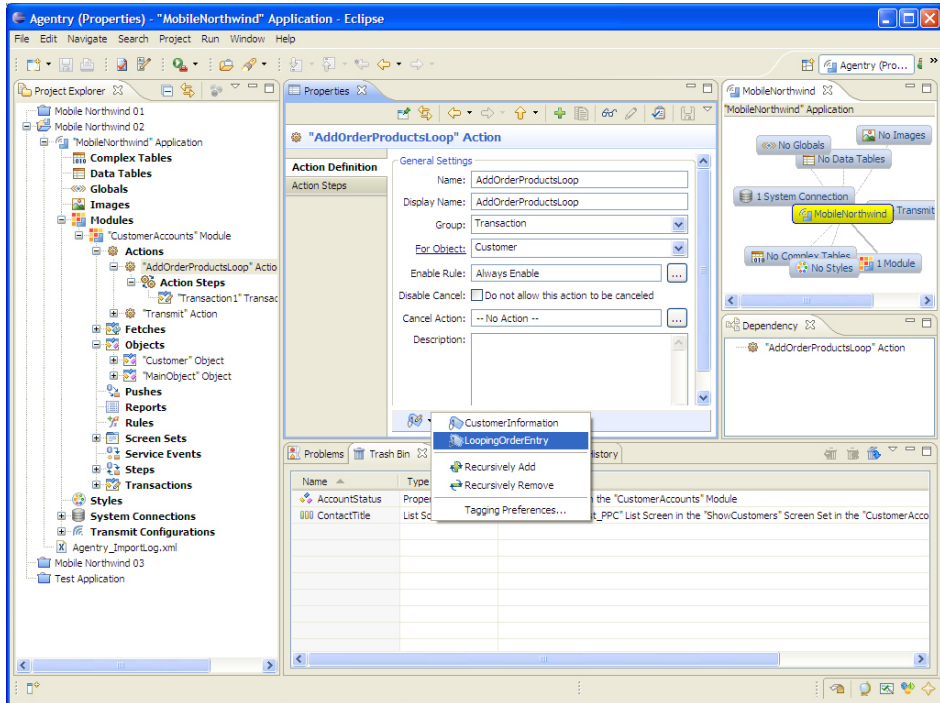
Task

This procedure describes the steps necessary to apply a public tag to a definition within the Agentry application project. It also describes the process of recursively applying the same tag

to multiple definitions. When this procedure is complete the definition(s) will include the selected tag and can be organized or selected in various operations by this tag.

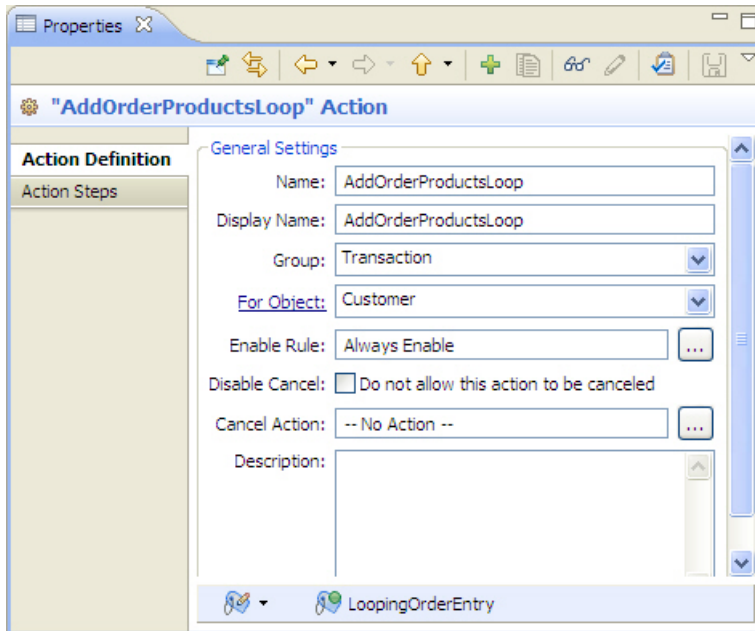
1. Display the definition to be tagged in the Properties View. Click the down arrow for the tag button.

This displays a menu of options related to tagging the current definition and includes a list of all public tags for the current project:



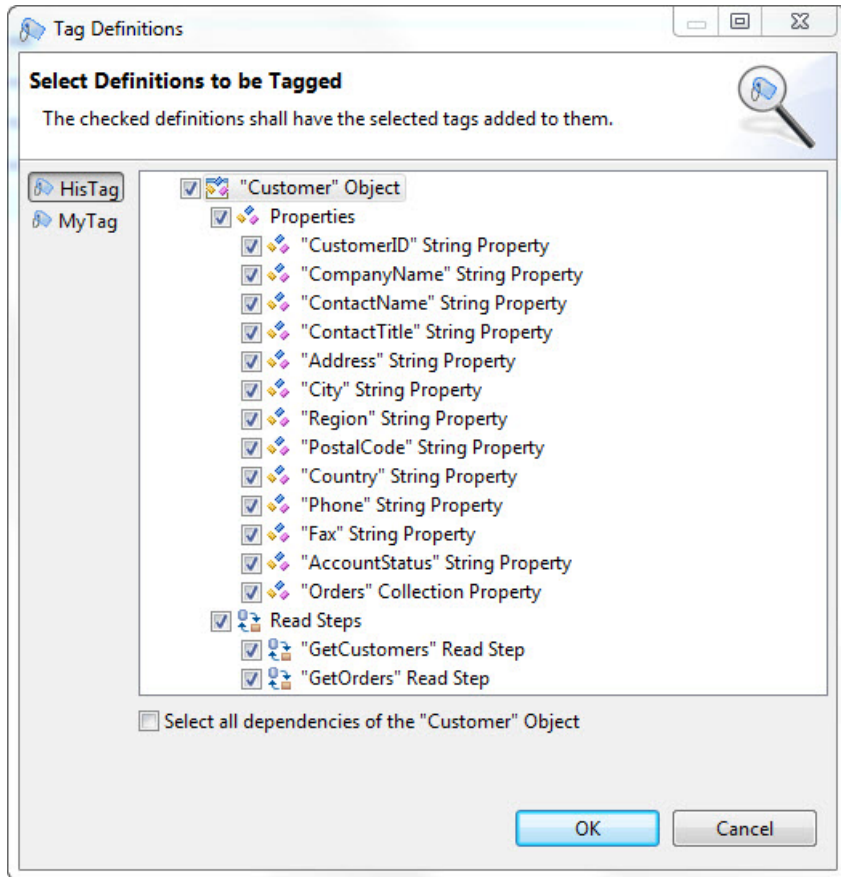
2. A tag can now be applied to the current definition, or to the current definition and all its descendents.
 - a) To apply a tag to the current definition only, select it in the menu.

The tag is now applied to the current definition and displayed in the Tag Bar at the bottom of the Properties View.



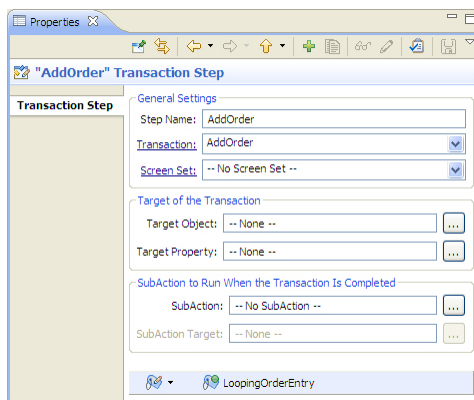
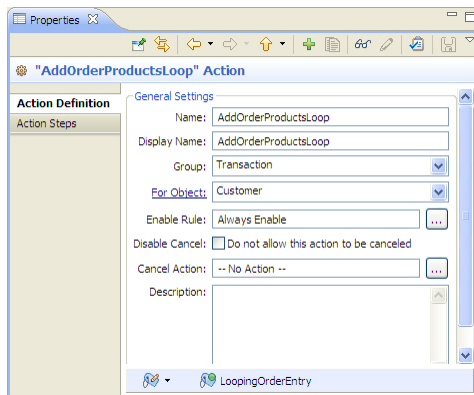
- b) To apply a tag recursively to the current definition and all its descendents, select the menu item **Recursively Add**.

This displays the Tag Definitions screen listing the available tags on the left and the definitions under the current definition. Below the list of definitions is the check box to Select all dependencies of the current definition. Selecting this option will tag not only all descendents of the current definition, but also all definitions dependent on it, i.e., those definitions that reference it in some way. Note that this same screen is displayed when choosing to recursively remove tags:



- c) Select the tag to apply on the left of the screen. In the tree control, the default selection is the current definition and all its descendents. Definitions can be unchecked to not apply a tag to that definition. Once the tag and definitions are selected, click the [OK] button to apply the tag recursively.

The selected tag(s) is applied to all the selected definitions. The Properties View for all selected definitions now displays the tag in the Tag Bar at the bottom of the view:



The selected definition or definitions now contain the selected public tag or tags. These public tags are displayed at the bottom of the Properties View for all affected definitions.

Next

The tags can be selected in export operations to select the definitions to export by public tag. Commit operations to a share repository will include this tag information.

Introduction to Team Configuration

With the release of the Agentry Mobile Platform version 5.2 a new feature set collectively called Team Development has been implemented. These features are provided to support multiple developers performing work on the same application project. These features include the following:

- A common share repository for storing work from multiple developers and capable of tracking multiple revisions of a given project
- New import behavior related to the share repository
- Extended export functionality supporting the export of definitions below the module level

- Definition tagging for various organizational purposes
- Replacement of the Compare Dialog with the Compare View, which includes additional functionality related to the share repository

The general approach of team development is to provide a central share repository to all developers working on a common application project. Each developer then creates a project within the local Eclipse workspace based on the contents of the share repository. Developers then modify their local versions of the project and periodically commit their changes to the share, and also update their local projects from the share repository.

In support of this workflow, several new operations have been added to the Agentry Editor specifically for working with the share repository. Additionally the import operation has been augmented to support working with a share repository. Also, the concept of definition tagging has been added to the Agentry application project and Agentry Editor. The Agentry application project itself can be connected to a share, which then allows for the tracking of changes and revision history, and the share repository operations to update and commit from and to the repository.

The following sections provide overview information on the different functional areas and concepts related to the team configuration behaviors. Each of these is covered in more detail in subsequent sections, including instructions and requirements where applicable.

Share Repository

At the center of the team development architecture is the share repository, or simply “share.” This share is placed in a location common to and accessible by all developers on a team. Changes can be committed to this share from each developer’s local application project and from this share other developers can then update their local projects to retrieve committed changes.

The location of the share repository must be one that is accessible to all developers on a team, and should also be one that is backed up in some manner, usually via a version control or source control system. This location is typically a file server common to all members of the development team and to which each member has read-write privileges.

Stored within this repository is the common project share by the development team. Each developer has a local Agentry application project within the Eclipse workspace created from this share. Developers can work with the local project, defining behaviors just as with any project. When a stable point is reached in the development work, the developer commits the changes made back to the share. From here, other developers update their local projects from the share. The share itself maintains multiple revisions of the project, one for each commit. From this share a developer can update to the latest, or “tip” revision, as well as to earlier revisions when necessary.

Share Repository Compression

The compression of the share repository is a behavior added in version 6.0 of the Agentry Mobile Platform. This is the standard behavior of both creating a new share repository, and when committing new revisions to an existing repository. For share repositories created with a

version of Agentry prior to 6.0, the existing revisions in the repository will remain uncompressed and subsequent revisions to it will be compressed. Once such a revision is committed, the entire repository can only be used by versions of the Agentry Editor (6.0 and later) that support compressed share repository revisions.

Share Repository Operations

Operations related to the share repository are begun by right-clicking the root project node in the Project explorer view. This must always be the open Agentry application project. In the context menu displayed there is a sub-menu **Team**. Within this menu there are several operations related to the share repository. If the open project is not currently connected to a repository, the only option available is to create a new share. This operation should only be performed when creating a new share for other developers to connect with, typically at the beginning of the project.

If no project is currently connected to a share, then a new project can be created from a share using the Import operation, which is separate from those in the Team menu. This is the same Import operation as others previously available. The selected import source is a share repository available to the Agentry Editor.

Once a project is connected, the Team sub-menu for the open project includes several options. Note that the first menu item is Apply Patch... This item is not a part of the team development functionality for an Agentry project and does not apply to these discussions.

The remaining menu options do apply and include the following:

- **Commit:** A commit operation updates the local Agentry project to the share. Changes between the local and share projects are determined, and the new or changed definitions from the local project are updated to the share. Likewise, definitions removed from the local project are removed. A new revision is created in the share as the tip revision for others to retrieve via an Update operation.
- **Revert:** The revert operation allows for the local project to be reverted to an earlier version of the project within the repository. Note that a commit of any local changes must be performed before the local project is reverted. When a revert operation completes, the local project matches the selected revision in the repository.
- **Update:** The Update operation updates the local project to the tip revision in the repository. As a part of this update process checks are made for conflicts between the local project and the tip revision in the share. When conflicts exist the developer is informed and given options on resolving them.
- **History:** The history menu item opens an additional History View within the Agentry Perspective. This view lists all revisions within the repository. Each item in the list includes the revision number, the description entered when that revision was committed, and the author and date and time of the commit.
- **Disconnect:** The disconnect operation removes the link between the local project and the share. Changes made subsequent to a disconnect operation are not tracked in relation to the share. Updates and commits can no longer be performed.

Update Conflicts and Resolution

When working with a share repository the possibility exists that changes committed to that share and changes made by individual developers conflict with one another. These conflicts become prevalent when a developer attempts to update the local project from the tip revision in the share.

The processing logic within the update operation includes checks for such conflicts. These items are then either resolved automatically by the update operation or are noted by the operation and the developer is informed. In the latter case, the developer is presented with options to resolve the issue. The specific options depend on the nature of the conflict.

Whether the conflict is handled automatically or via manual intervention by the developer, the goal of the operation is to update the local project to match the tip revision, or to modify it in such a way that the next commit performed by the developer updates the share so that no further conflicts exist.

Import

The import operations have been augmented in the 5.2 release of Agentry with the addition of a new import source that is the share repository. This source is selected in an import operation to create a new local Agentry project based on the tip revision of a selected share. When completed a local project is added to the Eclipse workspace that matches the tip revision in the share repository.

Within the import wizard the share repository is selected. Note that this operation always creates a new project. Imports cannot be performed from a share to an existing project. Rather, this operation requires the developer to perform an update operation for a project already connected to a share.

Tagging Definitions

Any definition within the application project can have one or more tags added to it. Tags are created by developers within the project and include a name and description. Tags do not affect the mobile application at run time.

Certain operations, such as exporting, allow for definitions to be selected by the associated tags. As an example, if implementing a new behavior in an existing project, the definitions added or modified to support that behavior can all be given the same tag. During an export, the developer can easily select all these definitions to be exported by simply selecting that tag, which is displayed in the export wizard. In support of such behavior it is possible to enable auto-tagging. This feature allows the developer to select one or more tags to be automatically applied to definitions are modified or added.

The impetus for implementing tags in the 5.2 release of Agentry is in support of the team development functionality. However, they are not tied solely to this functional set and can be used in any manner found useful by the developer.

Team Configuration: Share Repository Requirements and Operations

The share repository, or simply “share”, is the central component to the Team Configuration functionality available as of the 5.2 release of Agentry. The share is the common project storage location of work performed by all developers for a single Agentry application project. When working with a share the Agentry project must be connected to that share. This then links the project with the share, tracking the changes made locally with the project as it exists in the share repository.

Share Repository Requirements and Details

The basic requirements for a share repository are that it be stored in a location to which all developers on the team have read-write access to that location. Typically this is a common file server or equivalent that is accessible to all developers and is linked to each developer’s workstation as a mapped network drive in Windows. The directory in which the share is placed must exist prior to sharing the project. Multiple share repositories cannot be created in the same base directory. However, multiple shares can have a common ancestor directory.

As an example, a share can be created in the directory M: \SharedProjects \MobileNorthwindCRM. A second share can be created in the folder M: \SharedProjects \MobileNorthwindInventory. However, it is not allowed to create two shares in the directory M: \MobileNorthwindApps.

Each developer will perform operations related to the share that include reading from and writing to the share’s directory, and therefore each must have permissions to perform these operations on all files within the share location.

When a share is created (see “Creating a Share Repository” for details) a local project is first selected. A directory is then selected to store the share. Within this directory a file named `share.ini` is created containing information about the share. This file should never be manually modified unless directed by a Syclo support specialist. When checking out from a share to create a new local project based on that share’s tip revision, the `share.ini` file is selected as a part of that operation.

The initial revision in the share will then be the definitions in the selected local project. These items are written to the directory named 1. Subsequent commits to this share for developers create additional directories, each numbered to match the share revision created by that commit operation, e.g. 2, 3, 4.... As with the `share.ini` file, the contents of these sub-directories should never be modified manually unless under the specific direction of a Syclo support specialist.

Share Operation: Share Project

The Share Project operation is the first step in creating a team environment for a common Agentry application project. This operation creates a share repository at a designated location. The new share contains a single, initial revision, i.e., revision 1. The contents of this revision match the contents of the local Agentry application project open within the Agentry

Perspective when the share project operation is executed. This operation can only be executed on Agentry application projects not currently connected to a share repository.

The new share should be created in a location common to all developers on the team and according to the requirements of the share repository. The local project should be in a state in which it makes sense to share the project contents. This state will vary from one project to the next and depends on the division of work among the developers. The only requirements from a technical standpoint are that an Agentry application project exists within the Eclipse workspace and that project is open. No validation or check on publish is performed as a part of the share project operation. This means the project need not be in a publishable state prior to creating the share.

In practice, it is likely desirable that some useful functionality exist prior to creating the share. In many use cases the first revision of the share is the standard implementation of a product application, such as those provided by Syclo. For new application projects the functionality need not be nearly as robust, or even completely implemented before creating the share. Rather, the core pieces to the project, such as objects and their properties, fetches and pushes that may or may not yet contain step usage definitions, and screen sets with or without platforms or screens may all be a part of the initial revision of the share when created.

Typically when planning and creating a share, the developer responsible for creating these core definitions should perform their initial work (though it need not be the final planned result) and then create the share from their local project. This developer, as well as the rest of the team, can continue to work with the definitions once the share is created.

Share Operation: Checkout (Import from Agentry Share)

Once a share is created, other developers can access its contents for their own portion of the work for the project. To begin this work the developers will each need to check out the tip revision of the share repository. This is performed via the Import operation. During this operation the import wizard is displayed, the first screen of which provides the developer with the list of choices for the import source. One of these options is **Agentry Share | Checkout Project from an Agentry Share (share.ini)**. Selecting this option indicates a new project is to be created in the local Eclipse workspace by checking out the tip revision from the share repository.

Once the share is selected as the source of the import, and other information is provided, the import operation creates a new Agentry application project within the workspace by importing the definitions within the share's tip revision. When the operation is complete, the developer can modify and extend this project for their portion of the overall implementation. Typically a checkout is performed only once to create the local project. After this point, the developer performs commit operations to commit changes made to the local project to the share; and update operations to retrieve changes committed to the share by other developers.

Share Operation: Commit

When working with a project connected to a share, the developers on a team must perform the commit operation to commit changes made in their local projects to the share repository. A

commit operation results in the addition of a new revision to the share. This new share then becomes the tip share that other developers receive when performing updates until a subsequent commit is performed by any developer connected to the share.

During the commit operation, the wizard screens displayed include a comments field. Within this field comments are automatically added to note all changes made to the local project. The comments reflect the type of change made, which can be add, edit, or delete, and the definition modified. These default comments can be edited prior to performing the actual commit. The final contents of this comment field are then the comments for the revision created by the commit, and will be viewable by all developers in the History View.

When performing a commit, the Agentry Editor first checks the local project for changes as compared to the share's tip revision. If no changes exist, the commit will not be performed. The commit wizard's OK button is disabled. The summary view within this wizard indicates no differences exist between the local project and current tip revision.

Another of the share operations is revert. Using this operation it is possible to revert the local project to a share revision earlier than the tip revision. When the local project is reverted to a previous revision and subsequent changes are made to the local project, a commit operation will display a warning message indicating the difference in revisions. Note that the current state of the local project will be committed to the share as the new tip revision. Any changes made and committed to the share between the reverted revision and the current tip revision will be lost when the new tip revision is committed. For this reason, reverting to a previous revision and then committing should only be performed in rare circumstances.

Share Operation: Update

The update operation is performed by the developer to update the local Agentry application project to the tip revision of the share repository. This allows the developer to retrieve changes made by other developers working on the same project. During an update a check is first made for differences between the share revision and the local project. If changes exist for the same definition in both the local and share projects, a conflict exists. This requires manual resolution by the developer. The specific behavior of the update and the resolution depend on the nature of the conflict. When a conflict does occur, and there are other definitions in the share that should be imported that are not in conflict with the local project, those definitions are updated to the local project, leaving only the conflicted definitions in need of resolution. See the information on "Update Conflict Resolution" for details.

Share Operation: Revert

The Revert Operation for a share repository replaces the local project with the specified share revision. This revision can include the tip revision when the local project is at an earlier revision. The difference between a revision and an update is the, first, a specific revision can be selected, and, second, there is no conflict detection performed. This last revert behavior is important to note as it means that any uncommitted changes made to the local project are lost when the revert operation completes. One use for the revert operation can be to remove unwanted changes from the local project.

Note that while the tip revision can be selected in a revert operation it should never be used in place of the update operation. Reverting to the tip revision should only be performed when it is desired to remove all local changes. In such situations the developer should be careful to verify all local changes should be removed before proceeding.

As an option to a revert operation it is possible to create a new local project based on the selected share revision. This can be useful when wanting to branch development from an earlier revision of the repository for a separate development effort. In such a situation, the proper overall procedure is as follows:

1. Execute the revert operation, selecting the earlier revision from the share, and selecting the option to create a new local project.
2. The new project is connected to the share repository from which it was imported. Disconnect from this share.
3. To support team development with the new local project, create a new, separate share by performing the share project operation.

Share Operation: Show History

The Show History operation does not affect either the local Agentry application project or the share repository. This operation opens the History View within which each repository revision is listed. The revision of the repository from which the last update to the local project was performed is highlighted.

Within this view the revision number, the date it was created, the user that created it, and the revision description as entered during the commit operation are listed. This is a read-only view intended to provide information about the local project as it relates to the share, as well as information about the share itself. This view can be refreshed at any time and will display any new revisions added to the share since the last refresh.

The history view should be reviewed prior to performing commit or update operations to understand the current state of the share before changes are made to it or the local project.

Share Operation: Disconnect

The Disconnect Operation disconnects the local Agentry project from the share repository. Once a project is disconnected it is no longer tied in any way to the share. Subsequent changes to the local project are made out of synch with the share. Such changes are not privately tagged by default.

The disconnect operation should only be performed on a project that should no longer be a part of the team efforts. This may be useful when it is desired to retrieve a project from an existing share by performing a checkout but for which changes to that project should not be included in the share. This procedure would involve the following steps:

1. Perform an import from an existing share repository, create a local Agentry application project, which is currently connected to the share. The project contains the tip revision from the share.
2. If a previous revision from the share is desired, revert the local project to that revision.

3. Once the desired share revision has been imported into the local project, disconnect the local project from the share.
4. Optionally, create a new share from the local project to support team efforts.

This is only one scenario for disconnecting a project from a share. Others may exist, and the operation can be performed to meet any needs found by the developer.

Update Conflicts and Conflict Resolution

When performing an update operation the local project is updated to match the tip revision of the share repository. When the update is performed, however, it is possible the state of the local application project is in conflict with the state of the share's tip revision. This occurs, potentially, when the same definition is deleted or modified in some manner in both the local project and share.

During the update operation, the Agentry Editor checks for any conflicts. If found, it may be necessary for the developer to manually resolve these conflicts. This manual resolution is performed in the Comparison View in the Agentry Perspective. This view is displayed whenever an update operation is performed. If a conflict is found, a message is displayed after the update is finished indicating there is an issue. Any definitions in the tip revision not in conflict with the local project are imported. Those in conflict are not imported, but rather are highlighted as differences in the comparison view.

This manual conflict resolution is similar in behavior to a compare and import from some other import source, such as an export file. The local project and the share's tip revision are displayed in the comparison view, with the share on the right side as the comparison source. The developer can then select the individual definitions within this view to be imported from the share, or leave them as is within the local project. This selection can be different for each definition found to be in conflict.

When the conflict resolution is completed by the developer, the definitions selected in the share are imported into the local project. Any conflicted definitions not imported from the share are left unchanged in the local project. At this point a commit can be performed and those definitions not imported from the share are committed to the share as a part of the new revision.

The nature of the conflict and how it should be resolved depends on the type of changes made to the definitions in both places. The following list describes the potential conflicts requiring manual resolution by the developer. This list is then followed by sections describing the resolution choices for each:

- **Share definition edited - local definition edited:** In this situation, both the share and local definitions' attributes have been modified in some manner. Since the nature of the change to both may not be compatible in one manner or another when merged, this change is left to the developer to resolve.
- **Share definition deleted - local definition modified:** If the local definition has been edited by the developer, and the same definition is deleted from the share revision, the

developer must specify whether to keep the local definition, or to import the deleted definition from the share.

- **Share definition deleted - local definition modified and deleted:** If the share definition has been deleted (contained in the trash bin) and the local definition was modified and then deleted, a conflict exists. While the share definition is simply added to the trash bin if brought down during the update, there is a question as to which version of this definition should be stored in the trash bin for possible later recovery, the local version or the share version. Therefore, the developer is required to make this selection.

Share Definition Edited - Local Definition Edited

If both the local definition and the definition in the share have been edited, a conflict exists and must be resolved manually. Edited share and local definitions includes changes to the definitions' attributes, whether or not it is the same attribute. Changes to child definitions are not considered conflicts.

When such a conflict occurs, the developer must manually specify in the Comparison View which definition to keep. If the share definition is selected in this view, it will be imported into the local project, replacing the local definition. If the share definition is not selected, the local definition will remain. During the next commit from the local project, the local definition that was in conflict will be treated as a changed definition and committed to the share.

Share Definition Deleted - Local Definition Modified

If the definition in the share has been deleted, and the local definition has been modified, a conflict exists. Any change to any attributes in the local definition constitute a change. The share definition is considered deleted if it has been removed but remains in the trash bin. Permanently deleted definitions in the share repository, that is, those that have been removed from the trash bin, and those that have been edited in the local project are not conflicted.

Share Definition Deleted - Local Definition Modified and Deleted

If the share definition has been deleted and resides in the trash bin, and the local definition has been modified and then subsequently deleted prior to committing, a conflict exists. In this situation, the developer must compare the two definitions and determine which should reside in the local trash bin.

If the share definition is selected in the Comparison View, the local definition in the trash bin is replaced with the share definition. Otherwise the local definition remains in the trash bin and will then be added to the next commit performed from the local project.

Automatically Resolved Conflicts

Additional conflicts may occur during an update from the share's tip revision that will not require intervention on the part of the developer. These may not even be considered conflicts, but rather the behavior of the update operation under certain conditions other than when the local definition has not been modified and the share definition has.

The following list describes these situations and the resulting behavior of the update:

- **Share definition deleted in trash bin - local definition not modified:** If the share definition has been deleted and currently resides in the trash bin, and if the local definition has not been modified in any way and matches the share definition, the local definition is removed from its parent and placed in the trash bin.
- **Share definition deleted - local definition deleted, not modified:** If the share and local definition have both been deleted, and if there is some difference with the local definition that is not tagged, meaning it was not made since the last commit, the share definition will replace the local definition in the trash bin.
- **Share definition deleted - local definition does not exist:** If the share definition is deleted and resides in the trash bin, and there is not corresponding local definition, the share definition is added to the local project and resides in the trash bin.
- **Share definition does not exist - local definition exists, not modified:** If the local definition exists and has not been modified, and the share does not have a corresponding definition, the local definition is not modified or removed. This is listed as a difference in the Comparison View and can be removed here or by simply deleting the definition. If the definition is not removed from the local project it will be added to the share during the next commit operation.

Share Operation: Creating a Share Repository

Prerequisites

The following items must be addressed prior to performing this procedure:

- The directory in which the new share will be placed must exist prior to creating the share.
- The directory for the share must be empty.
- The directory should be in a location accessible by all developers on the team. This includes both read and write access to the directory and its contents.
- The Agentry application project from which the share is to be created must be the open project in the Agentry Perspective.
- The Agentry application project should be in a state where it makes sense to share with other developers on the team.
- The Agentry application project cannot be connected to a share.

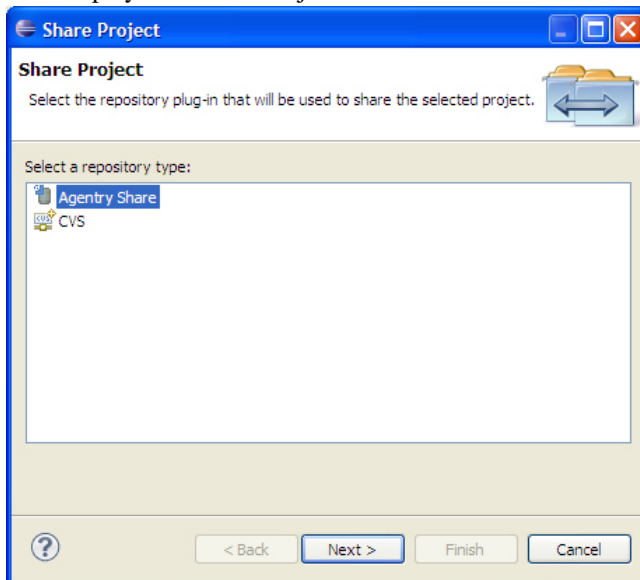
Task

This procedure describes the steps involved in creating a new share repository. When this procedure is complete, a share will be created with the initial revision. The contents of this revision will match the current state of the Agentry application project from which the share is created. This local project will be connected to the new share. The share will be available to others for import to create local projects based on the initial revision. Going forward the share will support all share operations by developers with access to the share.

1. Open the source Agentry application project in the Agentry Perspective within Eclipse.

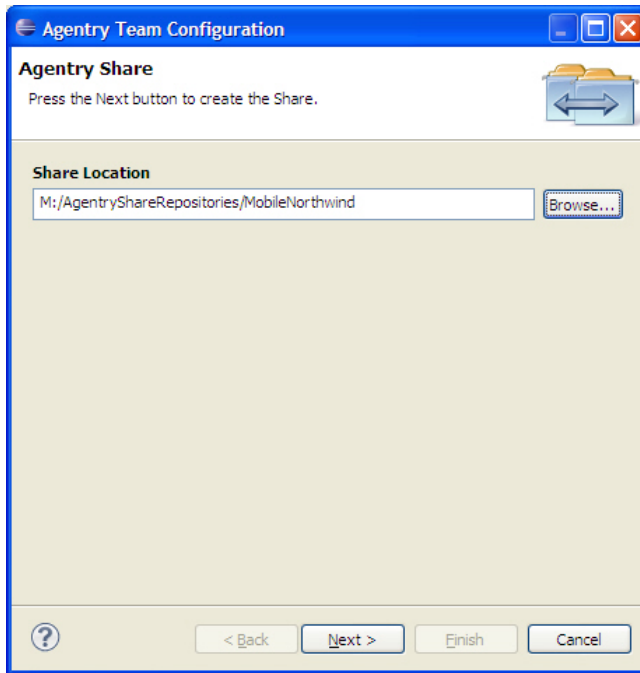
2. Right click the root project folder in the Project Explorer View. In the context menu select the item **Team | Share Project...**

This displays the Share Project Wizard:



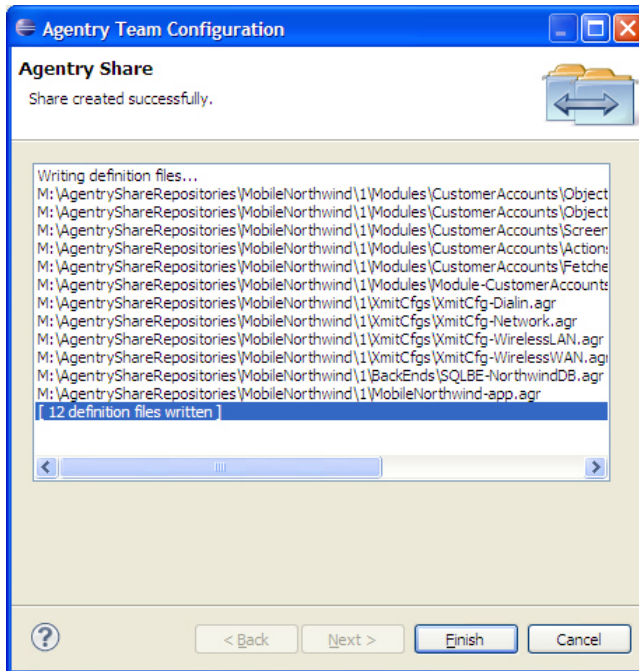
3. Select the item Agentry Share from the list displayed in this screen. Click the [Next >] button.

This displays the Share Location screen:



4. Enter the location of an existing directory accessible to the developers on the team and meeting the requirements of an Agentry Share Repository. Click the **[Next >]** button.

This creates the new share repository at the designated location. The status screen displays the definitions as they are added to the initial revision of the share:



5. Click the [Finish] button to close the wizard and return to the Agentry Perspective.

When this procedure is completed, the new share repository is created. The initial revision of this share matches the current open project in the Agentry Perspective. This local project is connected to the newly created share repository. The share is now available to other developers with read-write privileges to the selected location from their Agentry Editors.

Next

Developers with access to the share can import the initial revision, or the current tip revision, creating local Agentry application projects. Work performed by all developers can then be committed to this share, making it available to all others on the team.

Share Operation: Checking Out (Importing) From a Share

Prerequisites

The following items must be addressed prior to performing this procedure:

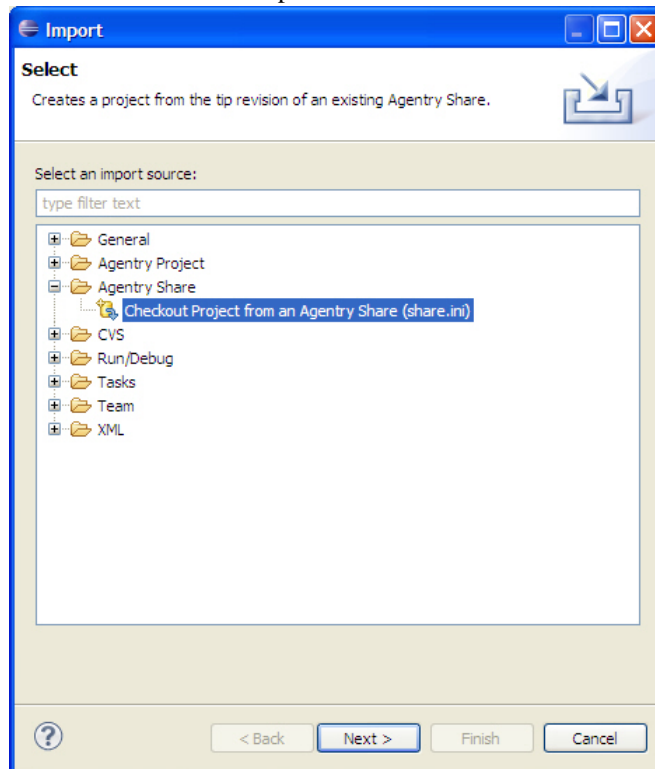
- The Agentry Perspective must be open in Eclipse.
- The developer performing the check out must have read-write privileges to the share location.

Task

This procedure describes the steps to check out the tip revision of a share repository, creating a local Agentry application project in the current Eclipse workspace. When this procedure is completed, a new Agentry application project will exist in the workspace and will be connected to the share from which it was checked out. This procedure is similar to an import performed with a non-share source, such as an export file or published application on an Agentry Server. However, the import source for this procedure is an existing share repository.

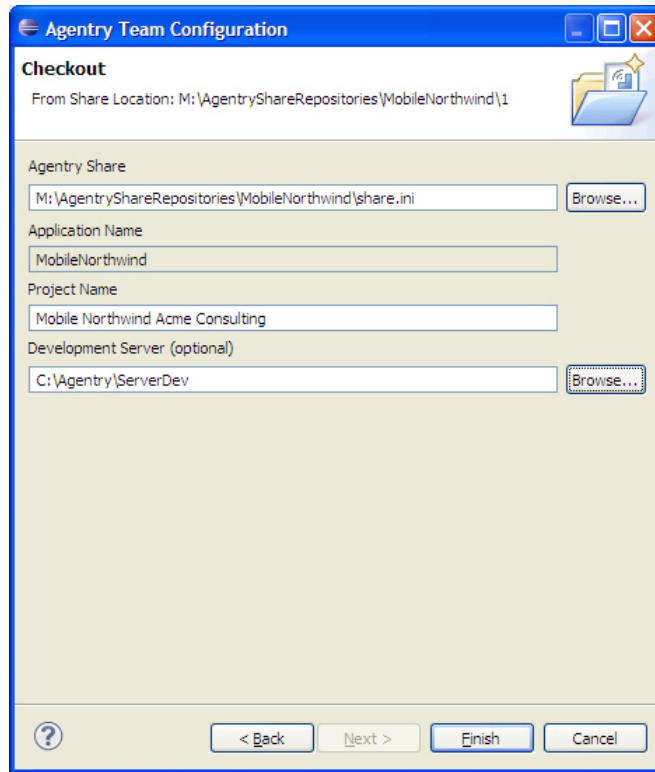
1. Right click anywhere in the Project Explorer View within the Agentry Perspective. Select the menu item **Import...**

This displays the first screen of the Import Wizard:



2. Within this wizard screen the import source is selected. To check out from a share, select the item **Agentry Share | Checkout Project from an Agentry Share (share.ini)**. Click the **[Next >]** button.

This displays the second screen of the wizard:



3. Within this screen set the fields according to the following instructions and then click the [Finish] button:

- **Agentry Share:** Enter the path to the share repository, including `share.ini` file. This can be entered manually or selected in a Windows File Dialog by clicking the [Browse] button and navigating to the share location and selecting the `share.ini` file.
- **Application Name:** When checking out of a share, this field is read-only. It is set to match the Application Name attribute in the share's tip revision.
- **Project Name:** This is the name given to the project in the Eclipse workspace. This name can be any unique value and is not committed back to the share.
- **Developer Server (optional):** Enter the path to the Agentry Development Server for the local project. This path can be entered manually or selected in a Windows File Dialog by clicking the [Browse] button and navigating to the Server's installation directory.

The new project is created by importing the definitions from the selected share's tip revision. The wizard is closed and the project is displayed in the Project Explorer View.

When this procedure is complete a new project is created in the current Eclipse workspace. This project contains the definitions matching the selected share's tip revision. The project is automatically connected to the share from which it was imported.

Next

The new project can now be modified by the developer. Changes made can be committed to the connected share and updates retrieved from it.

Share Operation: Committing Changes to the Share Repository

Prerequisites

The following items must be addressed prior to performing this procedure:

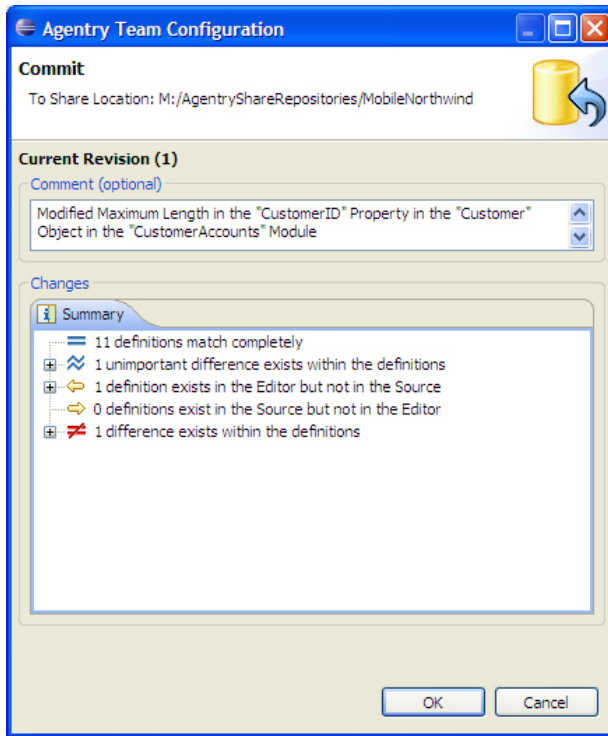
- The current Agentry application project must be connected to a share repository.
- The Agentry application project should be in a state believed to be stable and/or ready for integration into other developers projects.
- Any desired public tags should be applied prior to performing the commit.
- Any additional information needed for the revision about to be created should be noted in preparation for adding such information to the revision's comments during this process.

Task

This procedure describes the steps involved in committing changes made to a local Agentry application project to the share repository to which the project is connected. When complete, all definitions modified in the local project since the last commit will be added to the share repository as a new revision. This revision will be the tip revision of the repository received by other developers during subsequent updates up until another revision is committed.

1. Right click the open project in the Project Explorer View. In the context menu now displayed select **Team | Commit**.

This displays the Commit Wizard:



2. This screen displays, first, the comment field for the new revision and, second, the summary of changes made to the local project. Edit the comments as necessary. Click the [OK] button to proceed.
3. The definitions that have been modified in the local project are updated to the share as a new revision.

When this procedure is complete a new revision is added to the repository as the tip revision. This revision includes the local Agency application project definitions as they existed at the time of the commit. Any private tags on definitions resulting from modifying those definitions have been removed from the local project.

Next

When the new revision has been created, other developers can retrieve the changes and integrate them with their local projects by performing update operations.

Share Operation: Updating From the Tip Share Repository Revision

Prerequisites

The following items must be addressed prior to performing this procedure:

- The tip revision within the share repository must be newer than the local Agentry application project's revision.
- The current Agentry application project must be connected to a share repository.

Task

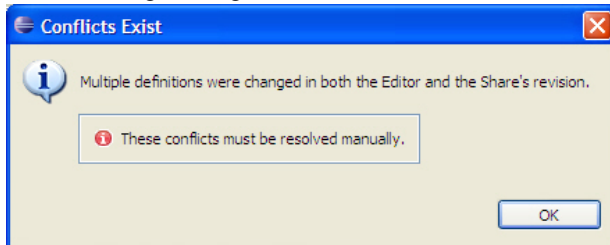
This procedure describes the steps necessary to update the local Agentry application project to the latest, or “tip” revision in the share repository to which the local project is connected. When this procedure is complete, the local project will be updated to match the application definitions in the share's tip revision. As a caveat to this result, if there are conflicts between the tip revision and the local copy of the definitions, it is possible the resulting local copy will differ from the tip revision. These differences can result from how any conflicts are resolved during the update, which can include keeping the local version (which would differ from the share's tip revision) or merging the differences between the local copy and tip revision.

1. Right click on the local project in the Project Explorer View. Select the menu item **Team | Update** from the context menu.

The update from the share begins immediately. If there are no conflicts with the share and the local project this procedure is complete. The local project is now updated to match the tip revision in the share.

2. If one or more definitions between the share and local projects are in conflict, a message is displayed indicating there is an issue. Also, the Comparison View is opened and the definitions in conflict are highlighted as differences within this view.

The following message indicates there are conflicts:



3. Resolve the conflicts using the Comparison View (opened automatically during the update) to manually import from the share or keep the local version of the definition. In the latter case, these local definitions will be a part of the next revision in the share. For further information on conflicts and resolving them during an update, see the information on “Update Conflicts and Conflict Resolution.”

When this procedure is complete the local project is updated to match the tip revision in the share. If any conflicts occurred, and those conflicts have been resolved, the local project may contain differences from the share's current tip revision. Those differences will be a part of the definitions committed to the share the next time a commit operation is executed from the local project.

Share Operation: Reverting to a Previous Share Revision

Prerequisites

Prior to performing this procedure the following items must be addressed:

- The current Agency application project must be connected to a share repository.
- It must be determined if the revision to be reverted to should replace the current project, or if a new project should be created with the previous share revision to be selected.

Note: Selecting to replace the current project with a previous revision can lead to undesirable results, including the loss of all local, non-committed changes in the local project. Replacing the existing project should only be performed after a back-up of the local project has been made. This can be accomplished by either committing the project to the share, or exporting the project to an Agency export file by executing the Export Operation.

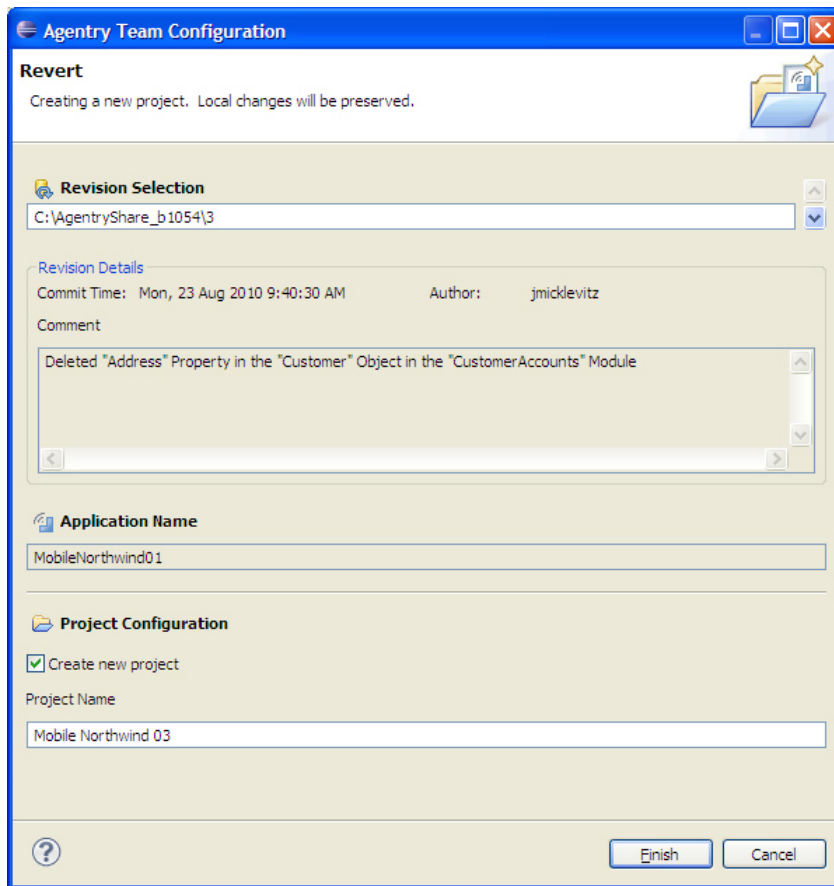
Task

This procedure provides the steps necessary to revert the local project to a previous revision within the share repository; or, alternately, to create a new local project in the current Eclipse workspace based on a revision of the share prior to the tip revision. When this procedure is complete, and depending on the options selected, either the current local project will be replaced with the revision selected in the share repository, or a new local project will exist containing the definitions as they existed in the selected revision.

Note that this procedure can also be performed to revert the local project to the tip revision in the share repository. This differs from an update in that the current local project will be reverted to the tip revision and any local changes will be lost. None of the conflict detection or resolution functionality that is a part of an update operation is performed in a revert operation.

1. Right-click the open Agency application project in the Project Explorer View. Select the menu item **Team | Revert | *revision***, where *revision* is the share revision to which the project should be reverted.

This displays the Revert Project wizard, with the selected revision's information displayed



2. To create a new local project based on the selected share revision, check the **Create new project** check box. Then enter a name for this new project in the **Project Name** field. To not create a new project, but instead replace the currently open local Agency application project, leave this box unchecked. Click the **[Finish]** button to proceed.

The revert operation now executes. Based on the selections made, either the local Agency project is replaced with the selected revision, or a new project is created in the current Eclipse workspace matching the selected revision from the share.

When this procedure is complete, either the local Agency project is reverted to the selected revision, or a new local project is created in the Eclipse workspace matching the selected revision. If a new project has been created it is connected to the share repository.

Next

The resulting project from this operation now matches the selected revision within the share. This project cannot be committed to the same share. Either the local project must be updated or

reverted to the tip revision, or it must be disconnected from the current share and new one created for the local project using the Share Project operation.

Overview of Mobile Northwind Sample Application

In the development guide for Agentry applications there is reference to the Mobile Northwind sample application. An overview of this application is provided here, including data structure, client behavior, and synchronization components.

The Mobile Northwind application is a basic order entry application with some light customer relations management and inventory-like functionality included. It is an extension of the Northwind sample database provided with MS SQL Server systems, a database for the fictitious company Northwind Trading.

Module Data Structure and Object Collections

The data definitions within the application include objects and related transactions, a complex table and a data table.

The object definitions include Customer, Order, and Order Item or Product. In the case of the Product and Order Item objects, these terms are interchangeable. The structure, purpose, and usage of the object is the same, regardless of which name is used. These objects are structured within the parent module Customer Accounts in a parent-child relationship:

```
MainObject > Customers > Orders > Products
```

The module main object contains a collection of Customer objects. The Customer object definition in turn contains a collection of Order objects. The Order object then contains a collection of Product objects.

The Customer object encapsulates customers of the Northwind Trading company. It contains property definitions for customer ID, company name, contact name, phone number, and address information.

The Order object encapsulates an order placed by the customer. It includes property definitions for the unique order ID, order date, delivery date, required date, and shipping information.

The Product/Order Detail object encapsulates an individual item ordered by a customer. It includes the unique product ID, product name, description, and quantity ordered within its property definitions.

Complex Tables and Data Tables

There is one complex table and one data table defined within the Mobile Northwind application. The data table contains a short list of shippers. The key field contains the Shipper ID value, and the value field contains the shipping firm name.

The Products complex table contains a list of the items which the Northwind company offers to its customers. There are fields defined for the Product ID, Product Name, Unit Price, and Quantity per Unit values. There is one index on the Product ID and one on the Product Name.

Transactions

Transactions exist for all three object types defined within the project. There is an add and edit transaction for the Customer object. The edit transaction allows the user to edit contact information for the selected customer, including contact name and phone number.

There is an add transaction defined for the Order object. Add is in support of order entry, allowing the user to record a new order for the selected customer. This transaction uses initial value rules on its transactions to set the initial value of all shipping address properties to the matching address properties of the selected customer object. The user can modify these values in the wizard for the transaction if necessary. The transaction also captures date and shipper information.

There is an add, edit, and delete transaction defined for the Product object. Products are added to the selected order object for the customer. Products are selected by the user from the Products complex table.

User Interface

The user interface for the Mobile Northwind application includes screen sets to display customers, orders, and products for orders. They are presented in a basic drill down navigation, with actions defined to go from one to the next.

Transactions are presented in standard wizards. The action to add new orders for a customer includes a looping SubAction step in addition to the transaction step for the add transaction for the order. The looping step executes the action for the Add Product transaction in a loop that continues until the user indicates they are finished. The screen flow from this then presents the screen set for the Add Order transaction once, followed by the screen set for the Add Product transaction being presented in a loop until the user has completed the entry of all desired products for the order.

The fields which capture data for the Add Product transaction are defined with update rules to calculate the total cost of the product order by multiplying the quantity being ordered by the unit price.

Target Paths and the Property Browser

Target paths are an important concept to understand when creating or working with an application project using the Agentry Editor. Target paths are selected using either a short list of likely options from a context menu for a given attribute field, or by using the Property Browser for more sophisticated paths.

To understand target paths it is first important to understand the concepts of target definition instances and referenced definition instances. A target is a definition instance that is affected

by some other definition at run time on the client. The specific impact on a target is typically setting a value in that definition. Targeted definitions are almost always a property.

A referenced definition instance is one whose value, or in certain circumstances the definition instance itself, is returned to the definition referencing it. What is done with the value or definition instance returned depends entirely on the referring definition.

A target path is then used to specify the definition instance to be targeted or referenced. Target paths always deal with a specific definition instance. A target path is a path evaluated in the context of the definition in which it is contained at run time on the Agentry Client.

A basic example of a target path is the value of a property to be displayed by a detail screen field. As a part of a detail screen field's definition the property to be displayed by the field is specified. This property is specified using a target path. This path may take the basic form:

```
:>"CompanyName" Property
```

In this basic example, the field is defined to display the property named `CompanyName` found in the definition being displayed by the detail screen containing the field. At run time this path is evaluated by the Agentry Client and the value of that property at that time is displayed in the field.

Target paths can retrieve data values from numerous definitions, including properties, the current value of a screen field, complex table fields, data table fields, and other definitions. The path itself can be as basic as the previous example, or far more complex and include rule evaluation and currently selected items in a list to determine the desired value. The complexity of the target path depends on numerous factors, including the context of the definition containing the target path and how that context relates to the logical location of the value to be retrieved, the parameters by which the value should be selected (e.g. a currently selected item in a list, a record from a complex table, etc.), and the nature of the definition in which the target value is located. Consider the following more sophisticated target path:

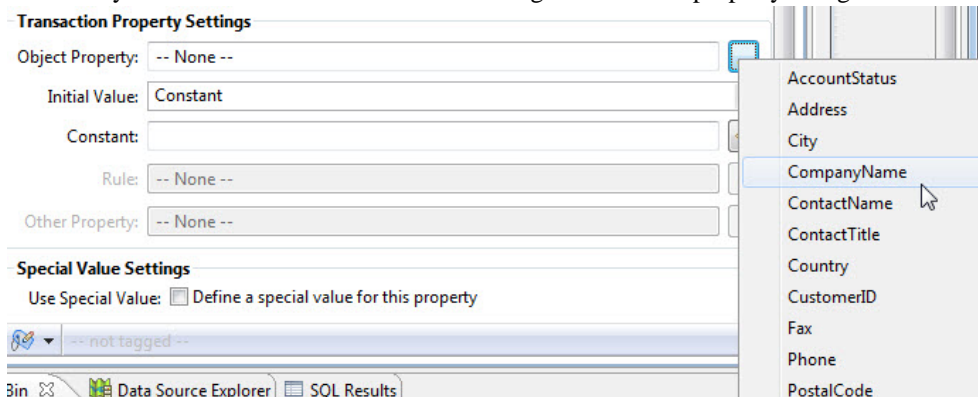
```
:>Main Screen Set>"ShowCustomers_List_PPC" List Screen>Current  
Object>"CustomerID" Property
```

This path returns the `CustomerID` property of the object that is currently selected in the `ShowCustomers_List_PPC` list screen, which is contained in the main screen of the current module. Such a path would be needed in a situation where the context of the definition needing this value is one where the module main screen set is not a descendent, for example a transaction definition. The above path may be one used to specify the initial value of a transaction property that must be initialized from the currently selected item in the list screen specified. Transactions are child definitions of the module, just as screen sets are. This means the transaction and screen set are siblings and the transaction is, therefore, not a descendent of the screen set.

Creating Target Paths: Attribute Field Context Menus

The first option when setting the target path for an attribute is to use the context menu displayed by the ellipses button for that attribute field. A basic example of this is a transaction property's target **Object Property**. This attribute is common among all transaction properties and specifies the object property whose value is to be set to the value of the transaction property when the transaction is applied.

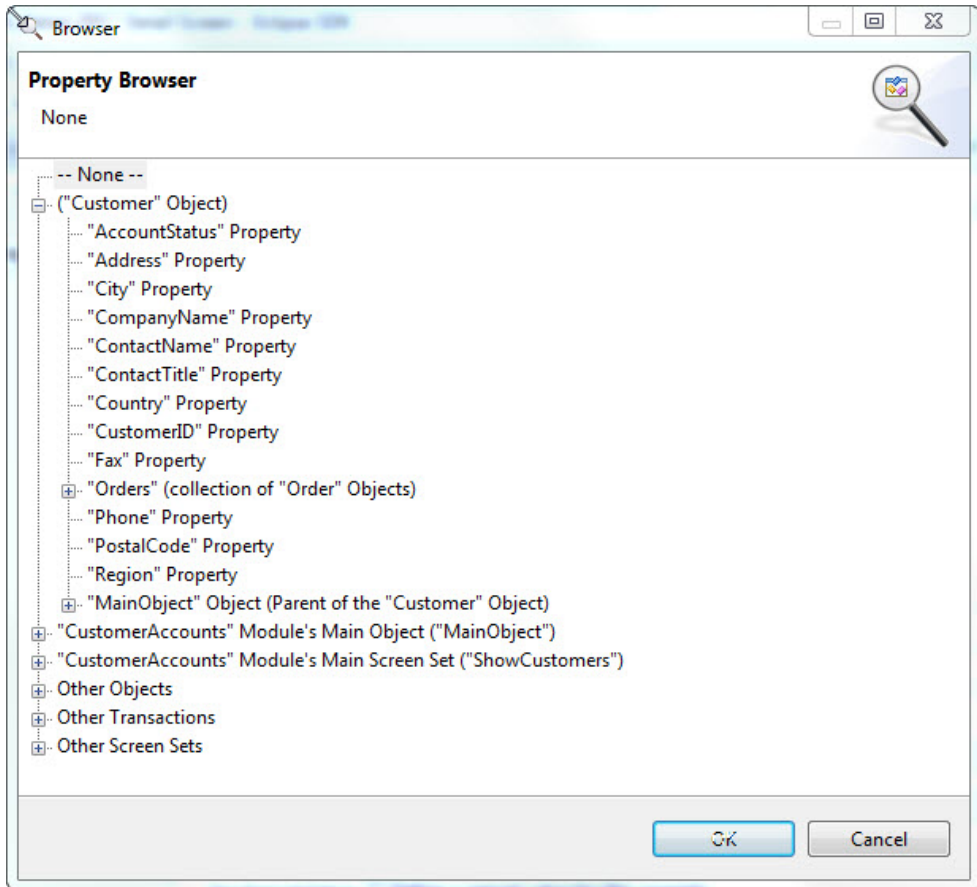
When setting this attribute, the ellipses button is clicked in the Editor for this attribute to display the context menu. Included in this menu is a list of all properties found in the object definition targeted by the transaction. These items are displayed by the Editor as they are the most likely candidates for selection when defining a transaction property's target:



Selecting an item from this menu creates a target path like the first example shown previously for the CompanyName property. Though not created using the Property Browser, this is still a target path. The Editor provides this list of likely options for the attribute as a shortcut to defining the application.

Creating Target Paths: The Property Browser

The selection of more involved paths is made supported by the Property Browser. The Property Browser can be displayed to set any attribute within the application project where a target path is allowed or required. In most cases, the menu displayed for such an attribute will include a list of likely definitions to be targeted by the attribute (based on the current context) and an additional menu item of **Browse...** or **Browse Objects...** This menu item displays the Property Browser, which provides a tree control of the various definitions in the structure of the application project that are valid selections for the attribute:



The available items in the property browser will vary from one definition to the next, depending on what the valid definition types are for the attribute and the definition containing that attribute. As with other aspects of the target paths behavior, this is driven by context.

An important concept to understand when using the Property Browser is that definitions are organized within it based on the data structure defined within the application project as it will exist on the Agency Client at run time. This is a different structure than the one presented in the main Project Explorer View of the Agency Editor.

Consider the example of two of the object definitions Customer and Order found in the Mobile Northwind application. When viewing this project in the main Project Explorer view, both of these definitions are listed under the module as its child definitions. Customer and Order are sibling definitions within the application project and are therefore presented at the same level in the hierarchy. By contrast, when viewing the Property Browser these definitions are presented differently. The Customer object may be presented as a root node in the tree (as in the previous example). Expanding it reveals all of that objects properties, including the Orders collection property which will contain the Order object instances for a given Customer at run

time. While the Project Explorer View also lists this collection property, the Property Browser goes further in that the Orders collection can be expanded to reveal several child nodes related to selecting a specific Order object instance based on some condition. Furthermore, under the selection criteria for an object instance in a collection, additional child nodes are displayed to allow for the selection of a specific property within that object.

The presentation of the definitions of the Property Browser is a reflection of the fact that the purpose of a target path is to select an instance of a definition at run time, based on some criteria. The criteria can be as simple as some property in the object instance targeted by a transaction; or it can be more sophisticated, such as a property in the parent object to the one targeted by the transaction, or even based on a rule evaluated at run time. Whereas the Project Explorer View deals with the definitions, the Property Browser details within specific instances of those definitions.

Beyond objects and their properties, the Property Browser can display several other definition types from which data can be retrieved on the Agentry Client at run time. These include items such as screen field values, the selected object in a list, a record in a data table or complex table, as well as specific fields within those records, and properties in transactions or fetches. The specific definition types available depends on the context in which the path will be evaluated at run time. For example, an object read step includes a **Read Into** attribute that specifies the target collection property into which data is read by that step during synchronization. When the Property Browser is displayed for this attribute there are far fewer options available than for some other attributes. In this situation, the options are limited to only object collection properties nested under the object for which the read step is being defined.

The Property Browser presents numerous options for selecting a specific definition instance, such as an object within a collection or a record within a complex table. Such options include the first or last item in a set, an object where the key property matches some specified value, or a record or object instance returned based on the evaluation of a rule. Understanding how to use the Property Browser to create such target paths is an important concept when developing many real-world applications or modifying existing applications to meet the specific needs of an implementation.

Basic Target Path Syntax

The syntax rules for a target path are important to understand so that when viewing a target path the developer can understand where the value being referenced is located within the application. There is no need, however, for the developer to understand the syntax at such a level as to be able to create such a path by hand. The Agentry Editor does not allow a target path to be entered manually for any attribute.

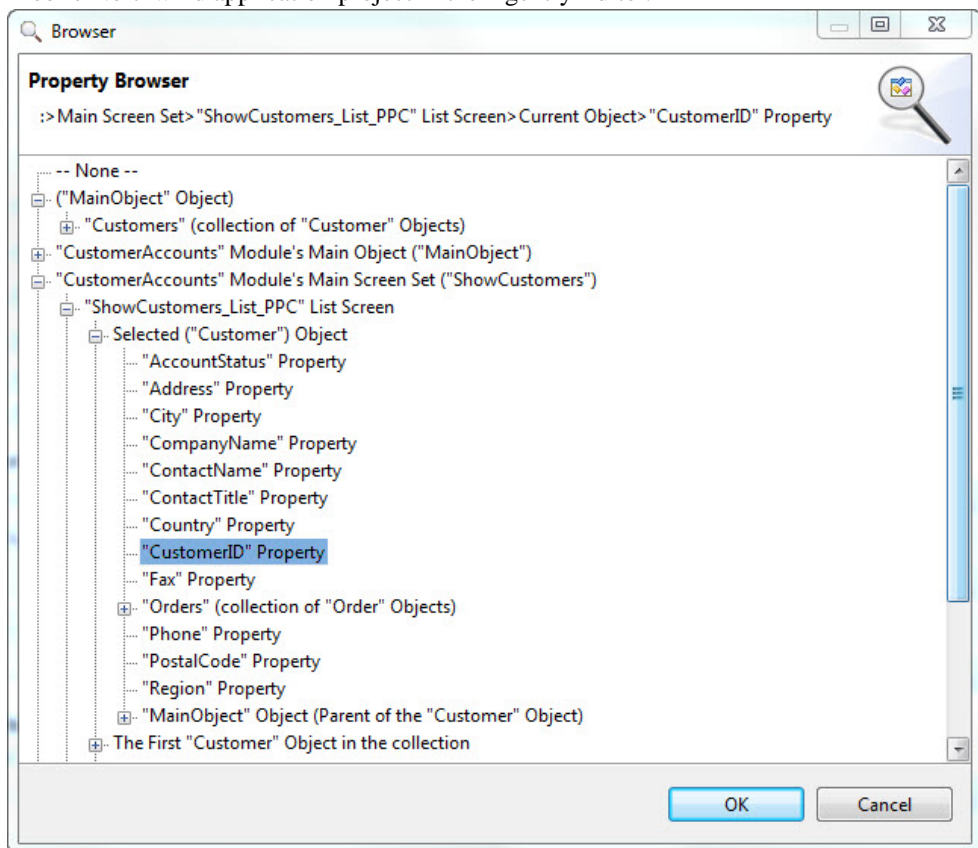
First, all target paths begin with the symbols `:>`. This simply denotes the beginning of the path for the target path parser built into the Agentry Client. This symbol is then followed by a definition name and type, or in some cases a generic definition type. A basic target path can end with just a single definition if that definition is found within the definition instance that is currently on context. Other paths contain multiple components, with each separated by the `>`

symbol. Each definition referenced in the path includes the name enclosed in quotes followed by the definition type.

Looking at the previous example, then:

```
:>Main Screen Set>"ShowCustomers_List_PPC" List Screen>Current
Object>"CustomerID" Property
```

The first component to this path is `Main Screen Set`. This is a generic component that refers to the main screen for the module. This value does not contain the name of this definition, which allows it to be used and evaluated properly even when the name of this screen set changes. The second component to the path is `"ShowCustomers_List_PPC" List Screen`. This component specifies the list screen of this name found within the main screen set of the module. `Current Object` is the next component and this refers to the currently selected object within the list screen. The final component is the `"CustomerID" Property`. This then specifies the `CustomerID` property within the currently selected object in the list. To create this path, the following was selected within the Property Browser of the Mobile Northwind application project in the Agentry Editor.



Note the selection, including the nodes above it in the tree control. The root node of the selected item displays “CustomerAccounts” Module’s Main Screen Set (“ShowCustomers”). However, this full description is not the item returned for the target path. Note the line of text displayed above the tree control in the screen header of the Property Browser. This is the actual target path that will be returned to the attribute for which the selection is being made.

In many cases a definition will be replaced in the target path with a more generic value, such as “Current Screen Set”, “Current Property”, etc. At run time, when the target path is evaluated by the Agentry Client, these more generic values allow for the reuse of a target path in multiple contexts. In some cases such paths are required. As shown in an example to be provided shortly, the target paths evaluated in the context of a detail screen that is in turn displayed through one of the tile category of detail screen fields (List Tile View, Tile Edit, and Tile Display) require such generic values in the target path when referencing another value on the same detail screen.

In other situations the target path selected in, for example, a rule definition may be replaced with a more generic value after that rule is saved. A common example of this behavior is when specifying a collection property for the @COUNT function. In many cases when the collection is selected the target path displayed in the rule editor will be something similar to “Customers” Collection Property. However, when the rule is saved and subsequently viewed or edited that same path will be the value “Current Property”. If the context of the rule evaluation includes the previously selected Customers collection, current property will resolve to that collection. Furthermore, the replacement of the name-specific value with the more generic one allows that rule term to be evaluated in the context of a different collection property, such as Orders.

Property Browser Details: Object-Related Options

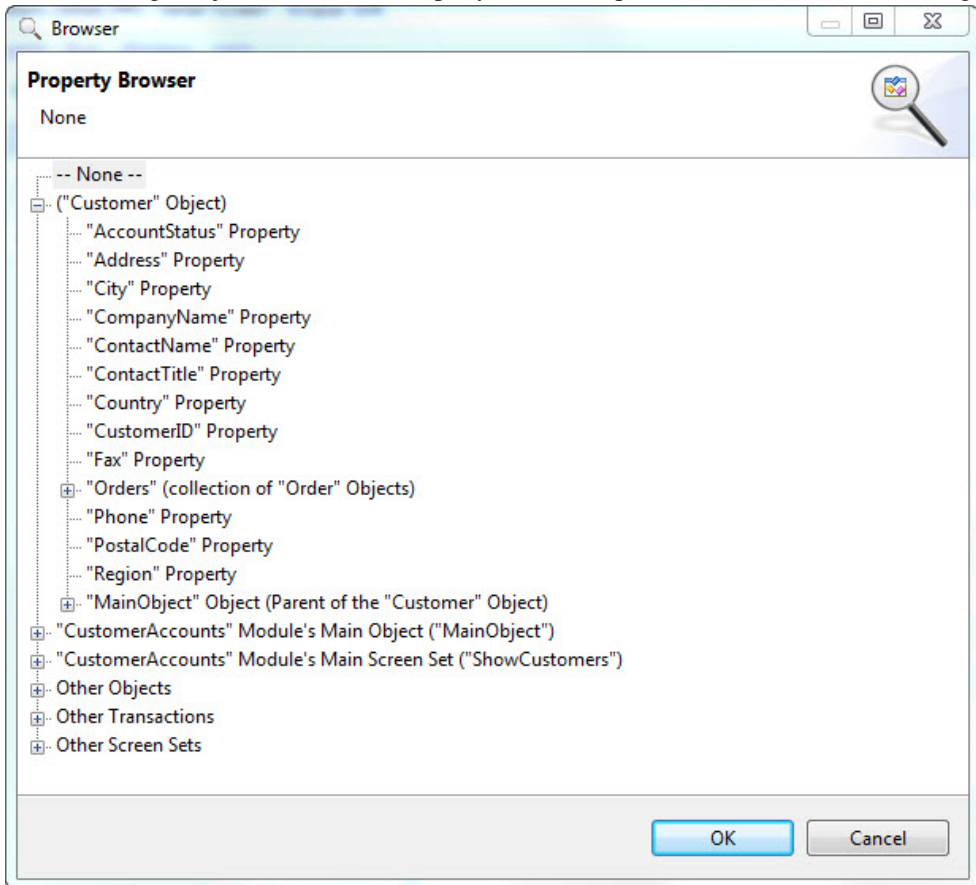
The property Browser presents numerous options related to the selection of an object instance and a specific property value within that instance. This includes both the selection of a value from the object in the current context and also the selection of an object within a collection. Many of the more sophisticated options are related to the selection of an object instance within a collection. These options include:

- Selection of a property within the object instance in the current context
- Selection of an object instance within a collection based on it’s position (first or last)
- Selection of an object instance within a collection based on a rule definition
- Selection of an object instance within a collection based on the key property value of that object

The specific layout of the module’s data structure within the Property Browser depends on the context in which the target path being selected will be evaluated at run time. In some cases the object is displayed as the first root node. In other contexts a collection is presented. In addition to these options, others may be available for screen sets, complex tables, data tables, and other options. These are addressed elsewhere, with the focus here on the target paths selected solely based on the object instances.

Target Path for an Object Property - No Collection

When setting the target path of an attribute, and when the context of the path to be evaluated includes a single object instance, the Property Browser is presented similar to the following:



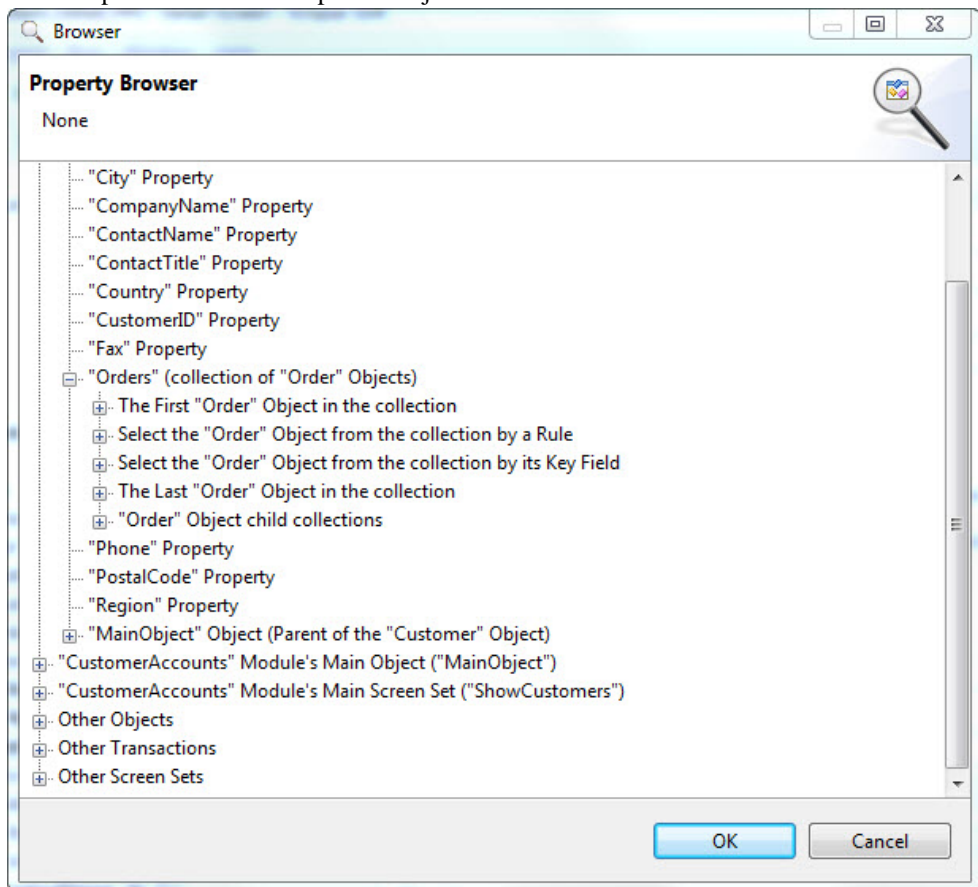
In this example the target object property of an edit transaction property is being set. The object in context is the object to be modified by the transaction. In the Property browser, the Customer object is displayed as the first root node and is expanded, as shown in the above example. If the proper option for selection is one of the properties under the Customer object, then it is likely that the Property Browser is not needed, as in almost all contexts the properties of the customer object would be displayed in the context menu of the attribute.

However, if the target paths should reference a collection property within the object, the Property Browser is needed. Collection properties are not displayed in the context menu of a transaction property's target object attribute. The collection is a valid selection if the edit transaction should replace the object collection in the object with values captured by the transaction.

It is also possible to reference the parent object of the one in the current context. Again in the previous example, the Customer object is expanded. Note the last child node under the Customer, which is MainObject. The description indicates this is the parent object of Customer, meaning the object instance is stored in a collection property of the MainObject. To select another property within the MainObject, that node can be expanded and the property selected from it. This same general procedure is available to all objects stored in collections to access the parent object containing that collection.

Target Path for an Object Instance - Selected From a Collection

In many contexts the object to be selected exists in a collection of objects of the same type. The specific object instance to be targeted must be specified. In almost all cases once the specific object instance is determined, a property of that instance is then the actual target being selected. When working with an object collection property, the Property Browser provides several options to select the specific object instance:



When the node for a collection property is expanded in the Property Browser, the following items are available:

- The first object in the collection
- Select the object from collection by rule
- Select the object from the collection by the key field
- The last object in the collection
- Object child collections

The First Object in the Collection: The first object in a collection is the first object instance added to that collection. This may be the first one retrieved from the back end system during synchronization; or if the collection is empty after synchronization, it is the first object instance added to the collection on the client.

In the case of data synchronization, the order in which the objects are stored in the collection is not guaranteed by the Agentry Client. The synchronization logic itself can include ordering of the data retrieved, which then provides an order to the objects stored in the collection. However, there are few use cases in real world applications where a target path is needed to select the first object instance in a collection.

The one real-world example of such a target path is when the collection is known to contain only a single object in all situations. One situation where this occurs is when an object exists in a collection of the MainObject to store user-related information. Typically the synchronization retrieves only a single object instance for this user information. When it is needed, a target path to that object must be created. Target paths for object collections always assume the possibility that multiple object instances can or will exist within a given collection. Therefore, selecting the first object in the collection in a situation where it is known that only one instance will ever exist is an acceptable manner in which to select this object.

Select the Object from the Collection By Rule: Selection this option requires the definition of a rule to evaluate the objects found in the collection. The default behavior at run time is for the rule to be evaluated once for each object instance in the collection until the rule returns true, at which point that object instance is returned, or until all objects have been evaluated. The rule is evaluated in the context of each object instance in turn, allowing access to the property values of each object.

As optional behaviors the object selected can be the first one for which the rule returns false; the last one for which the rule returns true; or the last one for which the rule returns false.

To select or define a new rule for this purpose, as well as to select options to change the default behavior, right click on the node in the Property Browser to display a context menu. Here a context menu is displayed with the following selections available:

- All rule definitions currently defined in the module.
- The option to define a new rule
- The option to specify if the object instance selected is the first or last to meet the rule's criteria (the first object found is the default)
- The option to specify whether to select the object instance based on a true or false return from the rule (true is the default)

If the option to select the last object meeting the criteria is defined, the rule will always be evaluated once for each object in the collection, as it must check all objects before determining

the last one to meet the criteria. This behavior should be considered if the selected collection has the potential to contain a large number of object instances, as it has the potential to increase processing time as the number of objects to evaluate increases.

Once the rule has been either selected or defined, and the optional other settings have been selected, the node can then be expanded to allow for the selection of the specific property within that object instance. The value of this property is then returned, or the property itself is targeted during run time.

Select the Object from the Collection by the Key Field: The object instance can be selected from a collection via the value of its key property, or alternately by another property within that object. In most cases the key property should be used as it is the only value guaranteed to be unique for all object instances within the collection.

To set this option, right click on this node in the Property Browser to display a context menu containing the properties in the object definition, as well as the value to which it will be compared. The key property is selected by default, but can be changed to any other non-collection property within the object.

The last item in the context menu **is equal to (Browse for Property)...** is then selected to display a second Property Browser. In this screen the value to be compared to the key property of each object in the collection is selected. This select is an example of a target path within a target path. This “nested path” is evaluated once for each object in the collection being searched, with the value it returns being compared to the key property of each object instance in the collection, until a match is found or until all objects have been searched.

One alternate selection in this context menu is the **Browse...** menu item directly below the list of all properties within the object. This can be selected for more complex search criteria, and will display a second Property Browser. Specifically, selecting an object from a collection that contains a nested collection with an object instance that meets some criteria. For example, selecting a Order object from the Orders collection where that Order object contains an OrderItem object with a Product ID value of 70.

To make such a selection, the property to be compared would be the ProductID property of the OrderItem object within the collection property of the Order object. The **is equal to (Browse for Property)...** menu item is then displayed to select the value to compare against the ProductID property of the child OrderItem objects. Note that such a selection will iterate over each object in the nested collection property of each object instance in the parent collection. So in this example, each OrderItem object is checked in each Order object until a match is found. Therefore, the potential number of iterations performed by the Agentry Client for such a target path is a multiple of the number of objects in the parent collection and the number of objects in the nested collection. This type of searching is rare, but there are a handful of use cases for which it can be applicable.

The Last Object in the Collection: The last object in a collection is the last object instance added to that collection. This may be the last one retrieved from the back end system during synchronization; or it will be the last object instance added to the collection on the client.

In the case of data synchronization, the order in which the objects are stored in the collection is not guaranteed by the Agentry Client. The synchronization logic itself can include ordering of the data retrieved, which then provides an order to the objects stored in the collection. However, there are few use cases in real world applications where a target path is needed to select the last object instance in a collection after data synchronization, and issues with such logic arise in that this selection becomes invalid if users add another object to that collection on the Client via a transaction.

A typical use case for building a target path that selects the last object in a collection is in the area of multiple transactions being instantiated and applied on the Agentry Client, where a transaction creates a new object, and a subsequent transaction must modify that new object in some manner. An example from the Mobile Northwind application is the order entry functionality. The Order object represents an order placed by a customer and includes header information such as the order date and the shipping address. It contains a collection of OrderItem objects that represent the products for that order.

From a usability standpoint, it makes sense for the creation of an Order object and the addition of one or more OrderItems to that object to appear as a single operation to the user. To accomplish this an action can be defined that executes two sub-actions. The first instantiates and applies the AddOrder transaction. The second instantiates and applies the AddOrderItem transaction in a loop. Any add transaction must know to which collection property the object it creates will be added. In the order entry scenario, the AddOrder transaction creates an object and adds it to the Orders collection property of the current Customer object. The subsequent AddOrderItem transaction must then target the collection of OrderItems contained in the newly created Order object. To specify this Order object it is safe to select the last Order object in the collection as the parent to the new OrderItem object created by the AddOrderItem transaction.

Object Child Collections: The selection of the Object Child Collections node under a given collection property displays a list of all collection properties nested under the current collection. Selecting one of these nested collections results in the return of all object instances in that collection in all instances of the parent collection.

An example to explain this is the Orders collection property contained in the Customer object of the Mobile Northwind application. If it is desired to define a list (list screen, list tile view field, list selection field, etc.) that displays all of the orders for all customers downloaded to the Agentry Client, the object child collections type of target path can be used. At run time this target path is evaluated by the Client and retrieves all Order objects from every Customer object. It can then display all of these Order objects in a single list regardless of the parent Customer object in which any of them are contained.

Property Browser Details: Screen-Related Options

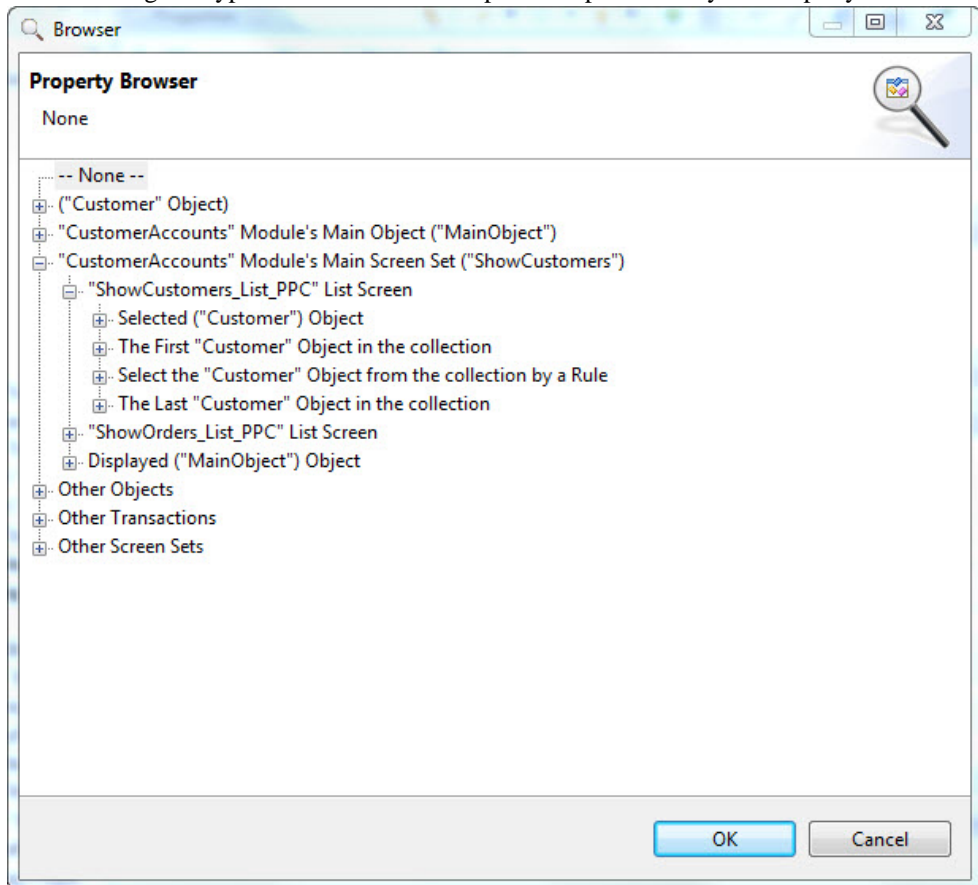
The Property Browser provides several options for creating a target path based on the selections made and the values entered or displayed on a screen. In general there are two categories into which these options can be organized: List-related options and Detail Screen Field options. The options available for different list controls are typically the same and, in

most cases, the item currently selected in that list is the target path created. For detail screen field options, this is typically related to the value displayed in the field, or in some cases the item represented by the selection made in a field.

Target Path for List Controls

The target path based off of a selection in a list control, which can include the list displayed on a list screen, or the list displayed by one of the detail screen field types that present a list of objects from a collection, will return a property from an object within that list. The collection being displayed can also be accessed directly, though typically this target path is selected via one of the object-related options also presented by the Property Browser.

The following is a typical list related set of options as presented by the Property Browser:



In this example the options for a list screen are shown. However, these same options are available for detail screen fields with an edit type of List Tile View and List Selection. All three of these list types display a defined object collection property at run time on the Client, and all three allow the user to make a selection from that list.

The options displayed here include:

- The selected object
- The first object in the collection
- Select the object from collection by rule
- The last object in the collection

Of these options, only the selected object relates specifically to the different list controls. The other three provide the same target paths and behaviors as the options described in the Object-Related Options for target paths and Property Browser.

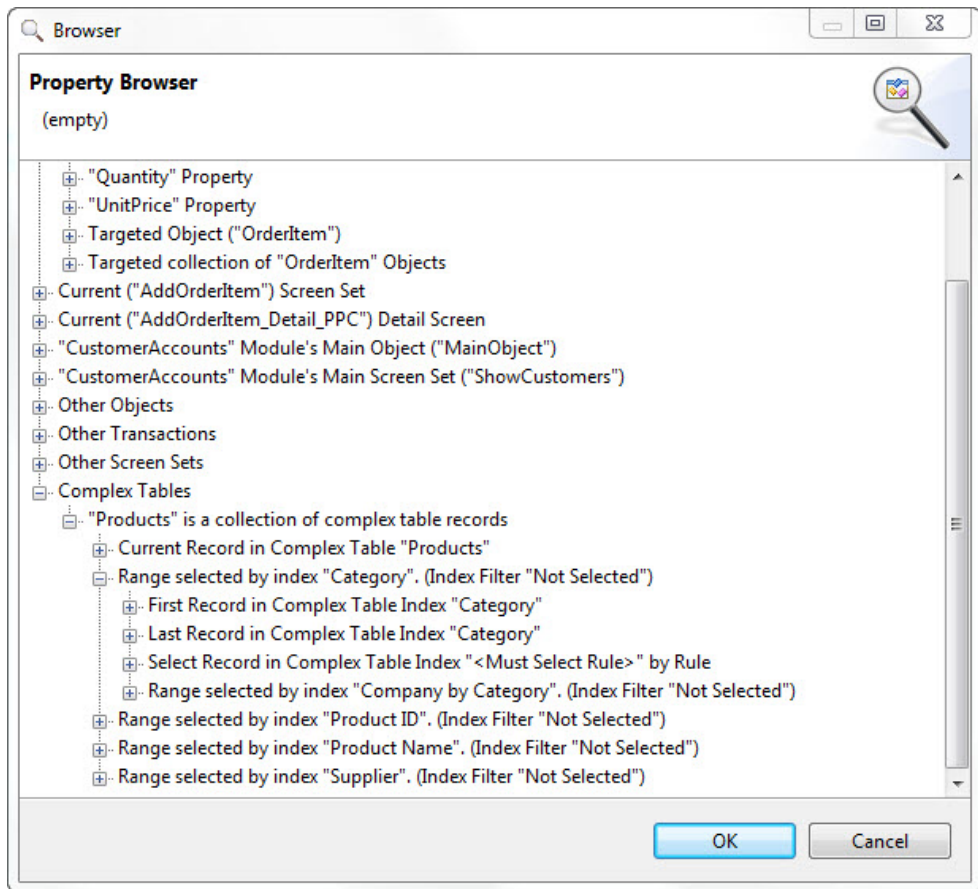
The Selected Object: Choosing the option of the selected object returns the object currently selected in the list. If the list is defined to allow for multiple selections, then this option provides the same options as when working with a collection property directly. The list of objects selected by the user are treated as a collection, with the instances in that collection being only those selected by the user. Like the options presented for a collection, there are numerous options that can be used to determine which specific object from those selected should be returned, including the first, last, by key field value, and by rule.

Property Browser Details: Complex Table-Related Options

The Property Browser provides several options for selecting a record from a complex table. Ultimately a specific field from the selected record is returned in almost all contexts. The specific field is a part of the target path created using the Property Browser. The selection criteria for the record can be based on one of the indexes defined within the complex table. When using an index, the first record, last record, or one selected by performing a search based on that index are all options. Searching on indexes in a complex table using a target path also includes support for parent-child indexes, if they are defined.

Target Paths for Complex Table Records

When a complex table record is targeted, the target path will ultimately include the specific field within that record whose value is to be returned. To select a record from a complex table, the selection criteria options are either the current record in the complex table, which is determined by context, or by using one of the indexes defined within the complex table. The following is an example of options provided in the Property Browser for a complex table definition:



The options displayed here include the following:

- The current complex table record
- Range of records by index
- Select the first record in an index
- Select the last record in an index
- Select a record based on a rule
- Select the record using a child index within a parent index

The Current Complex Table Record: This option creates a target path that returns the current record in the complex table. The concept of the current record can be somewhat flexible and is based on the context of the target path evaluation. In general, the current record in a table is one that is currently selected in a list that displays complex table records, or some similar behavior.

Range of Records by Index: Within a complex table there will always be one or more index definitions. The index provides order to the records of the complex table based on the values of

a field within the records. indexes are defined to support both sorting and searching behaviors. When selecting records by using an index, right clicking on the node **Range selected by index** “*IndexName*” displays a context menu where the search value is specified. The source for this value can be either the value returned by a rule definition, or by creating a nested target path.

In the case of a rule, the rule itself is evaluated in the context of the parent definition for which the target path is being generated. The data type of the rule context matches the data type of the field for which the index in the complex table is defined. So if the index is defined for a field with one of the four string data types, the rule is evaluated in a string context and this is the data type of its return value.

In the case of a nested target path, a second Property Browser is opened and the source for the value to search on can be selected. Note that in this case, the source selected must be of the same general data type as the field to which it is compared. If the complex table field is a string, the search value must also be a string. The Agentry Client does not perform any data conversion for this comparison as it does in other situations.

Whether using a rule or some other data source for the search value, at run time the Agentry Client searches the complex table using the search value and the selected index. The resulting record may be either a single record or a range of records with the matching value. When selecting a range, the return should be for some list that supports displaying complex table records. This behavior allows for creating a temporary list of records that is a subset of all records in the complex table. This subset is dynamic in the sense that the source for the search value may return a different value under different conditions, especially when a rule definition is used. When a range of records is to be returned, the specific field from those records is not selected.

Select the First Record in an Index: When selecting the node **First Record in Complex Table Index** “*IndexName*”, the Agentry Client uses the order provided by the selected index to select the record, with the first record in the index being selected. A single field is selected for this option as the value returned for that record. As an optional behavior, right clicking on this node presents a context menu that allows for the specification of a record position within the index, counting up from the first record, which is at position 1.

Select the Last Record in an Index: When selecting the node **Last Record in Complex Table Index** “*IndexName*”, the Agentry Client uses the order provided by the selected index to select the record, with the last record in the index being selected. A single field is selected for this option as the value returned for that record. As an optional behavior, right clicking on this node presents a context menu that allows for the specification of a record position within the index, counting back from the last record, which is at position 1.

Select a Record Based on a Rule: When selecting the node **Select Record in Complex Table Index by Rule**, a rule definition is selected or defined by right clicking this node and selecting the appropriate option from the context menu. The rule used here is evaluated iteratively, once for each record in the complex table, until a match is found. The data type of the rule’s context is Boolean.

The record selected is the first for which the rule returns true, by default. The order in which the records are processed is dictated by the order established by the index definition. As optional behaviors, the target path can be defined to select the first record for which the rule returns false. Also, the last record within the index for which the rule returns true (or false if selected) can be returned instead. Both of the optional behaviors are set by selecting the corresponding options in the context menu displayed for this node.

Select the Record Using a Child Index Within a Parent Index: When a child index is defined within the complex table, expanding the parent index reveals options for that child. These options are the same as the others for an index. However, the overall behavior includes child index searching. When one of these options is selected, selection criteria must also be specified for the parent index. Specifically, the selection of a range for the parent index must be specified, along with one of the selection options for the child index. This is required due to the nature of parent-child indexes in complex tables.

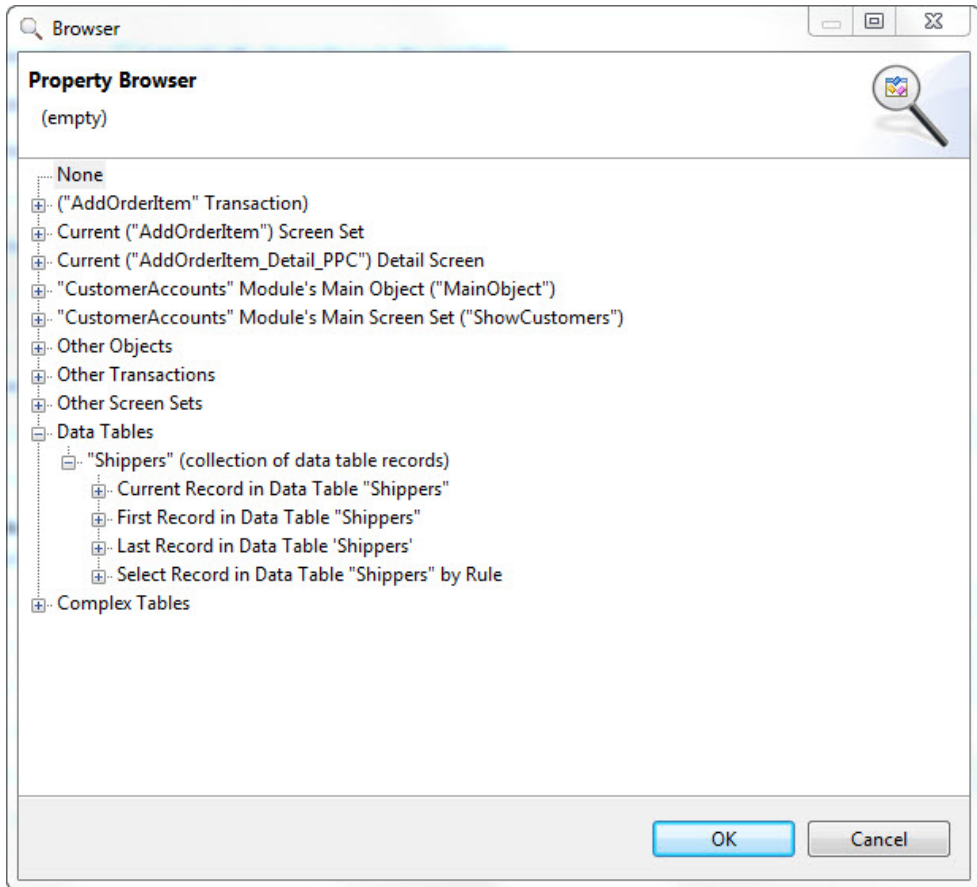
Property Browser Details: Data Table-Related Options

The Property Browser presents several options to select a record from a data table. The resulting target path created typically returns one of the two field values from the record, either the code or value. In some cases it may return the record, depending on the context in which the path is evaluated.

Target Paths for Data Table Records

When using the Property Browser for a target path to select a data table record, the options presented include selection via rule or position, as well as the current record. Typically the selection also includes specifying the field from the record to return. By default the value field is returned. Right clicking on any of the selection nodes provides a context menu where this default can be changed to the code field.

The following is an example of the options presented for a data table's target path in the Property Browser:



As shown in this example, the following options are available:

- The current data table record
- The first data table record
- The last data table record
- Select a record based on a rule

The Current Data Table Record: When the node **Current Record in Data Table** is selected, the record returned by the target path is the “current” record. The concept of a current record typically relates to the record in the data table currently selected in a list, or possibly in some other context where a current record exists, for example in a nested target path where the parent path includes a record’s selection based on some other criteria.

The First Data Table Record: When the node **First Record in Data Table** is selected, the record returned is the first one stored in the table on the Agency Client. It is important to note that the order of the records in a data table is not guaranteed, nor are the records sorted in any manner by the Agency Client. However, the synchronization logic of the data table can order the records during retrieval to provide an order for their storage on the Client.

As an option to this selection, right clicking on the node presents a context menu with the option to specify a record at a position relative to the first. This position is set numerically, with the first record in the table at position 1.

The Last Data Table Record: When the node **Last Record in Data Table** is selected, the record returned is the last one stored in the table on the Agentry Client. It is important to note that the order of the records in a data table is not guaranteed, nor are the records sorted in any manner by the Agentry Client. However, the synchronization logic of the data table can order the records during retrieval to provide an order for their storage on the Client.

As an option to this selection, right clicking on the node presents a context menu with the option to specify a record at a position relative to the last. This position is set numerically, with the last record in the table at position 1. The second to last record is then at position 2, and so forth.

Select a Record Based on a Rule: When the node **Select Record in Data Table by Rule** is selected, a rule definition must be selected or defined. This rule is evaluated once for each record in the table, with the record in context. The rule's return data type is Boolean. The default behavior is to select the first record in the table for which the rule returns true, at which point the rule is no longer evaluated.

As optional behaviors, the return value of false can be used to specify the record to select. Also, the last record for which the rule returns true (or false, depending on the option selected) can be the one returned. If the last record is selected, the rule will always be evaluated for each record in the data table.

Target Path: Selecting an Object By Property Value

Prerequisites

Prior to performing this procedure the following items should be addressed:

- The object collection to be searched must be defined.
- The definition containing the value to be compared to the property value of the object must exist. This could be a property, a screen field, a global, or any other definition from which a value can be retrieved.
- It is strongly recommended that the value to be searched on is the key property of the object type contained in the collection.

The following items and definitions are a part of the example application project used in this procedure:

- The module's object data structure is Customer -> Orders Collection -> OrderItems Collection.
- OrderItems is a collection of OrderItem objects that represent items within a given order. OrderItems can be added to the Order object using the AddOrderItem transaction and related wizard screen set.

- When adding an OrderItem, the complex table Products is used to select the item to be added. The key property of the OrderItem object and the key field of the Products complex table are both the ProductID value.
- The AddOrderItem screen set displays a complex table search field for the Products complex table. When a record is selected in this field it displays the ProductID value of the selected record.
- A field of edit type Label is present on the detail screen in the AddOrderItem wizard. The field is defined to display the label text DUPLICATE. The RedText style has been applied to this field to display the label text in red and in a larger font size than other field on the screen.
- Other fields not directly related to this procedure displayed on this screen are the unit price, quantity being ordered, and discount percentage.

Task

This procedure provides instructions and an example on how to create a target path that returns an object instance from a collection property where a property of that object instance matches some value. The value to be compared against the collection must be accessible in the current context and therefore must be defined prior to performing this procedure. For this example, a detail screen field displaying a string will be used.

At run time the behavior of the Agentry Client will be to evaluate the target path and to compare the selected property in each instance the object in the collection property to the selected search value until a match is found. The first matching object instance is then returned. Note that this iterative processing can include up to as many iterations as there are object instances stored in the collection property. Typically this is not an issue for performance unless the collection contains an exceptionally large number of object instances (hundreds or thousands). However, if this target path is itself evaluated as a part of some outer loop of iterative processing, a performance issue could be encountered. For example, if the collection contains 100 object instances, and the outer loop iterates as little as 10 times, the total number of evaluations could be as many as 1,000.

Such a situation could also be encountered if the target path is evaluated in an update rule for a detail screen field. Update rules are evaluated numerous times during the initial presentation of the parent screen, and additionally whenever the user interacts with the detail screen. On a wizard screen, this would result in the target path being evaluated each time the user enters a character in a string field, or makes a selection in some list or drop down field on the same screen.

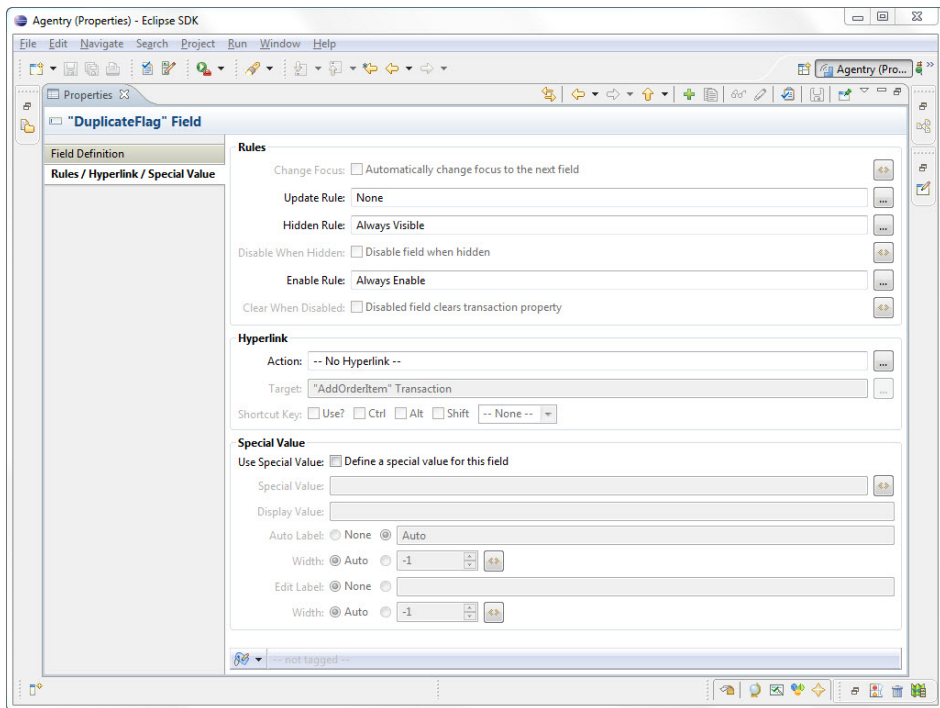
In this example the Property Browser is used to create a path that searches the OrderItems collection of an Order object for an OrderItem with a ProductID (key property) that matches the one currently selected in the AddOrderItem wizard. The goal is to display a clear indicator on the AddOrderItem wizard screen when a product is being selected that has already been added to the OrderItems collection. While the Agentry Client would prevent a second object with the same key property from being added to the collection, and an error message is displayed, this does not occur until the Agentry Client attempts to apply the transaction. A

cleaner user interface is possible that displays a message as soon as the duplicate product is selected.

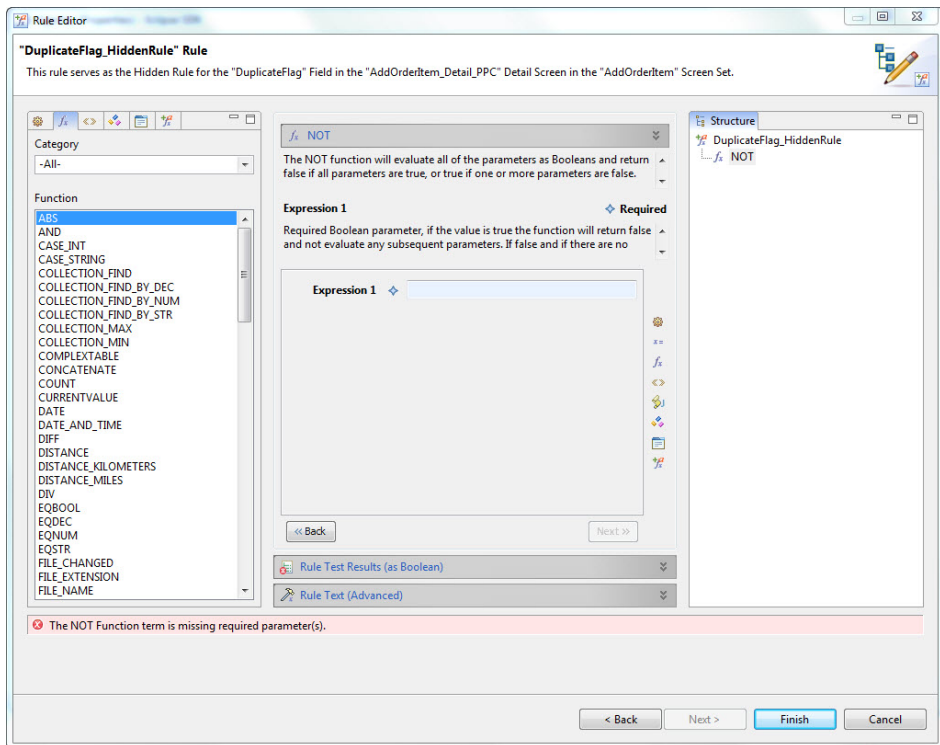
The label field contain the text DUPLICATE is to be modified with it's Hidden Rule attribute set to a rule that checks the OrderItems collection for an object with a ProductID value equal to the one currently selected in the Product ID field of the AddOrderItem wizard. The rule will contain a target path that makes this check and returns the property ProductID from the object found. If no object is found, then a null value is returned by the target path. The return from this path is to be passed as a parameter to the rule function NOT, which treats any parameters as Boolean values and inverts them. The Hidden Rule evaluates the rule it contains in a Boolean context. When such a rule returns true, the field containing the attribute is hidden from the user. When it is false, the field is displayed.

The overall logic, then, will be that the user selects a product from the complex table. The Hidden Rule for the label field DuplicateFlag will be evaluated. This rule contains the target path that compares the selected ProductID value to the ProductID property of each object in the OrderItems collection. If a match is found, the ProductID value of that object is returned to the NOT function within the rule. This is treated by the NOT function as true, since the path is evaluated in a Boolean context and any non-null value is true. NOT inverts this value, thus returning false to the Hidden Rule attribute. This results in the DuplicateFlag field being displayed. On the wizard screen the text DUPLCIATE is displayed in large red text. If the user selects a product that is not currently found in the OrderItems collection, the target path in the Hidden Rule will not find a matching object. The value returned by the path will be NULL. The NOT function treats NULL as false. It will then invert this value, returning true to the Hidden Rule attribute. This will hide the DuplicateFlag field on the screen.

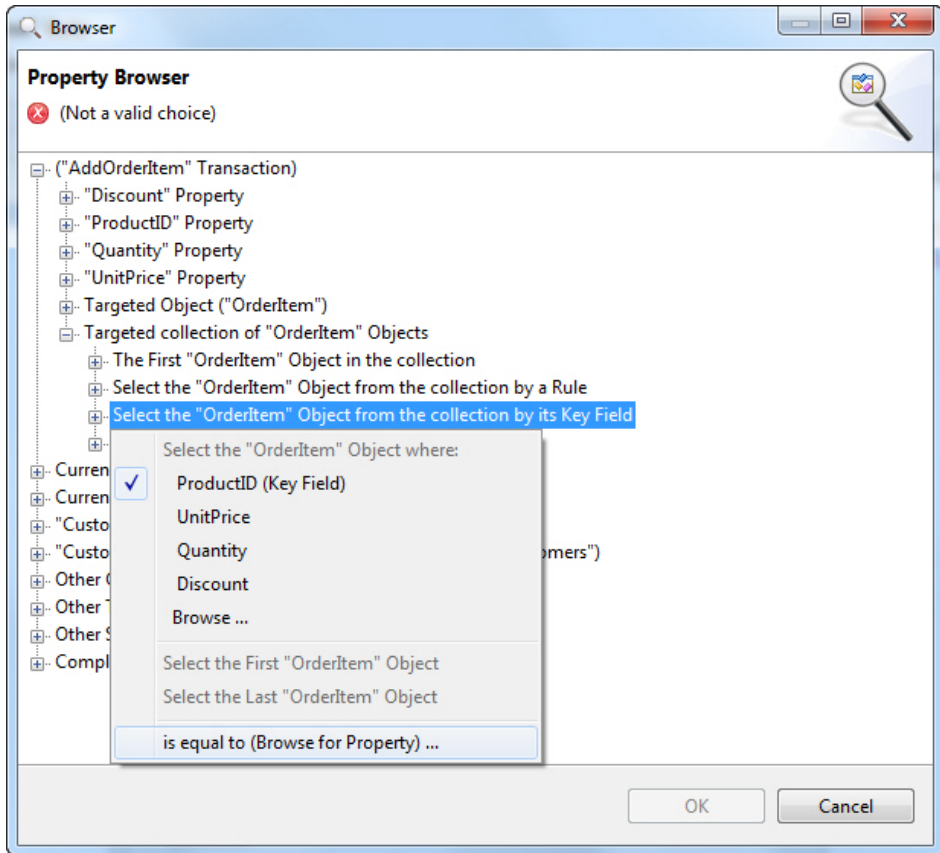
1. In our Mobile Northwind application we select the DuplicateFlag field definition in the AddOrderItem_Detail_PPC detail screen. In the Properties View we view the Rules / Hyperlink / Special Value tab:



2. Clicking the ellipses field to the right of the Hidden Rule attribute displays a context menu. Selecting the menu item **Add Rule** displays the Rule Editor. Here the name can be left set to the default of DuplicateFlag_HiddenRule. Advancing the wizard displays the main Rule Editor screen. The first term to be added to this term is the NOT function:

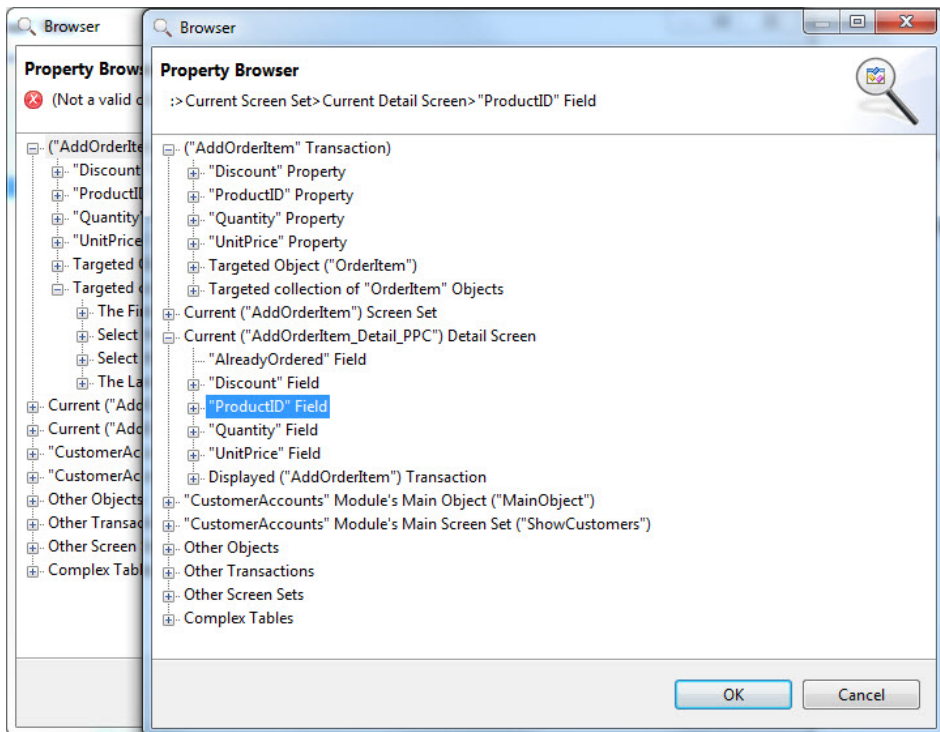


3. The first and only expression parameter for this function is to be the target path to search the collection. With the Expression 1 field selected in the Rule Editor, we click the Properties list and select **Browse Properties...** This displays the Property Browser. In this browser we will build the target path to search the OrderItems collection targeted by the transaction. We select the path **AddOrderItem Transaction | Targeted collection of OrderItems Objects | Select the OrderItem Object from the collection by its Key Field**. Right clicking the last item in this path displays a context menu:



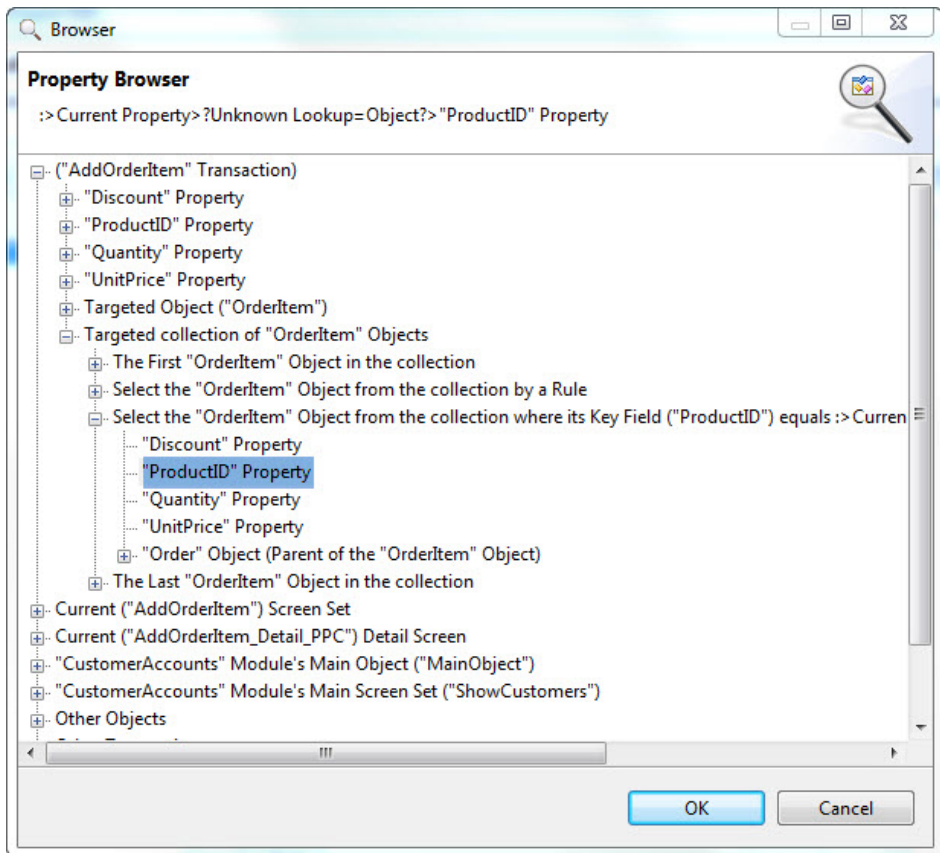
This selection creates the first part of the target path, which indicates the OrderItems collection targeted by the transaction. We are then specifying that we want a single object instance from this collection. There are multiple options for choosing the object instance. For this use case the proper selection is to look for the object by its key property.

4. In the context menu, first note that the key property ProductID is selected by default. The other properties are also listed, but typically the key property is used as it is the only value guaranteed to be unique within the collection. We now select the last item in the menu **is equal to (Browse for Property)...** This selection is where the value to be compared to the key properties of the object instances is selected. This selection displays a second Property Browser. The value we wish to search on is the current value in the Product ID field of the detail screen. So, the path we select is **Current ("AddOrderItem_Detail_PPC") Detail Screen | ProductID Field:**

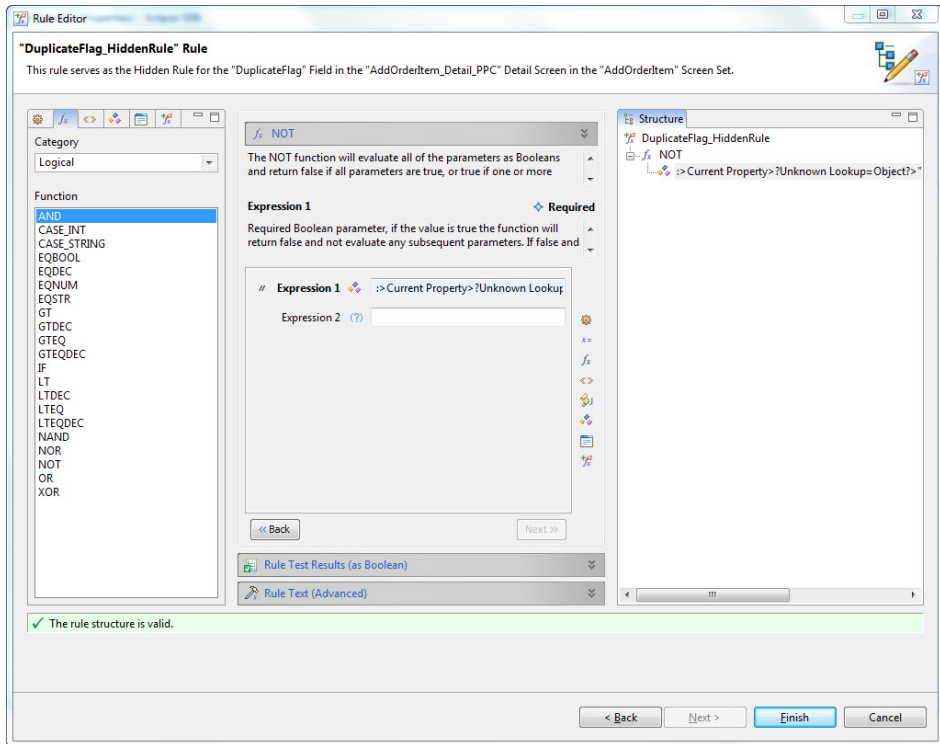


This selection creates a target path that returns the value currently displayed in the Product ID field of the detail screen. This path can be thought of as one that is contained within the path selected in the first Property Browser. This “inner path” is evaluated once for each object in the OrderItems collection until the value it returns matches the ProductID property of an object in that collection, or until all objects have been searched.

5. Click the OK button in this second Property Browser screen. We must now make one final selection in the first Property Browser, which is the property value to be returned from the OrderItem object found in the collection. This could not be selected previously as this selection cannot be made until the search criteria is specified. Now that it is selected, the node indicating search by key property can be expanded, revealing the properties within the OrderItem object. Here, the property to be returned is selected. The safe selection for our use case is the ProductID, as it will always contain a value. Other properties can be selected in other use cases, depending on what data is needed from the object found:



6. Close this Property Browser by clicking OK. We are returned to the Rule Editor, where the rule now appears as follows:

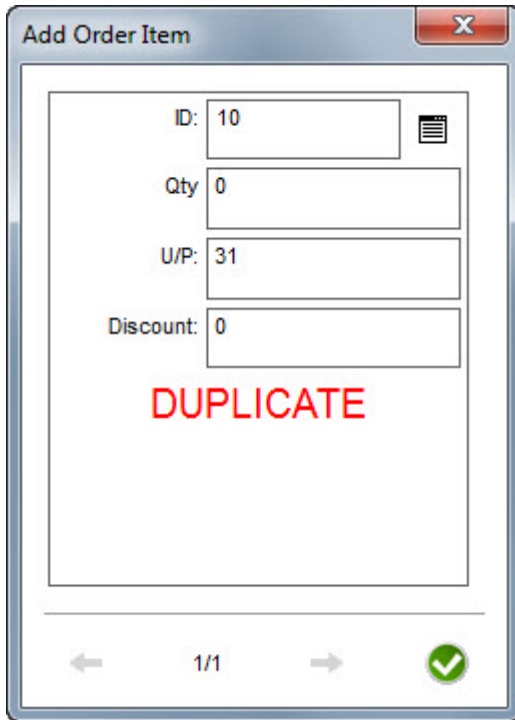


Clicking **[Finish]** returns us to the Properties View for the DuplicateFlag label field. The changes are saved in this view and the modification is complete in the Application Project.

At this point, the changes made are complete. We will now publish and test this modification. the following examples are from the Agentry Test Environment.

When the user adds an OrderItem for an Order, the AddOrderItem wizard is initially displayed:

When the user makes a selection in the ID field, the Hidden Rule for the currently hidden DuplicateFlag field, the target path built in the previous procedure is evaluated within the rule definition. The value displayed in the ID field is compared to the ProductID property in each OrderItem object in the collection targeted by the AddOrderItem transaction. If a match is found, the path returns the ProductID of that object instance, which is treated as true. The NOT function inverts the value within the rule, returning false. A false Hidden Rule value indicates the field should not be hidden, and the field's label text is displayed:



The screenshot shows a mobile application dialog box titled "Add Order Item" with a close button (X) in the top right corner. Inside the dialog, there are four input fields: "ID:" with the value "10", "Qty" with the value "0", "U/P:" with the value "31", and "Discount:" with the value "0". To the right of the "ID:" field is a small icon of a document with lines. Below these fields, the word "DUPLICATE" is displayed in large, bold, red capital letters. At the bottom of the dialog, there is a navigation bar with a left arrow, the text "1/1", a right arrow, and a green checkmark icon.

The user has now been informed that the product currently selected is already a part of the Order. If the user then makes another selection, the Hidden Rule, including the target path it contains, is evaluated again. If no matching OrderItem is found, the DuplicateFlag field is hidden and the user knows the product can be ordered:

The screenshot shows a mobile application dialog titled "Add Order Item". It contains four text input fields stacked vertically. The first field is labeled "ID:" and contains the value "21", with a small list icon to its right. The second field is labeled "Qty" and contains "5". The third field is labeled "U/P:" and contains "10". The fourth field is labeled "Discount:" and contains "10". At the bottom of the dialog is a navigation bar with a left arrow, the text "1/1", a right arrow, and a green circular button with a white checkmark.

Target Path: Selecting All Nested Collections

Prerequisites

Prior to performing this procedure the following general items must be addressed:

- The object data structure must be defined, including the parent object and the collection property within that object.
- The functionality described here is only available on Agency v. 6.0 and later.

The following items and definitions are a part of the example application project used in this procedure:

- The module object structure is defined as Customers -> Orders -> OrderItems. Each of these is a collection property of objects and are nested as listed. Customers is a top-level collection and thus is stored in the module main object.
- The main screen set named ShowCustomers is already defined. It includes a list screen displaying the top level collection Customers.

Task

In many applications the desired user interface layout includes displaying object top-level collections and also all instances of objects stored in nested collections regardless of the parent

object instance. As an example, it is desirable to display a list of all work orders in a work management application on the main screen set of the module. Additionally, a common change is to also display all of the equipment objects, which are stored in a nested collection of the work order object, in a single list regardless of the parent work order object.

Prior to version 6.0 of the Agentry Mobile Platform, in order to support this user interface layout, the equipment objects would need to be retrieved separately from the work orders and stored in a top-level collection of equipment objects. A list screen or a detail screen containing one of the list type fields was then defined to display this collection. The challenge with this approach was that in order to also display a list of equipment objects to the user for a single work order, one of two approaches was needed.

One option was to have a top-level collection of equipment objects, and a second, nested collection within the work order object. The issue with this approach was, of course, that the same data was retrieved twice, that being the objects for both collections. Both would then be stored on the Agentry Client. This was a waste of resources on the device, in essence doubling the amount of storage required to store the equipment information. The logic needed to retrieve all equipment for the work orders assigned to a given user was also sometimes challenging to write in an efficient manner.

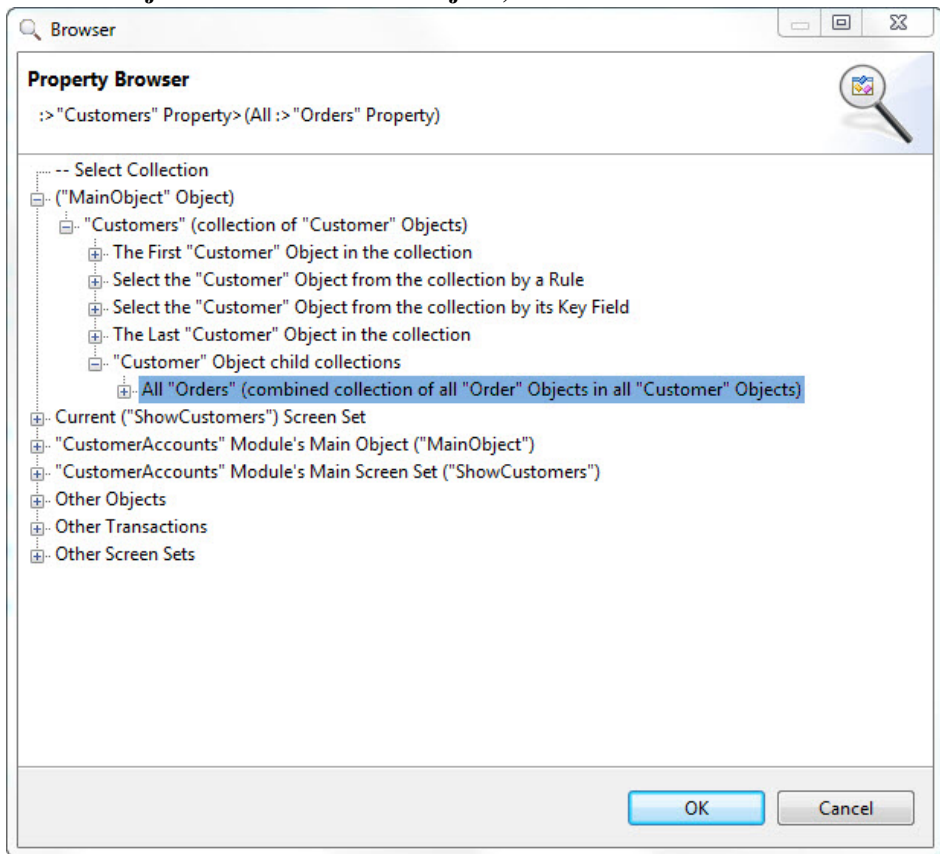
The other option was to store all equipment objects in only the top level collection. To then display the equipment objects for a given work order, it was necessary to define an include rule for the list in which these equipment records were displayed that would only return the equipment objects from the collection for the currently selected work order. This could cause performance issues if there was a large number of objects stored in the top-level collection of equipment. Also, if this behavior was desired for what would otherwise be nested collections, numerous rule definitions were needed, making the maintenance of the application project more cumbersome.

Fortunately this issue has been addressed in the 6.0 release of Agentry. A new target path option is now available in the Property Browser that allows for the selection of all objects in all instances of a given collection property definition. So, building on the previous example of work orders and equipment, the work order object is defined to contain a collection property for equipment. There is no top-level equipment collection, only the nested collection in the work order object. A list can now be defined to display all of the equipment objects found in all of the collection properties of all work order objects by selecting the proper target path to return this data.

In the following procedure this behavior is defined for the Mobile Northwind application. A list screen is added to the module main screen set ShowCustomers, which initially contains a list screen for the Customers collection. The second list screen to be added will display all order objects for all customers. The data, however, will only be stored in the nested collections, meaning a given instance of the Customer object will contain an Orders collection property of just that customer's previous orders. The Agentry Client will merge all instances of the Orders collection from all Customer object instances and display them as a single list in the list screen.

Note that while a list screen is used in this example, the option to display all nested collection instances for a given parent object type is available in numerous cases and contexts. This includes other list controls, such as a list view or list tile view detail screen field, and also in non-user interface contexts for operations that affect a collection.

1. We begin by creating the list screen (or one of the list type fields for detail screens, where applicable) and setting the attributes as normal.
2. When setting the Collection attribute, click the ellipses button to the right and select **Browse Properties...** from the context menu. This displays the Property Browser. Here, select the path **MainObject Object | Customers (collection of "Customer" Objects) | "Customer" Object child collections | All "Orders" (combined collection of all "Order" Objects in all "Customer" Objects)**:



3. Close the Property Browser and return to the wizard (or Properties View if editing an existing definition). Complete the definition as normal.

When this procedure is complete a list screen (or other list control) is created that will display all object instances from all of the selection collection property instances in the parent object type. In the above example this results in a list screen listing all orders placed by all customers:



Rules: An Introduction

The Rule definition type is a module-level definition within the application project. A rule defines evaluation logic processed on the Agentry Client. A rule is evaluated by some other definition that calls or references it. The rule will return a single value to the caller. This value is then used by the referencing definition for its own purposes. The caller of the rule sets the rule's context. Rules are made up of data terms and function terms.

The rule definition is the most complex of the definitions within the application project. Its overall purpose is to perform involved logical evaluation and to then return a single value based on or resulting from that evaluation. Within the application project there are dozens of attributes that can reference a rule definition. The uses for rules are varied and range from dynamically setting display values, to enforcing business logic, to enabling or disabling functionality based on some condition.

There are several concepts related to rule definitions that are necessary to understand before defining rules. These include:

- Components of a Rule's Structure
- Context of Rule Evaluation
- Rule Function Terms
- Rule Evaluation at Run-Time

Components of a Rule's Structure

A rule is a definition within the application project and, as such, does contain a handful of attributes. Specifically, each rule has a name and a group. The name uniquely identifies the rule within the module. The group is an application project-specific value used to organize rules.

The real definition of a rule, however, is contained in its Structure. The structure of a rule is the encapsulation of the logic to be evaluated at run-time. This logic then determines what the rule ultimately returns to the definition that called the rule. The components of a rule's structure are referred to as Rule Terms. There are in general two types of terms that make up a rule. These are Function Terms and Data Terms. How these terms are then organized within the rule definition provides the overall structure of the rule; i.e., its evaluation logic.

Function terms provide specific processing or logic performed during the rule evaluation. Most functions take one or more arguments, or "parameters" that provide values to be processed by the function. The function itself will then return a value to its caller based on the value of its parameters. A function's parameters can include both the return value from other functions and rule data terms.

Data terms are any term that is not a function and that provide data values processed by functions within the rule. There are several different sources for a data term, including properties, globals, actions, screen sets, and constant values set within the rule structure.

Context of Rule Evaluation

The processing of a rule definition on the Agentry Client is referred to as "rule evaluation." This evaluation is always performed in some context based on the definition attribute for which the rule is being evaluated, and the data type expected to be returned by the rule. The context of a rule's evaluation will affect what values are in scope for that evaluation and the overall behavior of the rule and its functions.

This behavior is driven by the data type specified by the context of the rule evaluation. A given rule is expected to return a specific data type based on the context in which it is evaluated. The rule will always return a value in that data type. If a caller of a rule expects a string, the rule will return a value with a data type of string. Within the rule, function terms have a similar behavior. Functions will always return a value in the data type asked for by whoever called the function. This caller will either be another function, or the caller of the rule.

The impact of a rule's context on the data type of its return value, as well as the impact of a function's context on its return value's data type, is one that is important in understanding the overall evaluation processing of a rule. Most importantly, it is necessary to understand that not all functions support all return types. If a function is asked for a value in a data type it does not support, it will return a null value in the data type for which it has been asked. It is therefore important to have a clear understanding of both the context in which a function is called, and whether or not that function supports the return type dictated by that context.

Context will also impact what definitions can call a given rule. Rules are normally defined in the context in which they will be evaluated. However, as with most definitions, rules can be referenced by more than one caller. For rules, the data terms referenced by the rule, specifically properties must be available in all contexts in which the rule will be evaluated. A property will be referenced within the rule's structure via a target path. This path must be one that is valid in every context in which the rule is evaluated. If it is not, the rule will return a null value for that property. This is likely to produce unexpected return values from the rule.

Rule Function Terms

Rule function terms, or simply rule functions, are the terms that provide the overall processing of a rule. Each rule has an entry point, which is the term that will return the value to the rule that is then returned to the caller of the rule. The simplest rule definition is one for which this entry point is set to a data term. However, such rules are uncommon as there are few situations in which it is necessary to return a value such as this using a rule.

In the majority of rule definitions, the rule's entry point is a function call. Most functions take parameters. The parameters to a function provide the values the function will process or manipulate in order to produce a value to be returned by that function. This is similar to functions or methods in other languages a developer is familiar with.

Where rule functions are different is in their dealings with data types. The context in which a function is called sets the data type for that function call. A given function may support one or more data types, referred to as the function's "supported return types." If a function does not support the data type dictated by the context, the function will return the null equivalent for that data type.

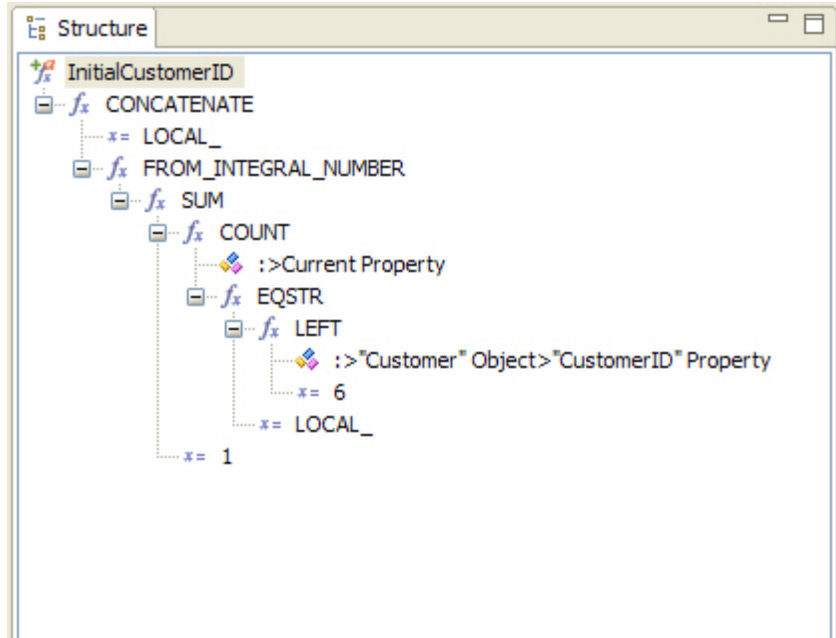
The context of a function call can also impact the behavior of the function's processing. This impact can include the data type of the function's parameters, as well as how those parameters are processed by the function. Many functions will take parameters matching the data type of the context in which they are called. These same functions will support multiple return types. This means that the function can take parameters of one data type in one context and another data type in another context.

Rule Evaluation at Run-Time

A rule is evaluated at run time on the Agentry Client when the definition referencing the rule calls it. Rules are a purely client-side definition, meaning they only directly impact the behavior of the client application. Rules have no effect on the behavior of the Agentry Server or on communications between Agentry Client and Agentry Server.

When the rule is called is dependent on the type of definition referencing the rule. Rules evaluated to set the label of a button will be called when the parent screen for that button is displayed. Initial value rules to initialize transaction properties will be evaluated when the transaction is instantiated, or possibly when the transaction is applied, depending on the definition of the property's initialization attributes. Other rule uses will result in different events resulting in a rule being evaluated.

The structure of the rule dictates the behavior of the evaluation. This structure is represented in a tree control in the Rule Editor within the Agentry Editor. Following is an example of this structure for a rule that creates an ID value for a new object instance on the Agentry Client:



This example is taken from the rule editor and is one of the components of that tool provided to developers for the purpose of defining a rule. In this example, the rule itself is represented by the root node of the tree control, which is named `InitialCustomerID`. As the only child node to this root is a function call to the function `CONCATENATE`. This function concatenates two or more string values, returning the result. The strings to concatenate are provided as parameters to the function.

In the example provided here the `CONCATENATE` function takes two parameters. The first is a constant value containing the text `LOCAL_`. This second parameter is a call to the function `FROM_INTEGRAL_NUMBER`, which is a conversion function that converts integral values to other data types.

Working down through this tree structure there are other rule terms for functions and data terms. At run time, this rule will be evaluated beginning with the innermost term. This term will then return a value to its caller, which will be a function. Each parameter to a function is evaluated in the order provided. In the above example, then, the first term evaluated is the first parameter to `COUNT`, which is a property denoted as `:>Current Property`. For this rule this refers to a collection of objects to be counted by the `COUNT` function. The second parameter to the `COUNT` function is then evaluated, which is a call to the function `EQSTR`. This function takes another function call as its first parameter, which is the function `LEFT`. Due to the nature of the `COUNT` function, the second parameter that is the `EQSTR` function

call will be evaluated once for each object within the collection referenced by the target path : >Current Property. Once COUNT's evaluation is complete, it will return a value to SUM, which will add this value to its second parameter, the constant value 1.

This evaluation continues back up to the top level term, which is the CONCATENATE function. The value returned by the CONCATENATE function will then be the value returned by the rule to the definition that called the rule.

Rule Context

The context in which a rule is called, as well as the context in which the rule terms are evaluated within a rule, will have a significant impact on the resulting processing of the rule and rule terms. The context of a rule definition describes where the rule is being used within the application at run time. The context is set by the definition referencing the rule within the application project and that calls the rule at run time. Included in the context of a rule's usage are the definition referencing the rule, the data type expected to be returned by the caller of the rule, data definitions such as objects, transactions, properties, and others that are in scope when the rule is called and how other data definitions are related to those within the application's overall data structure.

In addition to the caller of the rule, each function within the rule definition will also have an impact on the context, specifically on those rule terms passed to the function as parameters. A given function can dictate the context of the terms used as its parameters including what data definitions are in scope and what data type is expected of the term being evaluated as a function parameter.

The context of a rule and its functions affects the following:

- **Return Type** - Rule function terms do not have a set return data type. Rather, functions support one or more data types for their return values. The caller of the function will dictate which data type is to be returned, and the function will provide a value in that data type. If the function does not support the data type being asked for, the null equivalent of that data type is returned by the function.
- **Target Paths** - Any target paths within a rule are affected by that rule's context. If the property of an object is included in a rule, how that property is found and its value returned is affected by the context of the rule. A rule that contains a reference to an object property will not likely be one that can be reused for a different object. The target path to the property will be invalid.
- **Rule Function Behavior** - Many rule functions will behave differently based on the context in which the function is called. These differences can include both the data type of the function's parameters as well as how the function processes those parameters. Many functions with numeric parameters will evaluate those parameters as the same data type for which the function is being asked. A given function, then, can evaluate its parameters as integral numbers in one context and decimal numbers in another. Other functions may perform different processing in different contexts based on the data type of each context.

To define the term more precisely, context is the way in which a rule and its terms are called and that affects the behavior, data types, and target path resolution when that rule is evaluated

at run time. Context plays a role in the evaluation of each term within a rule, including function terms, data terms , and sub-rule terms.

Rule Data Types

Each term within a rule is evaluated and then returns a value. The value returned by a given term will be in the data type asked for by the caller of the term. All terms will return a value in one of the following data types, which are the only data types available within the rule structure:

- Boolean
- Integral Number
- Decimal Number
- String
- Location (or “GPS Location”)
- Property

This list does not restrict the types of values within the application that may be referenced in rules. Other data types, such as those found in property definitions, are converted to one of the above types when the term referencing the property is evaluated. The function terms available for use within the rule structure will return one or more of the above-listed data types.

Data Type Conversion in Rules

When a data term is evaluated within a rule, it will be asked for one of the data types available within rules. The native data type of the term’s source may not be one of these six data types. This is most likely to be true when working with property or global definitions. There are far more data types for each of these definition types than those for the rule definition.

When a data term is evaluated within a rule and the data type it is asked for is not one of the six for a rule definition, the value of that term will be converted to one of the rule data types assuming that data term supports such a conversion.

The following table contains a cross reference of all property data types and rule data types. Each rule type and property type intersection in the table indicates whether or not the property type can be converted to the rule data type:

From Property Data Type To...	Boolean	Integral Number	Decimal Number	String	Location	Property
Boolean	Yes	No	No	Yes*	No	No
Collection	No	No	No	No	No	Yes
Complex Table Selection	No	No	No	Yes	No	No

From Property Data Type To...	Boolean	Integral Number	Decimal Number	String	Location	Property
Data Table Selection	No	No	No	Yes	No	No
Date	No	Yes*	Yes*	Yes	No	No
Date and Time	No	Yes*	Yes*	Yes	No	No
Decimal Number	Yes*	Yes*	Yes	Yes	No	No
Duration	No	Yes	Yes	Yes	No	No
External Data	No	No	No	Yes*	No	No
Identifier	No	Yes	Yes	Yes	No	No
Image	No	No	No	No	No	No
Integral Number	Yes*	Yes	Yes	Yes	No	No
Location	No	No	No	No	Yes	No
Object	No	No	No	No	No	Yes
Signature	No	No	No	No	No	No
String	Yes*	Yes	Yes	Yes	No	No
Time	No	Yes*	Yes*	Yes	No	No

* - This conversion may not be type safe or requires further explanation on the resulting value from such a conversion. See the description of the rule data type for more information on this conversion.

Boolean Rule Data Type

The Boolean data type within rules is similar to Booleans in all areas of software development, containing a value of true or false.

When converting from an integral or decimal number property type to a Boolean rule type a value of zero is treated as false and a value other than zero is treated as true.

When converting from a string property type to a Boolean rule type, the value of the Boolean will be set to true if the string value is “true.” Any other string value is treated as false.

Integral Number Rule Data Type

The integral number data type within rules stores whole positive and negative values, or zero. This is a 32-bit integer value.

When converting from a date property type, or any other data source of type date, to an integral number rule type the value returned will be the number of days from the epoch date of January

1, 1901. Positive numbers represent the number of days after this date and negative numbers are dates before it.

When converting from a time property type, or any other time data source, to an integral number rule type the value returned will be the number of seconds after midnight. This will always be a positive integer.

When converting from a date and time property type, or any other date and time data source, to an integral number rule type the value returned will be the number of seconds from the Agentry epoch date and time of January 1, 1901 12:00:00 am. A positive number represents a date and time after the epoch date and time, and negative numbers represent a date and time before.

A decimal number property can be converted to an integral number rule type. However this conversion is not considered type safe. Any fractional portion within the source decimal number will be truncated from the resulting integral number.

Decimal Number Rule Data Type

The decimal number rule data type stores numeric values with a fractional portion. This is the equivalent to a 32-bit floating point decimal number.

When converting from a date property type, or any other data source of type date, to a decimal number rule type the value returned will be the number of days from the epoch date of January 1, 1901. Positive numbers represent the number of days after this date and negative numbers are dates before it.

When converting from a time property type, or any other time data source, to a decimal number rule type the value returned will be the number of seconds after midnight. This will always be a positive integer.

When converting from a date and time property type, or any other date and time data source, to a decimal number rule type the value returned will be the number of seconds from the Agentry epoch date and time of January 1, 1901 12:00:00 am. A positive number represents a date and time after the epoch date and time, and negative numbers represent a date and time before.

String Rule Data Type

A string rule data type stores one or more unicode characters as a string value.

When converting a Boolean property to a string rule type, the resulting value of the string will depend on the definition of the Boolean property. The attributes within the Boolean property **True Value** and **False Value** contain the text value returned by that property in a string context. A data term for a Boolean property evaluated in a string context will then return the appropriate string depending whether or not the property is set to true or false.

When converting an external data property to a string rule type, the return value will be the full path and file name for the file referenced by the property. If the property does not reference a file, an empty string is returned.

Location Rule Data Type

The location rule type stores a value returned from a GPS unit that includes the latitude, longitude, number of satellites, and precision of the location value. This data type cannot be converted to other data types within the rule and other data types cannot be converted to a location.

There are specific rule functions within the System category of rules provided to work with the Location data type. These include functions for converting two decimal values assumed to be latitude and longitude coordinates to a location value, as well as those for calculating distances between two location values, and a function to retrieve a GPS location from the GPS unit for the client device.

Property Rule Data Type

The property rule data type is unique among the different data types within rules. A property type is the term applied to any point within the rule where a definition is expected. Certain functions are provided to allow for searching object collections, or to work with external data properties. These functions will take one or more parameters of type property and may also return a value of this type.

Many of the different functions for these types are intended for use with certain types of definitions within the application. The information for these functions indicate the expected definition type. The important concept for a property rule type is that what is returned from such a term is the definition itself. When a target path references a definition for a caller expecting a property rule type, that definition is returned, not just its value.

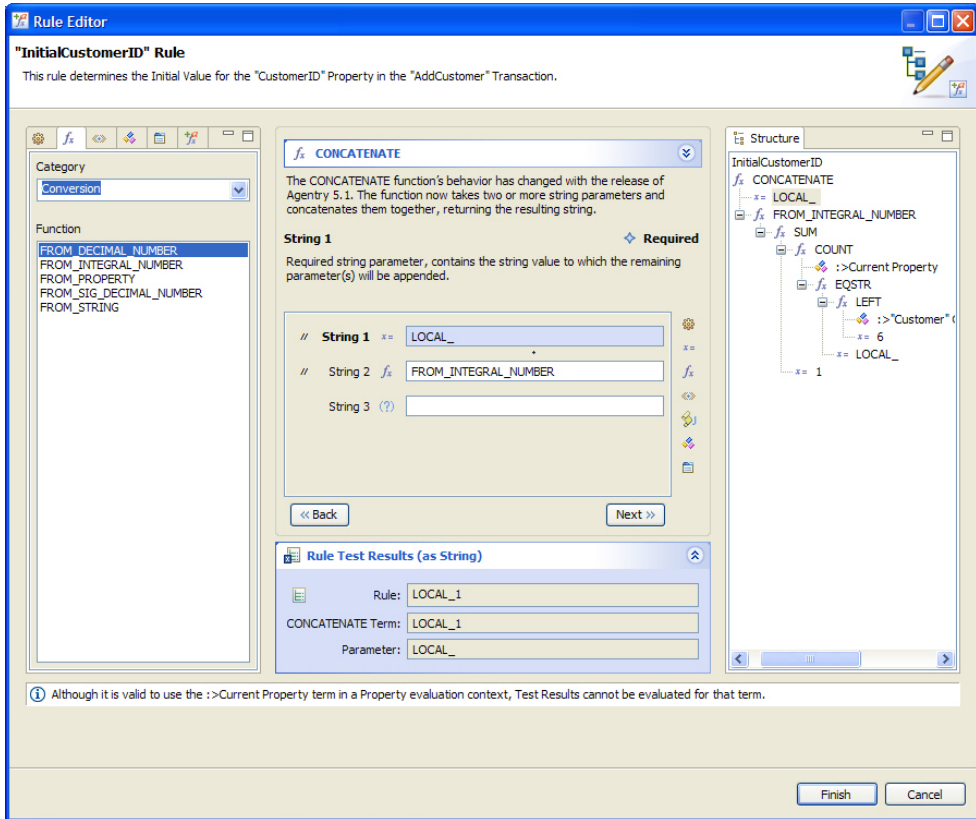
Rule Editor Introduction

Rule definitions within an Agentry application project are added to the module and defined using the Rule Editor. The Rule Editor is a tool used within the Agentry Editor to define the rule logic. It includes several tools and features to aid in the definition of this logic, including:

- Functions are presented with fields for each parameter to the function, with appropriate indicators for optional and required parameters, as well as the data type of the parameter.
- On-line help for each rule function term, including short and long descriptions of the function and descriptions of each function parameter displayed in the Rule Editor for each item as it is used.
- Test functionality providing the ability to test rules within the rule editor by providing test data for function parameters and viewing the return from the rule based on such values.
- Real-time rule structure validation and context information for the rule as a whole and each of its terms.
- Navigation items to make the different rule terms more readily accessible.

Overview of the Rule Editor

Following is an example of the Rule Editor. It displays a simple initial value rule that generates local ID's for new object instances created on the Agentry Client.



On the left of this screen there is a list of all items that may be added as rule terms. The various tabs allow for the selection of actions, functions, globals, properties, screen sets, and sub-rules. Selecting one of these tabs will then list the items of that type. When functions are selected a drop down list is displayed to allow the developer to select the function category, which will list only those functions within that category. There is also the option to list all functions.

The main center portion of the screen displays the function currently being added or modified. Included here is the short description of the function, as well as fields for each of the function's parameters. Selecting a parameter field will display the description of that parameter above the parameters list.

To the right of the parameters list are shortcut buttons. These buttons allow for adding the same terms as the list on the left. When the shortcut button is selected, a menu is displayed with the items of that type. In addition is a button to include a JavaScript term. This should only be used with the JAVASCRIPT function and is provided to allow for the entry of JavaScript in a multi-line editable text box.

Below the list of field parameters is the Rule Test Results section, which may be expanded or collapsed (expanded in the example). This section can be used to test the results of the rule by specifying test values for various terms within the rule.

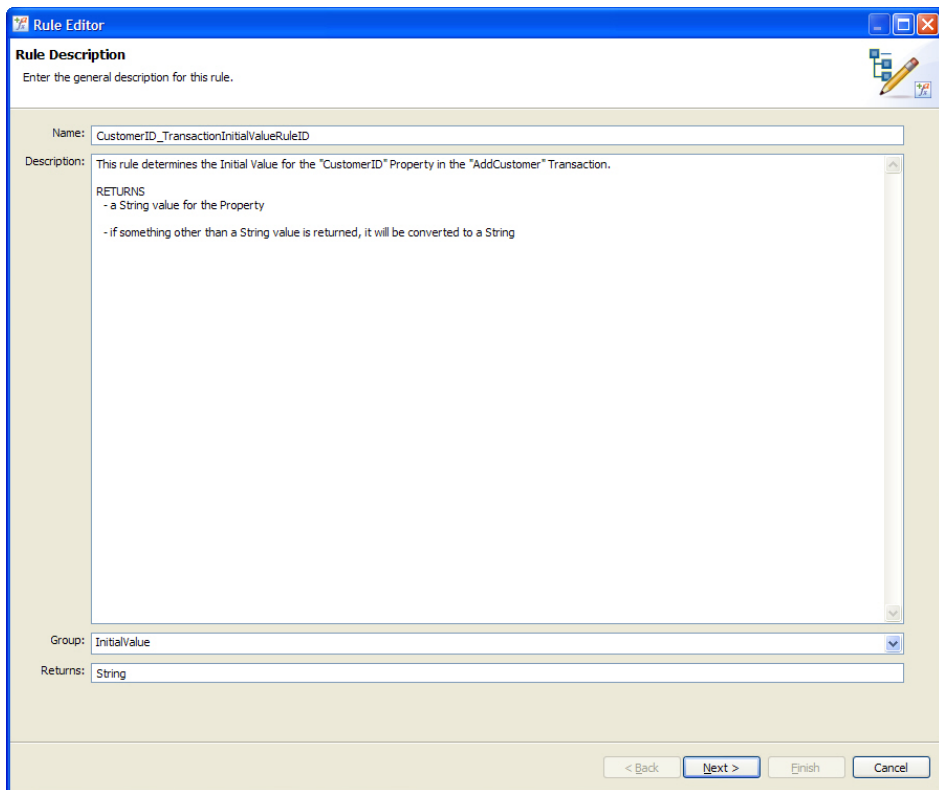
To the right of the screen is the Structure view for the rule. This displays the entire rule structure. For developers familiar with the rule editor in previous releases of Agentry, this view presents the rule and allows for the same functionality. Within this structure view, the rule terms may be added, edited, deleted, or dragged and dropped to different positions within the structure.

Creating Rule Definitions

To define a new rule definition within the Editor, the Rule Wizard and Rule Editor are used. The Rule Wizard is displayed to capture the attributes for the new rule definition, including the name and group. The Rule Editor is displayed next to allow for the definition of the rule's structure. This procedure uses an initial value rule for a transaction property as an example. The same process is followed to define a rule regardless of where it is to be used within the application.

1. Start the Add Rule Wizard by selecting the Add Rule menu item in the menu displayed for the attribute to reference the rule definition. This is normally an ellipses button to the right of the attribute field.

The first screen of the wizard is displayed with a default name and group, based on the definition referencing the new rule definition:



The screenshot shows the 'Rule Editor' window with the 'Rule Description' tab selected. The window has a blue title bar and standard Windows window controls. The main area is divided into sections for 'Name', 'Description', 'Group', and 'Returns'. The 'Name' field contains 'CustomerID_TransactionInitialValueRuleID'. The 'Description' field contains a multi-line text description. The 'Group' dropdown is set to 'InitialValue'. The 'Returns' field is set to 'String'. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Rule Editor

Rule Description
Enter the general description for this rule.

Name: CustomerID_TransactionInitialValueRuleID

Description: This rule determines the Initial Value for the "CustomerID" Property in the "AddCustomer" Transaction.

RETURNS
- a String value for the Property
- if something other than a String value is returned, it will be converted to a String

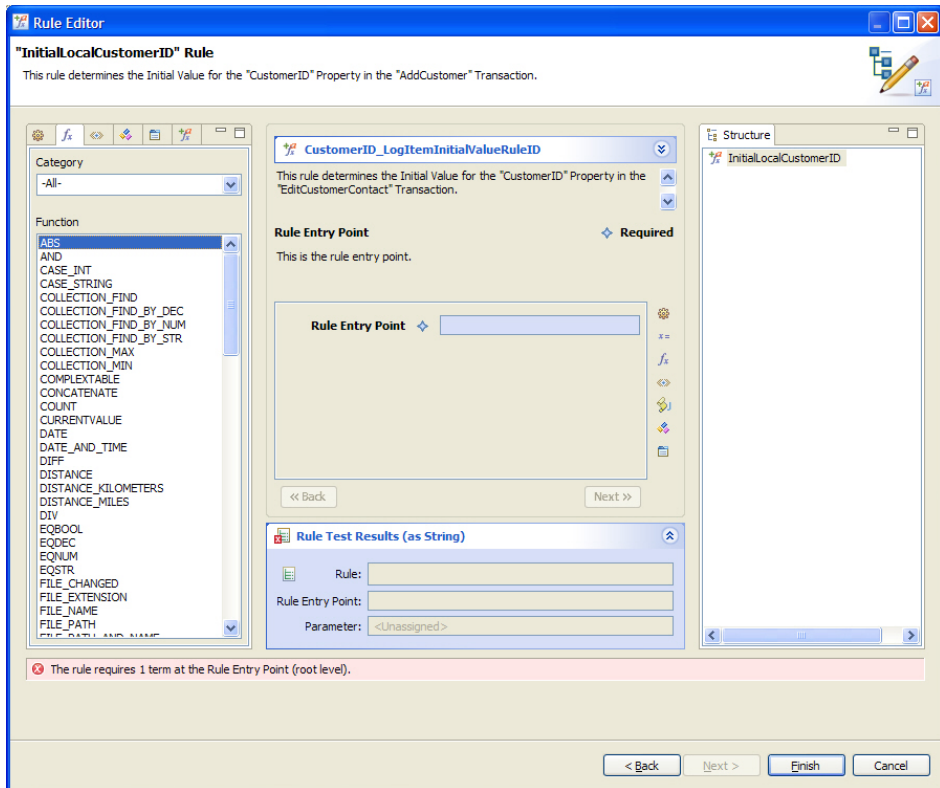
Group: InitialValue

Returns: String

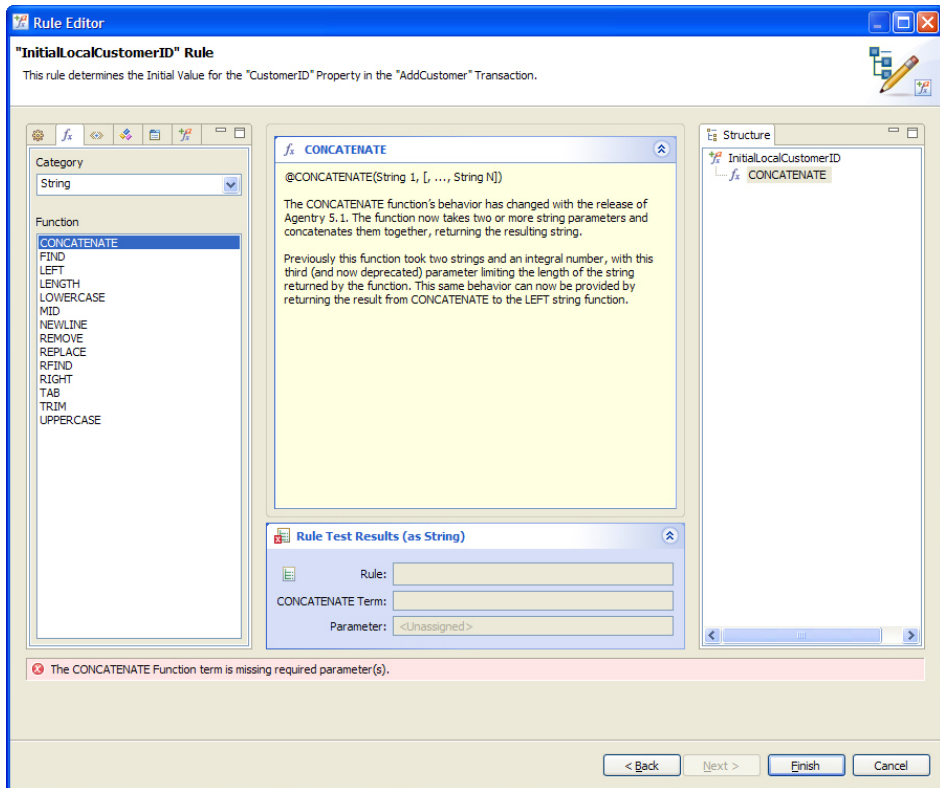
< Back **Next >** Finish Cancel

2. Set the Name and Group attributes as desired for the new rule definition. The Description field may also be edited. The default text display is based on how the rule is to be used. The Returns field is read-only and specifies the data type of the value to be returned by the rule when it is evaluated. Click the **[Next >]** button to advance to the Rule Editor.

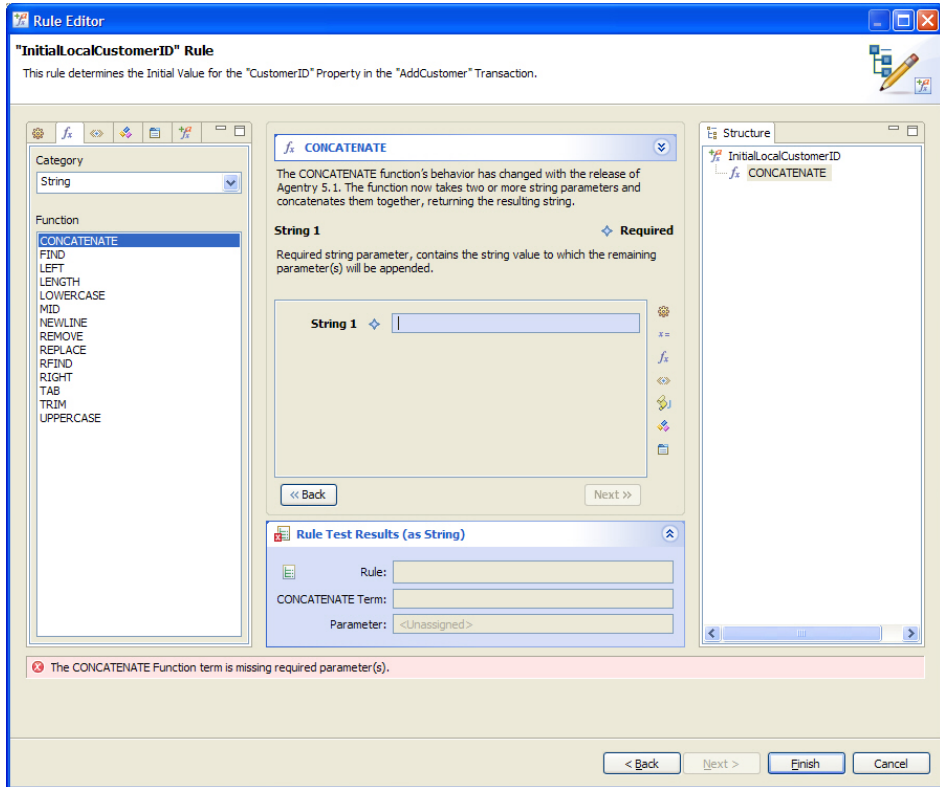
The second screen of the Rule Editor is displayed with the Rule Entry Point selected:



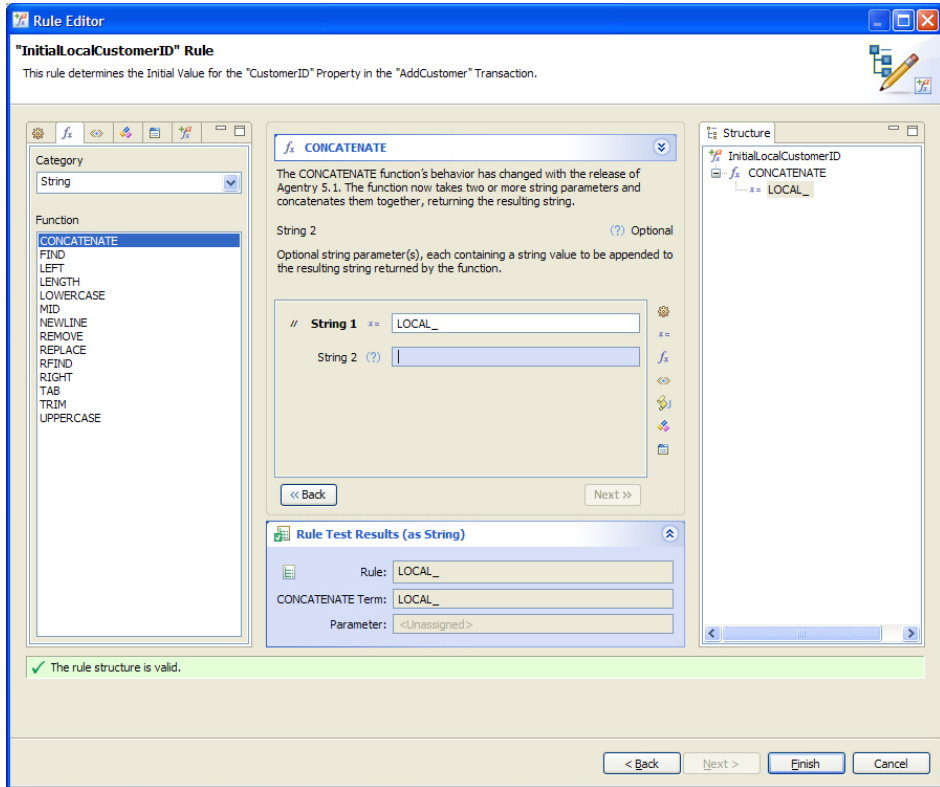
3. To begin defining the rule structure, begin by selecting the field Rule Entry Point. This is the first term for the new rule definition. Selecting this field will allow for the addition of a rule term, either a function or a data term. This term's return value will be the value returned by the rule at run time.
4. To add a term to the entry point in the rule, select an item from the list of terms on the left. By default the list displays the available rule functions. In most cases this is a rule function. Other options include an action, global, property, screen set, or sub rule. To change the list of terms to select from, select one of the available tabs above the list.
5. If any term is selected other than a function, the rule's definition is complete, as no other terms can be added below a data term within the rule's Structure. If a function is selected from the list, the editor will display that function in the center of the screen, with fields listing the function's parameters. The function name followed by its short description is shown at the top-center of the screen. Clicking the name of the function, or the arrows to its right will display the function's long description.



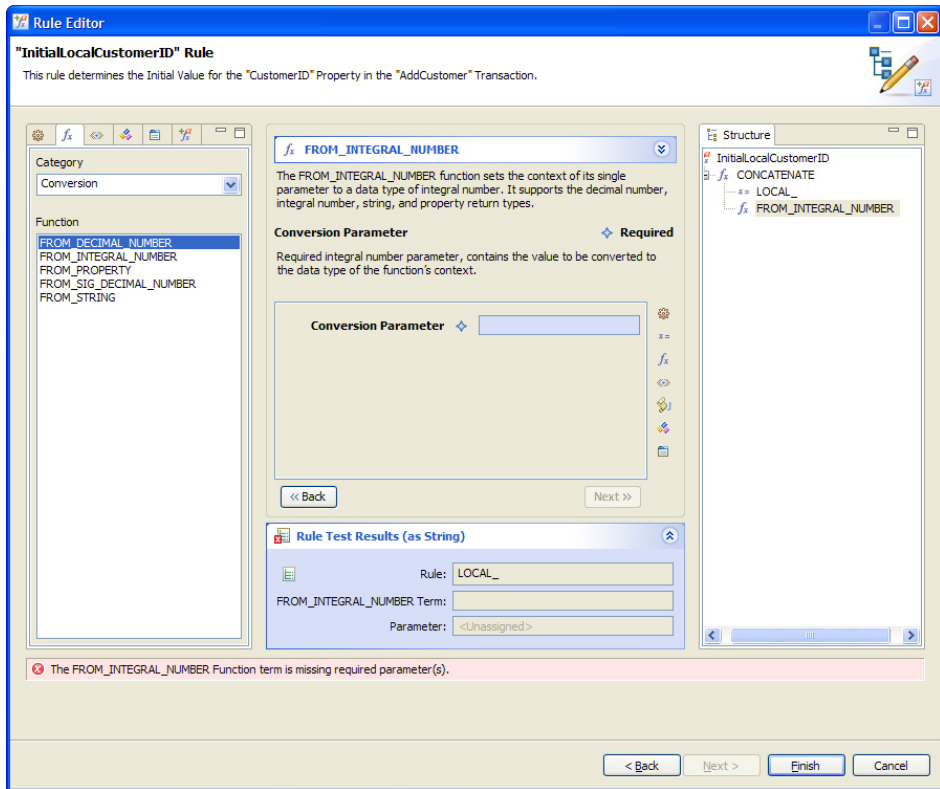
- When a parameter field is selected in the Rule Editor the description of that parameter is displayed. Selecting a field will display the list of terms on the left. Any term that supports the return type for the parameter may be selected:



7. To add a constant value as a parameter to the function simply type that value in the parameter field.



8. To add any other term type, select it in one of the lists on the left side by double-clicking it. If a function is added as a parameter to the current function, that function will then be displayed in the middle portion of the screen, along with its short description and list of parameter fields.



9. At this point the process is repeated until the structure of the rule has been defined.

Note the structure view to the right of the rule editor. As functions are added, their position within the overall rule structure is represented. This structure view can be used in the definition of the rule as well. Right-clicking on any function in the rule structure displays a popup menu allowing for the addition of parameters to this functions. The menu also provides options to replace terms and delete them. Additionally, terms may be dragged and dropped to different locations within the structure if it is desired to modify the rule in this manner.

Next

Once the rule has been defined it can be tested within the Rule Editor. This can be done before finishing the Rule Editor, or the developer can return to the definition later and perform any testing.

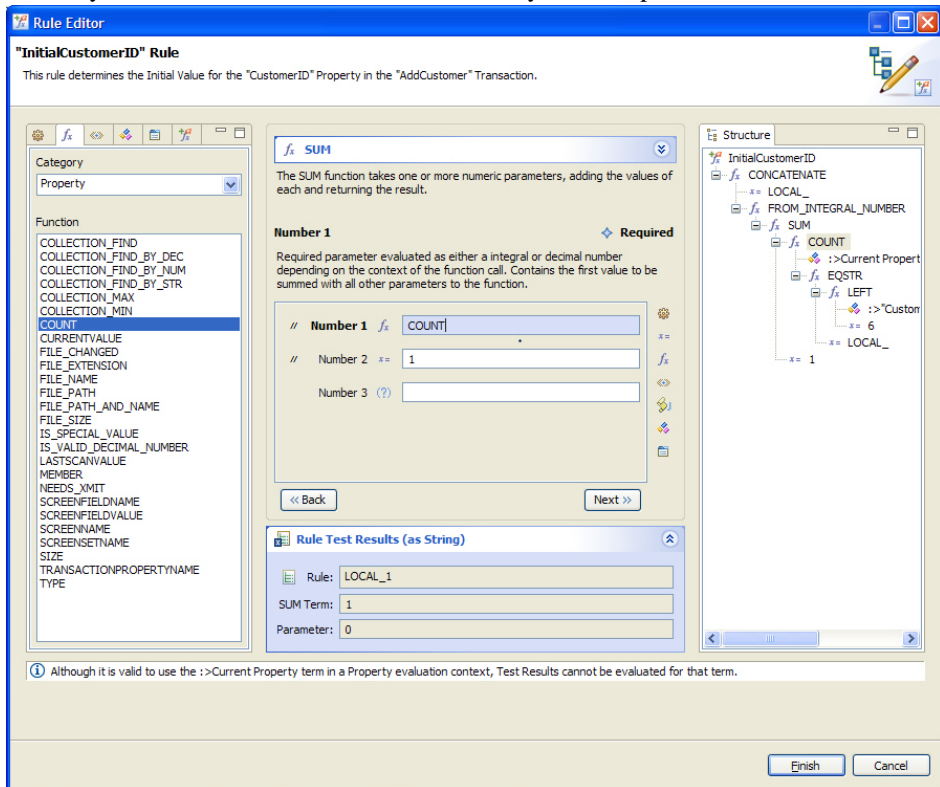
Testing Rules in the Rule Editor

The rule editor, in addition to providing the interface to define and edit rule definitions, also provides tools for testing rule definitions. Within the Rule Editor, below the list of function parameters, is an expandable frame labeled “Rule Test Results.” Expanding this frame will display the return value for the rule as currently defined, with certain assumptions made about

return values within the rule. Using this test frame, the developer can select a parameter and then provide a test value for that parameter. The return value of the rule will then be determined and displayed, using the new test value.

1. Within the Rule Editor, expand the Rule Test Results frame. Within the Structure view select the rule function term for which a parameter test value is to be specified. The function will be displayed in the middle of the screen. Select the parameter for which a test value is to be entered.

Within the test results pane, fields are displayed for the values returned by the rule, the currently selected function term, and the currently selected parameter to that function.



2. To change the test value for the selected parameter, click the button to the left of field Rule within the Rule Test Results frame.

A screen is displayed allowing for the entry of a test value. When entered, the return values for the function term and the rule will be updated automatically:

3. The Rule field displays the value the rule will return based on the test data entered. The line below it displays the value returned by the function, again based on the entered test data. Below these fields is the section where the test data can be entered. The fields displayed here include all values under the selected term which may be replaced with test data. Excluded from this list are constant values and object collection properties. Changing any of the values will automatically update the return value from the selected function term and the rule as a whole.

The results of entering test data will not affect the rule definition in any way. However, proper use of the test functionality can significantly improve the stability of the rule before it is published for development and unit testing. Additionally, areas of logic for which the proper structure is unclear can be made easier when using the test data to determine if the expected values are actually returned by the logic within the structure. Note that while this functionality is a powerful development tool, it is not intended to replace the proper run-time testing that should be a part of any software development or product implementation project.

Syclo Data Markup Language

When synchronizing data between the mobile application and the back end system, it is necessary to have access to the mobile application's data values. This access is provided in Agentry using the Syclo Data Markup Language, or SDML. The SDML is a markup language consisting of tags that provide access to the data values of the mobile application. Additionally, the SDML includes a full set of functions, or function tags, that can be used to perform logical operations in relation to this values or to drive the overall logic the Agentry Server will execute against the back end system.

The SDML tags used during synchronization are a part of the text within the scripts for step definitions defined for SQL Database, HTTP-XML, and File system connection types. Also, the synchronization components of data tables and complex tables for each of these system connection types can contain SDML. In addition to SQL Step definitions, other .sql script files run by the Server may also contain SDML tags. Steps defined for Java Virtual Machine system connections also include the ability to access SDML tags, but these tags may not be contained directly in the source code of the Java Steplet files used by these steps.

The Agentry Server will pre-process the script files of steps containing SDML markup. This processing is referred to as tag expansion. Each tag within the script is expanded, with the value it represents replacing the tag at the exact position of that tag within the file. Function tags are expanded with the results of their expansion being placed in the exact position of the

function call within the file. Once the tag expansion has completed, the resulting text is submitted to the back end system for processing.

The two categories of tags within the SDML are data tags and function tags. Data tags represent data values available to the script file based on when it is executed. This information must be known when writing the script in which the SDML will be contained. For step definitions the values in scope are dictated by the step usage definition running them. For .sql scripts run by the server, but not a part of the step definition, the values in scope will vary depending on how that script is used. Certain values are globally available, such as the user ID as entered by the user to log into the Agentry Client.

Function tags are globally available, with certain exceptions. Function tags provide the logical, mathematical, string manipulation, and other similar functionality to the SDML. Function tags can take values passed in as arguments, parameters, or expressions. These values are processed by the function during tag expansion, with the resulting value of the function call being placed within the script.

Following is a basic example of a simple SQL statement containing SDML data tags:

```
SELECT
    A.FIELD1,
    B.FIELD2,
    C.FIELD3,
FROM
    TABLEA A,
    TABLEB B,
    TABLEC C
WHERE
    A.NAME = '<<user.agentryID>>' AND
    A.ACCTNUM = '<<object.acctnum>>' AND
    B.ACCTNUM = A.ACCTNUM AND
    C.ACCTNUM = B.ACCTNUM
```

In this example, the value <<user.agentryID>> is replaced with the user ID as entered when the user logged into the Agentry Client. The data tag <<object.acctnum>> will be replaced with the value of the acctnum property of the object currently being processed.

SDML Syntax and Data Tag Expansion

The SDML is a markup language containing data tags and function tags. The basic syntax for a tag is to enclose a given tag within the tag markers << >> to denote it as an SDML tag. Within these tag markers is the name of the tag, as well as any values that may be a part of the tags expansion processing. Following is the general form of data tag and function tag syntax:

Data Tag

```
<<parent.tagName.parameter namedParameter="value">>
```

Function Tag

```
<<functionName argument "expression" namedParameter="value">>
```

Named parameters to both functions and data tags have the specific requirement that the parameter name, equal sign, and the value can not be separated by any white space:

Correct Parameter Syntax

```
<<functionName namedParameter="value">>
```

Incorrect Parameter Syntax

```
<<functionName namedParameter = "value">>
```

Depending on the nature of the SDML logic and processing needed, it is common for one tag to be nested within another. When this is the case the end markers for such tags may be adjacent. In this situation at least one white space character must be used to separate the two end markers. This may be a space, tab, or a newline:

Incorrect

```
>>>>
```

Single Space

```
>> >>
```

Tab

```
>>      >>
```

Newline

```
>>
>>
```

Data Tag Syntax

Values within a given tag will depend upon a number of different factors. Data tags generally take one of the following forms:

```
<<parent.tagName namedParameter=value>>
<<parent.tagName.parameter>>
```

The parameter and named parameter for a given data tag will depend on the data type of that tag. Some may have no parameters or named parameters, others may support one or both. A parameter generally provides access to different contents of the data tag's value, such as raw or string. A named parameter generally provides formatting instructions for how the value should appear when the data tag is expanded.

The value for a named parameter may be a hard coded value or another tag within the SDML. When the value is plain text, it must be enclosed in double quotes. When the value to the parameter is another SDML tag it cannot be enclosed in double quotes.

Parameter from Plain Text

```
<<parent.tagName namedParameter="value">>
```

Parameter From Data Tag

```
<<parent.tagName namedParameter=<<tagName>> >>
```

Function Tag Syntax

Function tags within the SDML take the general form:

```
<<functionTag argument "expression" namedParameter=value>>
```

A given function may take multiple arguments, expressions, and/or named parameters, or it may not take any of these depending on that function's prototype and purpose. Separating each of these values to the function is one or more white space characters.

Arguments should be enclosed in tag markers if a data tag is used, unless specified otherwise for a given function. Certain functions, notably `<<if...>>`, `<<foreach...>>`, and `<<case...>>` specify that if the first argument is a data tag it cannot be enclosed in tag markers, but rather should only be the name of the data tag. If the argument is a hard coded value it should be enclosed in double quotes.

Expressions are always enclosed in double quotes, regardless of whether or not they contain SDML tags. The contents of a given expression can span multiple lines, which is often the case as expressions tend to be longer text values.

The value for the named parameter can be a data tag, in which case the tag should be enclosed in tag markers. If the value is a hard coded value it must be enclosed in double quotes. The value for a named parameter can contain white space within the double quotes and can be a combination of SDML tags and plain text.

These values for a function tag can span multiple lines, with the opening marker preceding the function name and the closing marker somewhere after all specified values for the function, as in:

```
<<functionTag
    argument
    "expression"
    namedParameter=value
>>
```

When arguments or named parameters contain function or data tags, those tags will be expanded before being passed to the function for processing. Expressions containing tags will not be processed until the function returns that expression.

SDML Expansion

At run time, when the Agentry Server processes a script file the tags it contains are parsed and expanded. This process is called SDML expansion and occurs for all scripts not using a Java Virtual Machine system connection that are run by the Server.

Tags are expanded in a top-down, inside-out order. This means that each line of a script is processed starting with the first in the file and working in order to the last line. When a line is processed, the data tags are expanded from the innermost tag to the outermost one. Consider the following example:

```
3.....2.....1          .....
1a.....                  4
```

```
<<if <<ne <<object.acctnum>> <<parent.acctnum>> >> "not equal" >>
```

Ignore the numerical notations for the moment, as they are for reference purposes only and not a part of the SDML text. This line says that if the values of the `acctnum` properties in the object and the parent of the object are not equal to return the string “not equal”. This begins with the `<<if . . .>>` function. The single argument to this function is `<<ne . . .>>`, which is another function whose name is short for “not equal”. The `<<ne . . .>>` function takes two arguments that are compared for equality. The two arguments in the example are both data tags for property values within objects.

In this case, the expansion goes as follows. First, the two arguments, noted as 1 and 1a, are expanded. These are data tags, so the tags are replaced with the values of the two `acctnum` properties in the object being processed and the parent of that object. If the object property `acctnum` has a value of 1234 and the parent has a value of 1122, the line would expand as follows:

```
<<if <<ne 1234 1122 >> "not equal" >>
```

Next, the `<<ne . . .>>` function is expanded. The two values of 1234 and 1122 are passed as arguments to this function. The function then compares the two values and determines they are not equal. The line would now then appear as:`<<if true "not equal" >>`

The `<<if . . .>>` function, which provides if-then-else logic, takes the return value from the `<<ne . . .>>` function value as an argument. The “not equal” text is the expression that is returned when the argument is true. In the example provided the resulting text placed at the point of the `<<if . . .>>` function call will be the text: not equal.

SDML Syntax Quick Reference

Following is a quick reference of the basic syntax rules for the Syclo Data Markup Language:

- Function and data tags are enclosed in the tag markers <<tagname>>. Named parameters to both function and data tags, as well as function arguments and expressions are enclosed within the same set of tag markers.
- Hard coded values passed to parameters or arguments are enclosed in double quotes. Data or function tags are enclosed in tag markers and should never be enclosed in quotes.
- Expressions are always enclosed in double quotes, whether or not they contain tags. Tags within an expression are also enclosed in tag markers.
- Named parameters for both function and data tags take the form of a key and value pair separated by an equal sign (=). No white space can exist between the named parameter, equal sign, and the beginning of the value for the parameter. The parameter itself can contain white space and, when it does it should be enclosed in double quotes.
- Adjacent end tag markers must be separated by at least one white space character. Excluding this character will result in an error during tag expansion.
- Function tags can span multiple lines in a script file. This is commonly the case with expressions.

Agentry Data Definitions Overview

In any application the data structures and the processes to synchronize the production data are the foundation of the functionality. Within an Agentry application project there are three definition types intended to define data stored on the Client: objects, complex tables and data tables. All three define the data for the application and also include components for synchronizing data with the back end system.

Objects exist at the module level and complex and data tables are defined at the application level. Complex tables and data tables are defined to store lists of records on the Client and normally contain values displayed to the users in lists or other controls from which they can make selections. Objects store the production data for modules and normally encapsulate some business entity.

While objects, complex tables and data tables are the main data definitions, there are others related to the storage and synchronization of data on the Client. The first of these are object properties. An object property defines a single piece of data for the parent object. Note that there are also transaction properties, which are similar to object properties but are defined within a transaction. The discussion of properties here will focus on object properties.

Other definitions related to production data are those defined to synchronize the data. The primary definition for data synchronization is the step definition. A step defines a piece of processing to be performed by the Agentry Server with a specific back end system. Steps are used by other definitions that are processed during data synchronization. This allows for reusability as well as the multi-system support provided by Agentry. Steps are defined to synchronize data stored in objects and transactions.

The synchronization process for complex and data tables is defined within the definitions themselves. Both complex tables and data tables contain components responsible for the downstream, or back end-to-client data synchronization.

Data Synchronization Overview: The Exchange Data Model

When developing mobile software solutions, one of the primary considerations is the most efficient way in which to synchronize production data. While all client-server systems must account for this, mobile software development presents its own set of challenges, which stem from the almost universal truth that, at some point, mobile users will need to work in a disconnected environment.

Users will not always be connected, and in many cases will spend most of their day without network connectivity. Therefore, when users do synchronize their clients, information must be resolved concerning what data a user needs. It can be extremely inefficient to attempt to retrieve all production data during synchronization. Most production applications contain large amounts of data stored on the client devices and attempting to retrieve everything during synchronization can result in long delays during the synchronization process. This is unnecessary in most environments, as much of the production data stored on the client is likely to still be current and accurate as compared to the data in the back end system.

An Agentry application project accounts for this within its structure. The architecture of all synchronization components allows the developer to be far more selective about what data needs to be retrieved during the synchronization process. The method recommended by Syclo is called the Exchange Data Model.

Exchange data is the term used to refer to information about what production data the client has and when it was last retrieved, as well as the data contained in the back end system and when it was last modified. With this information available the developer can implement synchronization processes that only retrieve information that has been modified since the last time a client synchronized with the Server. Any unchanged data is not retrieved. This model will result in quicker and more efficient synchronization for users, as well as reducing the amount of resources needed by the system as a whole during the synchronization process. All data definition types and their related synchronization components allow for and are intended to be used in an exchange data model.

The use of the exchange data model requires certain information be available during synchronization. This information can include:

- The date and time when an object, complex table record, or data table was last downloaded to the Client.
- The date and time when the data in the back end system was added or last modified.

For the date and time of data retrieval on the clients, the synchronization processing within Agentry provides the ability to retrieve, store and access the client-side information about when data was last retrieved. Objects, complex tables and data tables all have the ability to store what is called the “last update” value that represents the date and time data was retrieved.

For date and time values related to changes made on the back end system, mechanisms must exist or be added for the mobile application to track changes to data that occur in between users’ synchronizations.

During synchronization the exchange data about which data has been retrieved, changed, and added, and when those events occurred is put to use according to the following general process. Note that this applies only to downstream synchronization:

1. The Agentry Client sends the information to the Agentry Server about what data it currently contains and when it was retrieved. This includes unique identifiers and the last update values for each data instance.
2. The Agentry Server processes the client-side exchange information according to the synchronization definitions. This can involve adding the data to back end objects created specifically for the mobile data synchronization, or by using existing back end objects that suit these purposes.
3. The Agentry Server next processes the synchronization definitions that determine what has changed in the back end system since the date and time for the client-side data. Comparisons are made between the client's date and time values and the date and time values in the back end system that reflect when the back end data was last affected. Items with date and time values more recent than matching items on the client, or items added to the back end not currently residing on the client, are flagged for retrieval. Alternately, and depending on the synchronization methods specific to the type of back end system, the comparison and retrieval may be accomplished at the same time. This data is retrieved using synchronization definitions and returned to the Agentry Server.
4. Data that should be removed from the client is determined separately from data that should replace or be added to the client. Definition types within the Agentry architecture exist to specifically look for and return items that should be removed from the client.
5. The Agentry Server builds object instances, or complex table and data table records based on data returned to it. These instances are then sent to the client to be stored in their respective structures.
6. Data to be removed from the client is denoted via its unique identifiers. The ID's are sent by the Agentry Server to the client. The messaging sent includes instruction to the client to remove the denoted item from its respective data structure. This includes deleting object instances or removing complex table records. Note that individual records cannot be deleted from data tables, for reasons explained in the discussions specific to this definition type.

The specific methods and mechanisms for accomplishing the above tasks will differ as a result of a combination of different factors that include the type of production data being synchronized (objects, complex tables, or data tables), the type of back end system in use, the capabilities of a specific back end system, the specific needs of a given application, and the specific needs of an implementation of a given application.

Regardless of the technical details of how the above steps are accomplished, the following summary of the exchange data model holds true. Begin by determining what the client has and when it received it. Next use this information to determine what is different in the back end system and when it was changed. Finally, retrieve only the data which is different on the back end. Any other data in the back end system can be ignored as it has not been modified and therefore is accurate and still current on the client.

Data Synchronization: Data Filtering Overview

When developing a mobile application the concept of data filtering should always be at the forefront of the developer's mind during all phases of the development life-cycle. Data filtering is the term used to refer to filtering the data provided to the mobile user so that unnecessary and unneeded data is not retrieved. When the proper data is retrieved for the user and unneeded data is excluded, the application ultimately provided will be far easier for the end users, and will operate more efficiently during synchronization and client-side operations.

The need for data filtering in mobile software development is driven by two main factors. First, while mobile devices continue to become more powerful and more sophisticated, they do not have the same capabilities as a traditional personal computer or work station. Attempts to store large amounts of data on such devices can result in, at the least, poor performance of the application, and at worst the client device can become overwhelmed and not function at all.

The second factor in the need for data filtering is the end user. Many users of mobile software need only certain information concerning a particular business entity. Additional information can result in a cluttered user interface as well as confusion on the part of the end users.

For these reasons as well as others the concept of data filtering can and should be applied to all areas of the application design and development process as it relates to the data structures.

Overall there are many areas in which data filtering should be applied and there are often many options available on how to do this. Which is used will depend on the type of data and what information is available about the data in the back end system. It is important to remember that the client device is not a permanent data store for the enterprise system in use. Rather, it is both a snapshot and a subset of data from that enterprise system.

Object Data Filtering: Property Definitions

Objects contain the child definition property. A property stores a single value for the object. A given object will contain multiple properties. When designing an object and the properties it will contain, it is important to consider what data the user actually needs.

Using a database back end system as an example, where an object is defined to contain data from a given table, the developer should always consider what data the end user will need from that table. Many database tables in a back end system can contain dozens or more columns. This data is necessary for the records within the table and is likely related to multiple processes. Examples of these processes and needs can include performance reporting, accounting requirements, change tracking, and auditing. However, much of this data is not needed by the end user. It will not be displayed to them nor captured from them on the Client. Furthermore it is not needed during downstream or upstream synchronization.

Because of this, there is no need to retrieve this data from the back end system. Though a given value may be small in size for a particular object, remember that it is likely that there will be dozens or hundreds of instances of a given object stored on the Client at a given time. As a result, a single unneeded value for an object can result in significant wasted resources on the

Client for storage, as well as unnecessary bandwidth and processing being consumed during synchronization. These same statements can be made about any object for any type of back end system.

Complex Table Data Filtering: Field Definitions

Furthermore, complex tables should also be designed and developed with data filtering in mind. Complex tables contain field definitions, with each record in the table containing the fields defined for the table. Like properties, a single unneeded complex table field can result in significant wasted resources on the client device. With complex tables, however, the resources wasted can be even more detrimental than with objects. While there may be hundreds of instances of a given object stored on the Client, there can be thousands of complex table records.

User-Specific Data and Data Filtering

Taking this a step further, entire object instances or complex table records may be excluded from the Client if the proper design and development considerations are applied as they relate to data filtering. Whenever possible the developer should consider what data can be user specific. For objects this tends to be the case most of the time. Objects are usually defined to encapsulate business entities in the back end system that are user specific. A work order is assigned to a specific technician. A customer is assigned to a single account executive. These are two examples of what would normally be object definitions in a mobile application.

Complex tables tend to store data that is applicable to multiple users. Complex tables may contain records of inventory items available to be ordered by any customer, or assets for the company that one of many technicians may work with. However, there are still ways to filter this data. First, in some cases the data of a complex table may be user specific. In this case the data can easily be filtered for a given user during synchronization.

In other more common situations, the data is not user specific. In these cases the developer should look for other ways to partition data. Some suggestions can include the location a user may work in can mean certain records will never be needed. If a technician works in location A, then the complex table containing assets need only contain the assets that reside in location A. Similarly inventory items customers may order can also be filtered. If an account manager services customers in a specific industry or of a certain type, there may be items within the inventory that those customers will not order. Assuming the inventory information can be cross referenced with an industry or customer type, records can be excluded from the complex table containing that data on the Client.

Object Development Concepts and Considerations

An object definition encapsulates a business entity and its related data. An object's child property definitions give that object its characteristics. An object can also define how its data is retrieved from the back end system. The object definition is the primary data definition for modules. At run time objects are instantiated during synchronization by the Agentry Server,

which then transmits those instances to the Client. Object instances can also be created at run time on the Agentry Client via add transactions.

The object definition contains only a few attributes related to its identifying value, or “key property,” and the value displayed for the object during synchronization. The heart of an object definition lies in its properties. An object property defines a single piece of data for the parent object. The definition of an object property should always match the aspects and behaviors of the back end value it is created to store. This includes data type, data sizes, and the name.

The name of the object property should match the name of the back end value whenever possible. The Server matches the values returned by any back end steps to the properties within the object by matching the names. Any value returned from the back end whose identity does not match the name of a property in the object is discarded. Any object property that does not have a corresponding value in the return set from the back end is initialized to null upon object instantiation.

If it is not possible to match the property name with the back end value’s identifier, the value should be aliased in some manner within the return set. As an example, in SQL select statements a field can be aliased using the AS keyword, as in `Field1 AS Name1`. In Java the data structure containing return values can be named to match the properties.

The definition of an object should also include properties that may be needed on the Client side only. These values may be used for client-side processing or behaviors. In this case the values of these properties will be initialized to null when objects are instantiated during synchronization. They can be set via transactions on the client at run time.

Object instances are stored on the Client in one of two ways. First, a single instance of an object can exist as a property of another object. Second, and far more common, is to store objects in a collection property.

Objects are synchronized with the back end system via the module level definitions fetch and push. A fetch defines how the Agentry Server synchronizes data for a target object collection by referencing the step definitions to perform this task. A fetch is processed during synchronization between the Client and Server, with the results being the retrieval of new object instances for the target collection, replacement of existing objects within the collection, and the removal of objects from the collection. All of these determinations are based on the definition of the fetch and its child step usage definitions.

A push defines when it is necessary to push an object in real time from the back end system to the Agentry Client and how that object’s data is retrieved. Pushes are used only when a constant network connection can be maintained between the Client and Server. Like a fetch, a push targets an object collection property and will synchronize object instances within that collection. Objects can be added, replaced, or removed from the collection based on push processing. Differing from a fetch, pushes are run asynchronously by the Server and pushing objects to the Client when changes to the back end system are made.

Another definition type involved in synchronizing objects are object read steps. An object read step references a step definition run to retrieve data from a back end system to populate an object's properties. When the synchronization process is defined using the exchange data model, it is often the case that the fetch is defined to determine what objects do and do not need to be retrieved, and the read steps are then run to perform the actual data retrieval. This is a common practice but not a requirement of the development. The fetch can be defined to accomplish both tasks without involvement of the object read steps. Likewise, push processing can involve running the read steps of an object to retrieve the object data from the back end system.

Regardless of whether a push or fetch is used, and also whether or not read steps are involved in the process, object data synchronization includes both the object collection targeted by the fetch or push, as well as any collection properties that are descendants of the objects within the targeted collection. Any collection property that is not defined within the main object, but rather as a descendent of the main object, is termed a “nested collection.” Nested collections are objects whose data is considered a part of the parent and ancestor objects within the module data structure. Therefore the synchronization process for a collection includes any nested collections within it.

Object Properties Concepts and Considerations

An object property definition defines a single piece of data and its type. A property can also define minimum and maximum values, a default, or “special value” and other data-related behaviors. The specific data-related behaviors will vary depending on the data type of the property. The properties of an object give that object its characteristics. A property is the equivalent to a variable in other development platforms or languages.

When designing and developing an object's properties, the properties should represent all of the values to be retrieved from the back end system plus those that may be necessary for client-side processing. Examples of this latter group include state-related values or other data that will not be retrieved from or updated to the back end, but that may be needed on the Client.

Property Data Typing

When data typing your properties, the primary driving factor in the decision should be the data type of the back end value the property is to store. However, it is not a requirement that the data type of the object property match the data type of the back end value. The Agentry Server will always attempt to convert data retrieved from the back end system to the data type of the property definition. Therefore, if a value stored as one data type in the back end system, such as an integer, needs to be stored and used as a different data type on the Client, such as a string, it is completely valid to create a string property. Be aware, however, that when data types differ in this manner that the rules of safe data conversion still apply. For example the conversion of a string to an integer is not considered type safe and can result in undesirable behavior.

Object Key Property

When designing an object's properties you must always include a "key property" that uniquely identifies each instance of the object. Any property definition of almost any data type within the object can be designated as the key property. In practice the key property should be the value that uniquely identifies the business entity in the back end system. Examples include the work order number or customer ID, which would be values that would be defined to be the key property. The object definition contains an attribute that specifies which of its properties is the key property. Therefore the property must first be defined, and the object then edited to specify which property is the key property.

The key property is then how instances of the object will be uniquely identified by the Agentry Client and the Agentry Server. Only one object instance with a specific key property value can exist within a given collection property. Furthermore, the key property is also how the Server identifies the object instances within the collection for synchronization purposes. If a collection property is created for an object definition, that object must have a selected key property. The Agentry Editor will not allow the collection definition to be created for an object that does not have a key property.

Property Names and Data Syn chronization

The name attribute of the property should be set to the same name as it is identified by in the back end system. This plays an important part in synchronization, as the Server will look to how data values are identified in the back end system and match those values with properties of the same name. In addition to this requirement, following the back end names also makes future maintenance of the application easier.

If it is not possible to name a property to match the back end value, the identity of that value should be aliased in some manner during retrieval. For example the AS (SELECT Field1 AS Name1) keyword in SQL allows for this. In a Java back end the data returned can be renamed using the return data structure that stores the data for objects, as the members of that structure can be named to match the property definitions, with their values assigned to variables with different names from the back end system.

Object Data Structure Concepts

The object definition is the primary data definition for a module. All production data for the module is stored in instances of the object definitions created in the application project. The definition of the objects includes not only the data to be stored within each object type, but also the relationship between the objects within the module. These relationships are hierarchical in nature, with one object type the parent to another.

The Module Main Object

The beginning point of a module's data structure is the module's MainObject definition. This object is a part of all modules, added automatically by the Agentry Editor whenever a new

module definition is created. The intended purpose of the MainObject is to contain the top-level object collection property or properties, as well as other module-level data.

The Module's "Primary" Object

The primary object of the module is the one around which most or all of the functionality within the module revolves. This includes both client-side behavior and data synchronization. It is important to note that there is no setting or attribute within the application structure that indicates an object is the primary object. Rather, this is a logical term reflective of the design of the module, its objects, and its functionality and behaviors. Examples can include a work order object for a work management module, or a customer object for a customer relations module. When a new module is defined, the Editor will prompt you to create an object definition as well as a collection property within the MainObject. The object created at this point should be the module's primary object.

Object Collections - Parent-Child Objects

Other object definitions can and likely will exist within the module. These definitions are then associated with the primary object as child objects. This is accomplished by defining collection properties. An object is comprised of properties that define the data the object stores. These properties are of various data types, one of which is collection. A collection can store multiple instances of an object definition. When one object contains a collection of other objects, those objects are said to be child objects of the first.

The reason for defining a collection within a parent object is to indicate that those child objects are data that is a part of the parent, but that are also themselves business entities in need of encapsulation. Examples of this can include the two object definitions work order and job plan step. For a work management module, the work order object is likely to be the primary object. Instances of the work order object would then be stored in a collection property of the module main object. Instances of the job plan step object would then be stored in a collection property of the work order object. The job plan step objects within a given work order object would be those representing the steps for that work order's job plan. Each work order instance then has its own collection of job plan step objects specific to that work order.

This structure can continue to several levels deep within the application structure. As an example a job plan step may require certain parts are used. The job plan step object, then, could contain a collection property of a third object type created to encapsulate a part. In practice it is usually not necessary to create an object hierarchy within a module that is more than three or four levels deep. The term **nested collection** is commonly used to refer to any collection property that is not an immediate child of the module's main object. In the preceding examples, the work orders collection would be a top-level collection, and the job plan steps collection would be a nested collection.

When working with collections it is important to keep in mind that the collection is a property of the object. This means the object instances stored within a collection are data that make up that parent object, just as any other property data type. Also, a collection property itself is not an object. When working with other definition types that affect objects and/or collections, be sure to note this distinction. An attribute, argument, or definition that is expecting an object

will not accept an object collection property. Likewise an object collection property cannot be used where a single object instance is expected.

Object Data Synchronization: Fetches

The synchronization of object data is handled by fetch and push definitions, with the fetch being the primary definition for this purpose. A fetch defines how the Agentry Server synchronizes data for a target object collection. This object collection must be a top-level collection within the module. A fetch is made up of steps that retrieve the data for the collection from the back end system. These steps are grouped into three categories within the Fetch definition: Client Exchange Steps, Server Exchange Steps, and Removal Steps. A fetch may also include properties to store data captured from the user and validation rules for those property values. A fetch may also contain property and validation rule definitions, though this is a less common implementation option for fetches.

General Fetch Processing

During a transmit the Server processes all main fetches within the application. This is the primary distinction between main and non-main fetches. If a fetch is not a main fetch it will only be processed by the Server if the transmit step within the action on the client lists it as one to be processed. Regardless of whether or not a fetch is a main fetch, the behavior of a given fetch is the same when it is processed by the Agentry Server.

A part of fetch processing also includes a separate definition type, object read steps. The fetch targets an object collection within the module. The object definition can include object read steps. If the object type within the collection targeted by the fetch contains read steps, those steps are processed as the last part of the fetch processing. Note that read steps in any nested collection of the target collection of the fetch are not processed.

The order of processing the step usage definitions involved in fetch processing is:

1. Client Exchange Steps
2. Server Exchange Steps
3. Removal Steps
4. Object Read Steps

Each of these definition types references a back end step definition within the same module as the fetch. A fetch can contain multiple definitions of each of these types, with the order of execution defined within a given type defined by the developer.

The fetches of an application are the last synchronization definitions processed during transmit. For applications with multiple fetch definitions, the order in which each fetch is processed is undefined. Therefore each fetch definition should be defined to operate independently of any others within a given application.

Object Retrieval and Replacement Processing

When a fetch is processed during transmit, any step executed by a client exchange, server exchange, or object read step can return data to create new objects or replace existing objects. This process involves a step returning data from the back end system to the Agentry Server.

The values of this data are named or identified according to the logic within the step. For example, a SQL step containing a select statement will return a data set of records. Each column in the return set is identified according to either the name of the database table from which it was retrieved, or according to any column alias for the values contained in the SQL statement. The Agentry Server then processes this data according to the following procedure:

1. During the transmit the Client provides the Server with the key property and last update value of any object instances currently contained in the object collection targeted by the fetch. These values are stored in an object collection in the Server's memory that mirrors the collection on the Client. This occurs before the steps of the fetch are processed.
2. When a step within the fetch returns data, a value with the same name as the key property of the object is searched for in the return set. If such a value is not found then the data is discarded, as the Server cannot determine to which object the data belongs.
3. When the key property value is found, the Server compares this value with the key properties of any object instances it currently contains in the object collection.
 - a. If a match is found, any other data within the record is processed. Each value identified with the same name as property in the object is assigned to that matching property. If the object property contained a value prior to this processing, it is overwritten. This is object replacement processing.
 - b. If a match between the return set's key property and an existing object instance, a new object is instantiated by the Server and stored in the collection. The remaining values in within the record is processed, with each value identified with the same name as a property in the object assigned to that matching property. Any properties within the object that do not have a value in the return set are initialized to null. Any values in the return set that do not have a match in the object properties are discarded.
4. This process repeats for each step executed as a part of fetch processing that returns data. When the fetch step usage definitions and object read steps have all been executed, the Server sends down any new or replaced objects to be stored in the object collection on the Client.

Object Removal Processing

In addition to retrieving object data, the fetch processing can also result in the removal objects from the Client. When a removal step returns a value identified as the key property for the object type being synchronized, the Server will send this value to the Client with the indication that it should be removed from the collection property. The removal step should normally only ever return the key property value, as any other data returned is not used by the Server and therefore unnecessary.

When an object is deleted from the Client it is important to keep in mind that the objects stored in any nested collections are also removed. Remember that these child objects are data for the parent object and, just as any other property would, they are removed with the parent object.

The order of processing during a transmit is such that, if an object is to be removed as the result of fetch processing, any data captured on the client in transaction definitions targeting the fetch will have already been processed. No data captured on the client is lost as a result of a

fetch removing the object, as the transaction will have already been processed and, presumably, updated any captured values to the back end system.

Object Read Step Concepts

The object read step definition is a child to the object definition type. An object read step references a step definition within the same module. Its purpose is to retrieve data for instances of the object from the back end system. The steps are processed by the Agentry Server during a transmit. The step being referenced can be executed once per transmit or iteratively.

Multiple object read steps can be run to retrieve the data for an object. During synchronization the Agentry Server will create instances of the object after the first set of return data that contains values for the object key property. Subsequent steps that return data for the object must also include the key property to indicate which object the data belongs with. These values will then be used to set the property values of the object. Any object read step can return any property value for the parent object type. A single step can return all of the data or multiple steps can be run to retrieve all of the data. This processing is defined by the developer and the nature of back end system and how data can be retrieved will dictate the proper way to retrieve the data.

Object Read Steps and Back End Steps

The two key items to keep in mind when defining an object read step are that, first, the step definition being executed is separate from the object read step. The step being executed is a module level definition containing the processing logic desired. The step definition must be defined first, and then the object read step can be defined to run it. Depending on the step type this can be a SQL statement, Java logic, or HTTP-XML calls. The object read step references the step definition and specifies when and why it should be run. It does not define the actual processing or the back end system to use. This separation of the logic and the context is an intentional part of the overall architecture that allows for multiple steps defined for different back end systems to be used an executed to synchronize data for a single object type.

The step executed by an object read step has access to certain data about the object. The specific values that are in scope depends on how the object read step has been defined to be executed. To access the in-scope values the Syclo Data Markup Language (SDML) is used.

Any step executed as an object read step must be defined to return not only the data for the object properties in need of values, but must also return the value of the object key property. The key property is used by the Agentry Server to determine which object instance should be assigned to values in the return set. If a read step returns a key property that does not match an existing object a new object is instantiated and the other values in the return set are assigned to the new instance's properties.

The purpose of an object read step is to return data for properties of the object. This includes object collection properties. While a collection contains object instances, from the context of the parent object the collection is simply another property containing data that is a part of the overall object instance. Therefore an object read step can return data to create instances of the object type stored in the child collection property. An object read step must be defined to read

the data into that child collection. The data returned by the step must include the key property of the parent object and the key property of the object instances to be added or updated within the collection property.

As an example of reading data into a child collection property, consider a work order object that contains a collection of job plan steps. An object read step can be defined in the work order object that retrieves data for object instances within the job plan steps collection. The step executed to retrieve this data must return the work order objects key property and the job plan object's key property. This data is needed by the Server to determine, first, which work order object contains the job plan step object and, second, which job plan object instance the data should be assigned to. Just as with any object read step, if a job plan object key property value is returned that does not match an existing object instance, a new job plan step object is instantiated and added to the collection property of the work order object.

Object Read Steps and Fetch Processing

Object read steps are run as a part of the synchronization process for objects and object collection properties. During Client-Server transmission, fetches are processed as the first part of object data synchronization. If the object type that makes up the collection targeted by the fetch contains read steps, those steps are processed after the fetch. In a common application architecture, the fetch will synchronize the exchange data and the object read steps will use that data to determine which object instances to retrieve from the back end system.

When object read steps are run as a part of fetch processing the steps executed are, in most cases, defined to retrieve data for all instances of the object in a single execution. The object read step is defined to run one time in this case, rather than to iterate over the object collection being synchronized. Note that this is the most common way object synchronization is defined, but is not a requirement. There are situations in which a portion of the object synchronization must be performed one object instance at a time, or iteratively. A ready example of such a situation is when file transfer, or "attached documents" functionality is being implemented.

Steps run as object read steps during fetch processing that are also defined to iterate over the object collection have access to the key property of each object instance in the collection and the last update value for each object. These values are sent by the Client to the Server at the beginning of the fetch processing during transmission.

Object Read Steps and Transaction Processing

Read steps may also be run after a transaction has been processed that targets an instance of the object. This processing only occurs when a server data state step or server update step within the transaction has been defined to replace the client object after transaction processing. In this situation the object read step is run for the object instance targeted by the transaction. Therefore the step executed should be defined to retrieve data for a single object. Read steps intended to retrieve data for object collection properties can still be defined to retrieve all data for objects within the collection.

When an object read step is run as a part of transaction processing, with the intent of replacing the object on the Client, the in-scope values for the step include the object's key property, the object's last update value.

Object Read Step Execution

The execution of a read step is controlled by the attribute Run. The Run attribute specifies how often to run the step in relation to the object instances currently being processed and in scope. This execution can be either once or iteratively. Running the step once means the step being processed is expected to return all of the needed data for all of the objects in a single execution. The Server is capable of processing such return sets to create or update multiple object instances.

For iterative processing there are two options. First, the step can run one time for each instance of the parent object currently in scope. For example, when processing a collection of work order objects, a read step within the work order object can be defined to run once for each work order object instance the Server currently contains.

The second option for iterative processing is to execute the step once for each object instance in a collection property of the parent object. This iteration then includes the parent object as well as the objects in the collection property. So if the work order object contains a collection property of job plan step objects, an object read step can be run once for each object within the job plan steps collection of each work order object.

In the case of iterative processing of the object read steps it is assumed a previous read step or one of the fetch steps has returned the data needed to create the objects. Read steps defined to be executed iteratively then run once for each of these object instances to continue the data synchronization process. If an object instance does not exist when the iterative read step is to be processed it will not be executed by the Server as there are no objects to iterate over.

Object Read Step Development Considerations

When designing and developing object read steps, the following items will factor into how the steps being executed should be defined as well as the object read steps themselves.

- The overall context of the read step execution, i.e. is it being executed for fetch, push, or transaction processing? Will the object read steps as a whole be executed for more than one of these?
- The data to be retrieved by the step and where it is intended to be used. Within the parent object of the read step, in a collection property of the object, in a descendent collection property?
- The overall requirements of the object data retrieval process, as dictated by the back end system. The order in which object read steps are executed is always an important consideration.
- The type of data or objects being retrieved by the step. File transfer functionality, for example, will have different effects on the design and development of the step than the retrieval of some other data types.

- The overall requirements of the back end system and system with which data is being synchronized.

One of the main aspects of the object read step definition to keep in mind during the design and development of an application is that the object read step is always run as a part of the processing of other synchronization definition types. There is no point in the synchronization process in which the object read steps are run by themselves. They are, rather, executed after the processing of a fetch, push, or transaction. Because of this overriding aspect of the read step definition, detailed discussions of the development of a read step are deferred to the discussions of the overall processes for these other definition types.

Here information is limited to the data available to the steps being executed as object read steps under various circumstances.

Read Step In-Scope Values: Fetch and Push Processing

The data values available to a read step run as a part of fetch or push processing is dependent in large part on how the step is being executed, that is, the setting of its Run attribute. Therefore the following table lists the data available to the read step organized according to the different settings of this attribute:

Run Attribute Setting	Available Data Values
Run One Time	Any SDML local data tags created by the fetch or previously executed read steps.
Run Once Per Object	Any SDML local data tags created by the fetch or previously executed read steps. The key property of the current object instance. The last update value of the current object instance.
Run Once Per Collection Object	Any SDML local data tags created by the fetch or previously executed read steps. The key property of the current object instance. The key property of the current collection object instance. The last update value of the current collection object instance.

Read Step In-Scope Values: Transaction Processing

The data values available to a read step run as a part of transaction object replacement is dependent in large part on how the step is being executed, that is, the setting of its Run attribute. Therefore the following table lists the data available to the read step organized according to the different settings of this attribute:

Run Attribute Setting	Available Data Values
Run One Time	Any SDML local data tags created by previously executed read steps. The key property of the object instance being replaced. The last update value of the object being replaced.
Run Once Per Object	Any SDML local data tags created by the fetch or previously executed read steps. The key property of the object instance being replaced. The last update value of the object instance being replaced.
Run Once Per Collection Object	Any SDML local data tags created by the fetch or previously executed read steps. The key property of the object instance being replaced. The key property of the current collection object instance. The last update value of the current collection object instance.

Fetch Development Using the Exchange Data Model

The design and development of fetch to synchronize an object collection should always be based on the exchange data model. The child definitions of a fetch, as well as object read steps and the object itself, are organized and architected with the intent of using this model. The main tasks to accomplish when developing a fetch to use the exchange data model are:

1. Create or configure the exchange components in the back end system to be used both during the synchronization process, as well as to track changes to the back end system between client transmits. These components should track the unique identifier values for each business entity, the date and time of the change, and the nature of the change. This last includes tracing the addition of new instances, modification of existing data, or the removal or other modification to be treated as a removal by the Agentry application.
2. Define the fetch to determine and record in the exchange components what objects the Client contains at the beginning of the synchronization process. This includes the date and time when each object was last retrieved from the back end system.
3. Define the fetch to use the exchange and tracking components in the back end system to determine what object-related data has been added or modified since the last time the Client synchronized.
4. Define the fetch to use the exchange data generated in the previous steps to determine the differences between the Client objects and the back end data. Retrieve only those objects that need to be added or replaced, and define the fetch removal steps to retrieve the key property values of those objects to be deleted from the Client.

Back End Exchange Data and Tracking Components

The design and creation or configuration of the back end components used in the exchange data model have certain general requirements. These requirements hold true regardless of the contents of the object data being synchronized.

Beginning with the tracking components, the purpose of these items is to track changes of interest to the object data in the back end system that occur between transmits from the Client. These components should track changes to the object data in the back end system. Depending on the nature of the back end system, changes to track can include adding new objects, modifying the data of an existing object to be reflected on the Client, or the deletion or modification of an object that should result in that object being removed from the Client.

Note: Regarding the removal of objects, this may occur within the Agentry application as the result of various types of changes to the data beyond the removal of the object from the back end system. Other changes to the object can dictate the end user should no longer have the object on the Client. Examples include the reassignment of a work order, a change in the objects status, such as inactive or deprecated, or similar modifications. This should be kept in mind when implementing both the back end components as well as when defining the removal steps of a fetch.

When one of these changes occurs, the information recorded must include:

- The value or values that uniquely identify the modified object in both the back end system and the Agentry application.
- The date and time, as provided by the back end system, when the change occurred.
- The nature of the change, that is, is it a new object, a modification, or a removal.

The Object Last Update Value

The object definition type is capable of storing a date and time value called **lastUpdate**. This data value is separate and in addition to the defined properties of the object. The date and time value it stores must be returned with the object data during synchronization and aliased as “lastUpdate.” The proper source for this value is the current date and time of the back end system when the object data is retrieved.

The purpose of the lastUpdate value is to store the date and time of the back end system when the object was retrieved. Each object instance has its own lastUpdate value. This value is stored with the object instance on the Client. It is accessible during subsequent synchronizations via the SDML tag <<lastUpdate>> and is intended to be used to determine whether changes have occurred on the back end system for the object since it was retrieved for the Client. It should be compared with the date and time value recorded by the back end tracking components for the same object.

This value is only accessible during synchronization and, while stored with the object instance on the Client, it is not exposed on the Client. It is not a property value and cannot be displayed to the user nor modified as a result of any client-side processing.

Client Exchange Steps in the Exchange Data Model

A fetch client exchange step defines how information about the target collection is processed by the Agentry Server. This definition references a step definition within the same module. This step has access to information about the target collection, as well as to any data captured in fetch properties. A client exchange step can be defined to execute once or iteratively, and can return data for an object collection. A fetch can contain multiple client exchange step definitions, which are processed by the Server in a defined order.

When a client exchange step is executed iteratively, its iterations are based on the object instances sent to the Server by the Client. The client exchange step then has access to the object's key property value and the object's lastUpdate value. If fetch properties have been defined, the client exchange steps of that fetch will also have access to all of these property values, regardless of how the client exchange step has been defined to execute.

The intended purpose of client exchange steps is to update the back end exchange data components with information about what object instances currently exist on the Client in the target collection of the fetch. This information should include the key property of each object and the lastUpdate value. Along with these values the client exchange step should also provide information about the user to whom the object belongs (SDML tag: <<user.agentryID>>). Finally it is a recommended practice that the Agentry Server instance also be uniquely identified (SDML tag: <<server.serialNumber>>).

Another task commonly handled by the client exchange steps is to clear out the exchange data for the current user from the previous synchronization. The step defined to accomplish this task should delete this data based on the user ID and, in most cases, also the Server's serial number. This client exchange step is run as the first step for the fetch. It is then followed by the client exchange step defined to provide the exchange data about the current objects on the Client.

When the client exchange steps of a fetch have completed processing during synchronization, the back end exchange data components should contain the information about what object instances currently exist on the Client and the date and time each was last retrieved from the back end system.

Server Exchange Steps in the Exchange Data Model

A fetch server exchange step defines how information about the back end system's data is processed. This definition references a step definition within the same module. This step has access to information about the target collection, as well as to any data captured in fetch properties. A server exchange step can be defined to execute once or iteratively, and can return data for an object collection.

Server exchange steps are always processed after client exchange steps. The intended purpose of a server exchange step is to determine what changes have occurred in the back end system to the object data for the user and to then either mark the objects in the exchange data components as those in need of retrieval, or to perform the actual retrieval of the object data. Which

behavior is defined is dependent on the type of back end system and its capabilities and behaviors, as well as the overall needs of the mobile application being developed.

Changes or differences between the Client objects and the back end data are found by comparing the information about the current Client-side object instances provided by the back end exchange data component with the information captured in the back end tracking components. The tracking components will contain the date and time when any object data has changed on the back end system, along with the unique identifier for that object. The exchange component will contain the date and time when the object instances were retrieved. The server exchange steps then should contain the logic to compare the information in the tracking components with the information in the exchange components. When an object is found to have been changed in the back end system more recently than it was downloaded to the Client, that object is one in need of replacement on the Client. Any new objects created in the back end system will not have a corresponding item in the exchange component. Such objects should also be treated as those in need of retrieval for the Client.

When differences are found, the functionality can take one of two directions. First, any changes for existing objects result in the object record in the exchange component being flagged or marked in some manner indicating the object should be replaced. Any new objects in the back end system not found in the exchange data components are added to the exchange component. This information includes the unique identifier, or key property of the object, the user ID, the server ID, the current date and time, and finally the same flag or indicator that the object should be retrieved. In this scenario, the actual object data is not yet retrieved. This logic is most common with SQL systems. In this scenario the data retrieval is then left to either additional server exchange steps, or, more commonly is handled by object read steps.

The second option is define the steps used as server exchange steps to make the determination about which objects need to be retrieved and to retrieve that data at the same time. In this situation, the server exchange steps do not update the exchange component with the information about the new changes in the back end. Instead, the data in the back end exchange component as provided by the client exchange steps is used only for comparison purposes with the tracking components. This logic is most common with Java and Web Service (HTTP-XML) systems.

Fetch Removal Steps in the Exchange Data Model

A fetch removal step is defined to determine which objects should be removed from the collection targeted by the parent fetch. A removal step references a step definition within the same module. This step has access to information about the target collection, as well as to any data captured in fetch properties. The step referenced by a removal step definition is expected to return the key property of any object(s) that should be deleted from the target collection on the Agentry Client.

The removal steps defined within the exchange data model should use the exchange data components to determine what objects the client has. It should interrogate the corresponding data in the back end system to determine if any of those objects should be removed from the Client. The removal of a Client object can occur in many situations, only one of which is the

actual deletion of the data in the back end system. In point of fact, this is usually the least likely situation, as most enterprise systems do not remove data once it has been added.

The removal steps can use the data in the back end exchange data component to look for object data for a user in need of removal. As an example, if a user currently has work order 123 on his Client and that same work order has been reassigned in the back end system to a different user, the removal step can return the key property of that work order object so that it is removed.

Object Read Steps in the Exchange Data Model

An object read step references a step definition within the same module. Its purpose is to retrieve data for instances of the object from the back end system. The steps are processed by the Agentry Server during a transmit. The step being referenced can be executed once per transmit or iteratively.

Object read steps may be run at different times during a transmit. One of these times is as a part of fetch processing. Object read steps are not required to retrieve data during fetch processing, as both client exchange and server exchange steps can accomplish this task. However, it is common to use object read steps for this purpose. In the exchange data model, the fetch is processed to determine what changes have been made to the back end data and the object read steps are then run to perform the actual retrieval. One of the primary reasons for this division is simple organization of the project.

When used in the exchange data model the steps executed as object read steps should contain logic to use the exchange data generated by the fetch processing. The object read steps should retrieve only that object data related to objects in need of retrieval or replacement on the Client. The processing of the object read steps and the logic executed should be the final culmination of all exchange data model processing performed to this point. The read steps take advantage of the information generated and gathered by the child definitions for the fetch, retrieving only the objects needed by the Client and excluding the retrieval of any unchanged objects.

Agentry User Interface Definitions Overview

The primary purpose of the architecture of the client-side user interface definitions within Agentry is to support multiple client device platforms from a single application. This architecture, then, includes the separation of the application's business logic from the user interface. This separation then requires the specific structure of the user interface definitions within the Agentry application project.

The primary interface definition is the Screen Set. The screen set is a module-level definition. A screen set definition defines the Agentry Client's user interface. The screen set defines the definition type to be displayed, which can be an object, transaction, or fetch within the same module. The properties of this definition type can then be displayed by the screen definitions within the screen set. Screen sets contain the child definitions screen and platform. The screen set is the definition referenced by other definitions for display and are universal to all supported platforms within the application project.

The platform definition is one of two child definitions to the screen set. A platform definition defines how a screen set's screens will appear on a specific device type. A platform is defined to use one or more screens within the same parent screen set. There are different platform types, each corresponding to a different type of client device. The platform affects the placement of buttons and the form factor of the screens it uses.

The screen definition is the second child definition to the screen set. A screen definition defines how the property values in the definition being displayed are presented to the user on the Agentry Client. This includes which values are displayed. A screen also defines, via its child control definitions, how a user can interact with the Client. There are two types of screen definitions: list screens, to display object collections; and detail screens, to display a single instance of an object, transaction, or fetch definition.

Screen Sets, Platforms, and Screens at Run Time

The overall structure of the screen set definition is intended to support multiple client device platforms from a single application project. Within the screen set there exists one or more platform definitions. Each platform definition matches a client device type within the implementation environment. A given platform is defined to use one or more screens within the same screen set. Depending on the types of client devices, there may or may not be overlap in the list of used screens among the platforms; i.e., it is possible for the same screen to be used by more than one platform. Alternately, a given screen or screens may be used by only one platform within the screen set.

When new or edited screen sets are published to the server, they are then downloaded to the clients at run time. When this occurs, the client provides the server with information about the client device upon which it is running. The server then interrogates the screen set definition, looking for the platform definition within it for that client device. When found, the screens used by that platform are then those downloaded by the client for that screen set. No other screens within the screen set are received by that client. A second client running on a different device type will receive the same screen set, but with different screens. These screens would be the ones used by the platform definition for this second device type.

The screen set, then, is the definition referenced by other definitions for display. Typically this is an action step, such as a navigation or transaction step. Actions and their steps are platform independent, meaning all clients receive the same action definitions regardless of the client device type. The action steps then dictate the screen set to be displayed. When a screen set is displayed on the client, the screens it contains are always the ones matching the device type, as the screen set on a given client will contain only those screens for that client device.

How the screens of a screen set are displayed depends on the definition type the screen set is defined to display. When a screen set displays an object, its screens are displayed in a tab control. Each screen definition is represented by a tab. Selecting a tab displays the corresponding screen.

When the screen set is displaying a transaction or fetch, the screens are displayed in a wizard format. This results in each screen being displayed one at a time, with wizard buttons (back,

next, cancel, finish, etc.) displayed at the bottom. The user navigates through the wizard using these buttons, entering data in each screen's fields.

Screen Types: List Screen

There are two types of screen definitions that can be added to a screen set: list screens and detail screens. A list screen definition displays an object collection property on the Agentry Client. Object instances from the collection are displayed as rows in the list. A list screen contains the child definitions column and button. A column is defined to display the property value for each object instance in the collection. Buttons are defined to execute actions related to the object instances. List screens include definable behaviors related to filtering, scanning, and sorting, as well as other screen enhancements for displaying data stored in the object instances of the target collection property.

The list screen definition is typically used for a basic presentation of an object collection property. Each object instance is displayed in a list control, which is the main feature of the list screen. This screen type can only be used to display object collection properties. The columns of a list screen are defined for the properties of the object type in the collection being displayed in the list control.

The list screen can be defined to allow or prevent users from resorting the list of objects by clicking a column header within the list control. The default is to allow resorting. Disabling this functionality will prevent the user from resorting the list of objects. This is often used when the objects represent some prescribed order, such as safety plan procedures. In this case it is generally considered good form to select a fixed sort property from the object collection being displayed. This is also defined in the list screen.

The columns themselves are defined to either be included or excluded from those values upon which the list can be filtered. The default for a column is to be included. If it is desired to prevent the user from filtering the list screen on certain object values, the columns for those values can be defined to be excluded from the filter values. As a separate attribute in the column is whether or not the column values should be included in scan filtering. Scan filtering is the behavior where a user can scan a barcode value and the currently displayed list is then automatically filtered to only those items that match the scanned value. The column definition includes the Scanner Filter attribute that specifies whether or not the column value should be used to filter the list based on a scanned value. This behavior only applies when the parent list screen is used by a platform for scanning, and only when the client device is equipped with a barcode scanner.

List columns can also display the values for each object as a hyperlink. When this feature is enabled, each cell in the column is displayed as a hyperlink. When the hyperlink is selected an action is executed. Part of the hyperlink functionality is to define the action to execute and the object instance to be targeted.

In addition, each list column definition can be enabled or disabled based on a rule, with disabled columns hidden from the user. Columns can also be formatted using the Format attribute. Finally, the default width of the column can be specified. Related to this is a behavior

defined in the parent list screen. It is possible to prevent users from resizing the columns by defining the list screen to disable this featured.

Screen Types: Detail Screens

A detail screen definition displays a single instance of an object, transaction, or fetch on the Agentry Client. The properties of the definition instance are displayed in fields, a child definition to the detail screen. Definable behaviors of a detail screen are predominantly controlled by the screen's child field and button definitions, which can include read-only or read-write values within the fields, as well as numerous field type behaviors. Detail screens for transactions and fetches do not have the child definition button.

The overall behavior of a detail screen is dependent in large part on the fields it contains. A detail screen field defines a field control for display on the parent screen. The field displays data to the user and, when displaying a transaction or a fetch, can capture data from the user. A field can be defined to have one of several edit types that will affect both the appearance and behavior of the field on the screen, especially when capturing data.

The field edit types vary from basic string fields to more robust fields including several different list types, a calendar control, date and time pickers, and several others. The proper field edit type depends on, first, the data type of the value the field is displaying, and, second, the desired method in which users should enter data. When fields are displayed on detail screens displaying object instances, data entry is typically not a part of the design as the fields are read-only when displaying object properties. In this case, the field's edit can be changed, but typically it is left set to default. When the field's edit type is set to default, the field's edit type at run time matches the data type of the property being displayed.

For detail screens displaying transactions or fetches, also known as wizard screens, the field edit type should always be considered carefully as these fields will be used to capture data from the users. The method of data entry provided to the user can have a significant impact on the applications usability, as well as the accuracy and validity of the data captured.

Fields for a detail screen can be hidden or displayed based on conditions checked by rules. Likewise fields can be enabled or disabled conditionally. the label for fields can be a simply text value, a hyperlink that executes a defined action, or can be committed entirely, leaving just the field itself with no label. Fields can be defined as read-only. This attribute affects fields on wizard screens (for transactions or fetches) or on fields for object screens when those fields do not target any object property.

The field includes several attributes related to its positioning on the screen, the viewable size of the field, and the amount of space within this size dedicated to the field's label. Fields can also have a shortcut key associated with them, which will set the focus to the field when selected.

Detail screens are defined with a certain number of columns and rows. These are for the purpose of layout. A given fields position on the screen is defined by specifying the column and row in which the field's upper-left corner should reside. Likewise, the fields width and height are also defined in terms of the number of columns wide and number of rows high. A

detail screen is created with a default number of rows and columns which can be edited by the developer. Note that changing the number of columns or rows for a detail screen does not change the size of the screen. Rather, it results in a larger or smaller number of “pieces” to that detail screen for the purposes of field layout.

When changing the number of columns and rows for a detail screen, it is recommended this be done before fields are added to the screen. If these values are changed after fields have been added, the layout of those fields will be affected. If fields are positioned on rows 1-10, and the detail screen is then edited to contain only 8 rows, the fields on rows 9 and 10 will no longer be displayed and must be repositioned some where in the first eight rows.

Button Definition for All Screens

The button definition is common to both list and detail screens. A screen button defines a button control to be displayed on a Client screen. The button may be displayed as a standard button control, a tool bar button, a menu or menu item, or as a separator. A button is defined to execute an action when clicked or tapped, unless defined as a menu or separator. When executing an action the button also defines the target object instance provided to the action for processing.

When developing mobile applications screen space is at a premium for many of the client device types in common use. For this reason, the button definition has multiple types. The type action button creates a traditional button control that when clicked executes a defined action. The exception to this is when the selected “action” is Popup Menu, which is one of the available options in the Action attribute of a button. When this is selected, the button will not execute an action. Rather, it is displayed with the defined label, plus an arrow pointing up. When selected on the client, a popup menu is displayed. Additional action buttons on the same screen can be defined to be displayed on this popup menu.

Application Menu is a button type that adds a menu item to the client’s menu bar, in the menu Actions. This menu is hidden unless at least one Application Menu button has been defined for the current screen. This type of button creates a menu item within this menu that, when selected, executes the defined action. This button type does not support image icons or the style attributes.

Toolbar Button is a button type normally used on Pocket PC devices, or devices with this form factor. It creates a button with no label and only an icon. The button resides below the screen in the toolbar of the client. When clicked it executes the defined action.

Separator is a button type that does not create any button or menu control. Rather, it is used to help organize buttons on the screen. When a separator is defined and the button is not defined to be displayed in a popup menu, additional space is placed at the position of the Separator button definition. When the separator button is placed in a popup menu, a menu separator is drawn at the position of the Separator button.

Buttons for List Screens

The button definition for list screens is defined to target, by default, the currently selected object or objects in the screen’s list control. Depending on the action being executed this may

or may not be the proper selection. The type of object targeted by the object must match the object type for which the action has been defined. The exception to this is when the action is not defined for any object type (attribute **For Object:** -- None --).

Typically either the selected object or the parent to the collection being displayed by the list screen are the two items selected for the button target. Buttons that execute actions to navigate to another screen set, and that execute actions to instantiate an Edit or delete transaction are normally defined to target the currently selected object. Buttons that execute actions to instantiate Add transactions, or actions such as Transmit or CloseThisScreenSet should be defined to target the parent object of the collection being display in the list screen.

Buttons for Detail Screens

The button definition for detail screens is defined to target, by default, the object currently being displayed in the detail screen. In traditional development work within Agentry this was not often changed. However, in more contemporary applications developed using later versions of Agentry, the selection of a different button target has increased in frequency. This is due to many of the newer field edit types added to Agentry. Many of these fields display lists of objects, or a selection from a list. These fields then support the selection of a target for a button from that field's current selection. This can be selected using the target browser within the Agentry Editor.

Client User Interface Considerations and Guidance

When developing the user interface for a mobile application in Agentry, the first consideration should be given to the screen flow, i.e. how the user should navigate through the information presented in the screens. In general it is a good starting point to look to the data within the module. In any real-world application there is likely to be multiple collection properties with a structure or hierarchy of their own. For example, Customers may contain Orders, which in turn contain Products. Likewise, Work Orders may contain Job Plan Steps.

When designing the screen flow, then, it can be useful to start with a basic drill-down approach. First, present the user with a list of the top level collection in the module. Then, allow them to select a object in this list to view details about that object. These details can include the property values of the selected object displayed in fields, as well as the collection properties it may contain, displayed in their own lists. If further nesting of object collections exist, this can be repeated for level of data within the module's data structure.

Note that this is a beginning point within the design. This structure need not be a part of the final implementation, and in fact may never be implemented exactly in this manner at all during development. However, it can be used as the foundation for the final UI design and implementation.

Once the basic drill down structure has been designed it should be further refined to match the needs of the application, and to reduce the amount of interaction required by the user, that is, to reduce the number of clicks required to get to the information or functionality needed.

Next, the portion of the user interface for transactions should be considered. If users can add instances of an object, consider the best point or points within the UI flow to expose this ability. Similarly, edits to the objects, and also deletes, should be exposed at points where it makes sense to the users. Again, when making these determinations, begin with the basics. If allowing users to add a given object type, expose this action on the screen where this collection is listed. Also, deleting objects is functionality typically exposed in the list for the collection.

Edits can be expose in lists as well, but may also be exposed in detail screens for the object. In many applications there are numerous edit transactions for the same object, with the different edits affecting different property values within the object definition. If these values are displayed in detail screens for the object grouped together in a manner similar to how they are organized in the different transactions, it makes sense to expose those transactions in those detail screens where the values the transaction affects are displayed.

Security Related Development Overview

When developing a mobile application security is always an important aspect to the process. Using the Agentry archetype many of the security features are implemented for the application as a part of the development of the Agentry application project. Information is provided here on the security features and development options available and how they are implemented in the application project using the Agentry Editor.

Client-Side Data Encryption

Any Agentry Client can support the encryption of all data stored locally on the client device. When implemented, production data retrieved from the back end system, as well as the application data (or business logic) of the application is stored encrypted. Subsequent information is provided on how to implement this functionality for your mobile application. This may be defined within the application project while it is being initially developed, or it may be a change made to an existing application.

Securing File Attachments From iTunes on Agentry Client for iOS

Depending on where file attachments are stored on an iOS client device, they may be accessible through iTunes when the client device is connected to that application. Information is provided on how to modify or define the External Data properties of the Agentry application project so that files stored on the client device are not accessible to iTunes.

User Lockout After Failed Login

A standard part of any IT department's security policies is a specification on the maximum number of failed login attempts can be made by a user before restricting their access to the system in some way. This behavior is supported in Agentry via the use of security settings within the Application definition of the Agentry application project. Included in this functionality is the ability to define the maximum number of login attempts allowed by the user, and the corresponding lockout action to take when this maximum is met. As a part of the definable behaviors it is possible to require the user to perform a full transmit before being

allowed to access the Agentry Client, as well as optionally removing some or all of the data stored on the client device by the Agentry Client.

Transaction Authentication

As a part of the workflow of the client application it is possible to require the user to re-enter their user credentials before a transaction is applied. When implemented the user will be required to enter their user ID and password, which is validated against the locally stored credentials for the user, before the transaction is applied and saved on the client. Additional information may be captured from the user as a part of this process. This data is both stored locally and is also available for update to the back end system as a part of the transaction processing during transmit.

Defining Client-Side Data Encryption

This procedure describes the process of defining the Application definition within the Agentry application project such that Agentry Clients will encrypt all data stored on the client device.

1. Open the Agentry application project for you mobile application in the Agentry Editor.
2. View the Application definition and select the Application Security tab in the Properties view.
3. Set the attribute **Client Database will be encrypted** to true. Save the change.
4. Publish or deploy the project to the Agentry Server.

Data stored on the Agentry Clients will be encrypted.

Securing Attachments on iOS Client Devices

Prerequisites

It is assumed that attached documents functionality has been defined for the mobile application. This procedure does not describe how to define or implement this functionality, but only how to define the mobile application for iOS client devices to prevent attached documents from being accessible via iTunes.

Task

This procedure describes how to define the mobile application to secure attached documents from iTunes access on iOS client devices. This process involves setting the iOS Base Path attribute of external data properties within the application to a value other than Documents, typically the Application Support option.

This procedure needs to be repeated for each external data property within the mobile application project.

1. Using the Agentry Editor navigate to the external data property definition within the mobile application project. Select the File Locations tab within the Properties view of the definition.
2. Within the iOS section of attributes on this tab, set the Base Path attribute to the option “Application Support.”
3. Save the changes made and repeat this process for any other external data properties within the application project.
4. After all external data properties have been modified, publish the application to the Agentry Server and test the behavior. Be sure to connect the iOS device to an iTunes application and verify the attachments for the mobile application are no longer accessible through iTunes.
5. When ready to make this behavior available to the mobile users, publish or deploy the application to the Agentry Server in the production environment. This new behavior will take affect when mobile users perform their next synchronization.

After this procedure is complete the files stored on the iOS device by the Agentry Client will no longer be accessible to the iTunes application.

Configuring User Lockout for Failed Login Attempts

Prerequisites

The following items must be addressed prior to performing this procedure:

- Determine the desired number of maximum login attempts before locking out a user.
- Determine the proper response by the client when locking out a user. Review the information provided in the *Agentry Language Reference*, specifically the section “Application Definition,” in the subsection “Application Security Attributes.”

Task

This procedure describes how to configure the user lockout behavior on the mobile application, which occurs after the defined number of failed login attempts by the mobile user. A part of this configuration is the resulting behavior, as set by the “lockout level”, when a user is to be locked out.

These settings should be configured to match the security requirements of the implementation environment. Possible lockout behaviors range from simply requiring the user to perform a successful login and full transmit with the Agentry Server before being allowed to proceed; removing all module-level production data (including object instances and pending transactions) and requiring a full login and transmit; or completely resetting the Agentry Client executable, removing all data stored by the application, and requiring a full transmit and synchronization before being allowed access to the application.

Defining this behavior requires the modification of the Application Security attributes found in the Application definition, followed by publishing the changes to the Agentry Server, with a

subsequent transmit by each Agentry Client to update the mobile application with the new settings.

1. Open the Agentry application project in the Agentry Editor. View the Application definition and select the Application Security tab in the Properties view.
2. Begin by setting the maximum number of login attempts to allow by setting the attribute Login Attempts to the desired value.
3. Select the desired lockout behavior by selecting the appropriate option for the Lockout Level attribute.

For details on the Lockout Level options, see the “Application Definition” section in the “Agentry Language Reference.” Review the information for the Lockout Level attribute found in the “Application Security Attributes” subsection.

4. Save the changes made to the Application definition. Publish the application to the Agentry Server used for testing and verify the desired behavior. Publish or deploy the application the Agentry Server in the production environment when you are ready for the mobile users to receive these changes.

The desired lockout behavior for mobile users reaching the maximum number of failed login attempts has been defined. The behavior will be exhibited on the Agentry Client for mobile users in the production environment once published or deployed to that environment.

Transaction Authentication/Electronic Signature Support

The purpose of transaction authentication is to validate that an authorized user is the one that entered the information captured by the transaction being authenticated. This functionality is also implemented to support electronic signatures in environments where audit trails are a requirement.

Transaction authentication is defined within the transaction itself and can be set as always required or conditionally required based on the Boolean return value of a rule definition. During transaction authentication on the Client the user is required to enter the user ID and password with which they logged into the device. Additional information may also be captured as a part of the authentication process where needed.

Any transaction defined within the application project can also be defined to include transaction authentication. Data captured during the authentication process is accessible during the synchronization of the transaction during transmit. There are different definitions involved in the transaction authentication processing, including:

- Object definition to store authentication data on the Client
- Screen set definition to display and capture data during the authentication processing on the Client
- Step definition to process the authentication information during transmit and data synchronization
- Rule definition (optional) to determine when the user should be required to authenticate

Authentication Object

The object definition displayed in the screen set during transaction authentication, termed the “authentication object,” should contain properties for each of the pieces of information to be captured from users during the authentication processing on the Client. This typically includes both the user ID and password values. It can also include additional information from the users as may be required for the specific environment. This data is accessible to the step definitions of the transaction during transmit.

Authentication Screen Set

The screen set definition displaying the authentication object during transaction authentication, termed the “authentication screen set,” should be defined to display the object definition. Unlike other object screen sets, however, when displaying the authentication object the screen set is displayed as a wizard screen set. It should contain only detail screens and the fields of those screens are defined to capture the desired authentication information from the user.

Step

A step definition can be defined to specifically process the authentication data, or this processing can be included in a step definition that processes the data of the transaction. Either format is acceptable and depends on the overall nature of the synchronization processing performed for the transaction. The step can access the values of the authentication object using data tags within the SDML. The following syntax is the manner in which these values are accessed:

```
<<transaction.authenticationObject.propertyName>>
```

The value `authenticationObject` must be replaced with the name of the object definition being used. `propertyName` is replaced with the name of the property definition to be accessed. In a JVM system connection Java steplet, the values are accessed using the “getter” methods provided in the `TransactionSession` class. The property names are passed in as:

```
authenticationObject.propertyName
```

As with the SDML tags, the object definition name and property definition name are substituted in the above syntax.

Rules (Conditional Authentication)

As a part of the definition of the transaction authentication processing it is possible to define a rule definition to be evaluated prior to presenting the authentication screen set. This rule is evaluated in a Boolean context. A true return will result in the user being required to authenticate; a false return will not require authentication.

Transaction Authentication Behavior

The overall behavior of the transaction authentication begins on the Client. When a transaction is instantiated for which authentication has been defined, the transaction is processed as normal on the client up to the point just before it is to be applied. At this point, if the transaction is to require authentication, the authentication screen set is displayed. The user then enters the user ID, password, and any other information required. The password is validated against the password for that user to log into the Client. If this validation fails, the user is presented with an error message and the authentication screen set is then displayed again. Once the authentication is successful, the transaction is applied on the Client.

During the next transmit, the pending transaction is sent to the Server and includes the information captured in the authentication object. The transaction's server data state and server update steps have access to all properties within the authentication object. The specifics of how these values are processed depends entirely on the requirements of the back end system. The step definitions that process the authentication information can be defined to perform whatever processing is required and supported by the back end.

Defining Transaction Authentication

Prerequisites

The following items should be addressed prior to performing this procedure:

- Determine if the transaction should require authentication at all times or conditionally. If conditionally, determine the specific conditions and the values on the client involved in the determination to support the creation of the rule definition that will be needed.
- Determine the requirements for the audit trail and/or electronic signature information dictated by the back end system, including what information is needed and how it should be recorded. Note this information to support the logic needed in the back end processing for the transaction as well as in the definition of the authentication object and its properties.

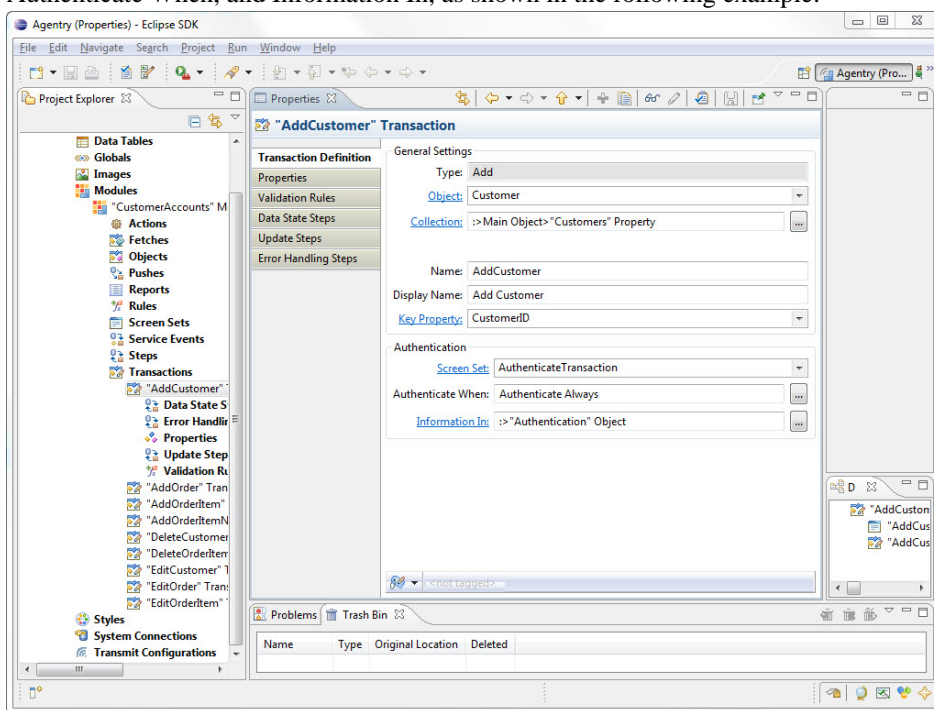
Task

This procedure provides guidance and information on implementing transaction authentication. The steps here include the main process to be followed as well as guidance on variations to these standards, which may be implemented depending on need. The main process presented here is the recommended best practice for implementing this functionality.

1. Begin by defining an object, preferably named "Authentication". Add to this object the properties needed to capture the values required for the transaction(s) for which it is to be used. If capturing the password for the user, define a string property for this purpose and set that property's **Password** attribute to true. This will prevent the value of this field from being displayed on the client, and protect it from being displayed in log files and other potentially non-secure locations.

As an alternative, it is possible to capture the authentication information in the properties of the transaction. However, the use of a separate object is the recommended method as it is easier to define and better supports conditional authentication.

2. Next define a screen to set to display the Authentication object definition. Add to it the platform(s) needed for the environment. Finally, define the detail screens and fields to display the properties from the Authentication object. Do not add button definitions to this screen set, as it will be displayed as a wizard screen set and the client will display the buttons needed automatically.
3. The transaction for which the authentication behavior can now be modified. Navigate to the transaction and view the main Transaction Definition tab in the Properties View. In the Authentication section of this tab set the authentication attributes for Screen Set, Authenticate When, and Information In, as shown in the following example:



If storing authentication values in the transaction properties rather than a separate object, be sure to leave the default Information In attribute setting of "Properties of this transaction." Also, the Authenticate When attribute is where a rule can be defined and referenced. Remember the rule is evaluated in a Boolean context with a true result requiring authentication.

At this point it is possible to publish this application and test the client-side behavior of the transaction authentication. Note that authentication-related data captured at this point will not be processed to the back end system.

4. **OPTIONAL:** If condition transaction authentication is defined, a property should be added to the transaction itself to be used as a flag indicating whether or not the authentication was performed for the transaction instance. The recommended practice is to define a Boolean property and to set the property to be initialized by a rule. In most cases the rule used to determine if the authentication should occur can also be used to initialize the Boolean property.
5. The final part of the transaction authentication is to define the logic to process the authentication data to the back end system. This is specific to the back end system's requirements for such information. When defining the step, authentication-related data is accessed using the SDML data tags `<<transaction.authenticationObjectName.propertyName>>`. For Java system connections, the values can be accessed in the `TransactionSession` class using the get methods and passing in the property names in the form `transaction.authenticationObjectName.propertyName`. In either syntax, the name of the object and the name of the property are used. The other standard transaction values, including properties and the transaction's time stamp are also accessible. Once the step is defined it can be run as a server update step within the transaction. If performing conditional authentication on the client, be sure the logic to process the authentication data checks the flag property value in the transaction to determine if the transaction was authenticated or not.

With the completion of this procedure, the transaction has been defined to require the user to authenticate themselves on the client before the transaction is applied and saved. Depending on the specifics of the configuration, the authentication may be conditional.

Next

Once this procedure is complete, the behavior should be thoroughly tested in an appropriate environment. Testing should include verification of the client-side behavior, especially any and all scenarios related to conditional authentication. Back end processing should also be verified as accurate and again scenarios should factor in any conditional authentication.

Attached Documents and File Transfer: Key Concepts

Within Agentry it is possible to retrieve files during the download portion of synchronization, store those files on the client device with reference to them from the mobile application, and to send files up from the client device during the upload portion of synchronization. Files can also be attached to objects locally on the mobile application if those files are stored on the client device.

The implementation of this functionality involves several different definition types within the application project:

- A property with a data type of External Data to reference and track the file.
- Object definition to represent the document within the mobile application.

- A list to display the documents attached to a given parent object.
- One or more step definitions of a type compatible with the back end system to retrieve information about the files associated with an object and the location of those files.
- One or more File - Document Management step definitions to download and upload attached documents.
- An action step of type Windows Command to display a selected file on the client device.
- A transaction to allow the user to attach documents locally on the client and upload them to the back end system.
- A transaction to process documents that have been changed on the local client device and need to be updated to the back end system.
- One or more rule definitions to check the state of a file, including its location, size, and whether or not it has been modified since being downloaded.

For a given implementation some or all of these definitions may need to be defined. As an example, if files are to only be downloaded to the client, but never attached locally or uploaded to the back end system, then transactions and steps to process such operations are not needed.

External Data Properties and File References

When implementing the attached documents or file transfer functionality it is important to understand how the file is stored and referenced by the mobile application. An external data property is defined to reference a file stored on the client device. Files downloaded to the client device, and files attached locally on the device for later upload are not stored with the production data of the mobile application. Rather, they are stored external to the production data, with their location on the client device referenced by the external data property.

Encapsulate the File - Object Definition

As a recommended practice, the files to be downloaded, attached, and/or uploaded should be encapsulated within the application project as an object definition specifically for this purpose. While an external data property can be added to any object definition, it is a cleaner and more manageable architecture and design to create an object definition specifically for the files and to then store instances of that object definition in a collection property at run time.

All file-related operations and behaviors are far easier to implement and maintain when following this model, including downloading and uploading files, listing the files associated with a given parent object, and other operations.

Typically an object definition to represent a file is defined to include the external data property, a property to contain the name of the file (normally a string property) and a property containing the location of the file on the back end system when that location is somewhere on the file system. Normally the property containing the file name is defined to be the key property of the object, as having more than one file with the same file name would cause issues during synchronization and storage, and designating this property as the key property will prevent such a circumstance.

Client-Side File Operations

As external files to the mobile application, files referenced by external data properties are not directly displayed in the mobile application. Rather, a list of the external data properties displayed on the client will include the name and location of that file. Similarly, detail screen fields displaying these properties include similar information about the file.

The mobile application can be defined to display an attached file in an application on the client device associated with the type of file being referenced. This application must exist on the client device and is not a part of the mobile application built in Agentry. To display a file in its native application, a Windows Command action step must be defined. It then contains as it's command the full path and file name, which is provided by the external data property. This path is passed to the operating system, which in turn “launches” the file in its associated application on the device. From this point the user can perform what ever operations the application for the file allows. The only exception to this is when the file is set to be read-only, an option within the external data property. Such files are then not editable on the client device.

When attaching files locally on the client device, a file dialog is displayed to allow the user to navigate the file system and select the file to be attached. This operation is a part of a wizard screen set displaying a transaction to support this behavior. The transaction is no different than any other, containing properties to capture data from the user. An external data property is displayed in a field type specific to that property type that supports the selection of a file from the client device's file system.

Rule Functions for External Data Properties

There is a set of rule functions available for rule definitions to work specifically with external data properties:

- **FILE_CHANGED:** This function returns a Boolean value indicating whether or not the referenced file has been modified since it was downloaded to the client device. this can be useful when determining whether or not files downloaded to the client device are in need of update to the back end system based on changes the user may have made to those files.
- **FILE_EXTENSION:** This function returns a string containing just the file extension of the file referenced by the external data property. This can be useful in filtering a list of files, e.g. show just image files (. jpg), and similar behaviors.
- **FILE_NAME:** This function returns the name of the file referenced by the external data property. This is the file name only, excluding any path information.
- **FILE_PATH:** This function returns string containing the full path to the location of the file referenced by the external data property, excluding the file name.
- **FILE_PATH_AND_NAME:** This function returns a string containing the fill path and file name of the file referenced by the external data property.
- **FILE_SIZE:** This function returns an integer that is the size of the file in bytes referenced by the external data property.

These functions can be used in various situations related to working with the file referenced by an external data property stored on the client device. Some use cases for these functions are provided in the information on implementing this functionality.

Data Synchronization for File Transfer

When defining the synchronization logic to retrieve files from the back end system for storage on the client device, it is necessary for this logic to perform a set of operations related to each file:

1. Retrieve the parent object that will contain the collection of attached documents. This logic is contained in a step type matching the back end system.
2. Retrieve the information about those files to be stored in the parent object, including the file's name, its storage location on the back end system, and excluding the file itself. This logic is contained in a step type matching the back end system.
3. Retrieve the file from the back end system. This logic is contained in a file document management step.

One of the keys to this functionality is that the back end system must have information available about which files are associated with which objects, and the specific location of those files on the back end system so that the Agentry Server can retrieve them. This location must be one to which the Server has read access.

While the above operations includes three distinct steps, it may be possible in some back ends to perform the first two operations in a single step definition. As with other synchronization operations, what data can be retrieved at which point is dependent on the back end system's structure and the interface type in use (i.e. Java, SQL, HTTP-XML). The retrieval of the actual file always requires a the definition of a file document management step.

When defining the synchronization logic to upload files to the back end system for storage in the back end system, it is necessary for this logic to perform a set of operations related to each file:

1. Upload the file to the Agentry Server and then to the back end system. This logic is contained in a file document management step.
2. Update the back end system with the necessary information about the file, including the business entity with which it is associated, revision/upload date, and any other information the back end requires for the file. This logic is contained in a step type matching the back end system.

Supporting Infrastructure Tasks

In addition to changes within the Agentry application project, there are certain tasks that should be performed in support of implementing this functionality:

- Verify the proper applications are installed to client devices in the environment for the file types expected to be a part of this functionality.
- The file system on the host system from the Agentry Server must contain a location to which the Agentry Server has read-write access. The Agentry Server typically requires a location

to temporarily store the files being transferred. This location should be noted for reference during the development and implementation of this functionality.

- The location to which files will be written on the client devices should be determined. This location is created by the Agentry Client during synchronization if it does not exist, and can include components based on the parent object of the file, as well as the user's ID.
- The method in which files are stored by the back end system should be known. The processes, tools, and/or other items that system employs should be evaluated to determine if they can be used by the file document management steps of the mobile application during synchronization. Typically such items include command line processes that can be called to check out and check in files from a version control system, or a process that extracts the files from a database when the files are stored in this manner.

Developing File Transfer and Attached Documents: Process Overview

The following are the high-level tasks in implementing the file transfer and attached documents functionality in an Agentry mobile application. Each of these items is discussed in detail in the series of procedures provided after this overview. Refer back to this overview of tasks as a check list of tasks when implementing this functionality.

1. Define the object that encapsulates the files to be transferred and attached (hereafter generically referred to as the “file object”) within the mobile application project.
2. Define the downstream synchronization logic to retrieve the files from the back end system for storage on the client.
3. Define the user interface behaviors for view files stored within a given parent object, and any related behaviors for those files, including viewing.
4. Define any transactions to add files to a parent object on the Agentry Client.
5. Define the upstream synchronization for files attached to documents on the client.
6. Define the logic, transactions, and upstream synchronization for files to be updated to the back end when they have been modified on the client after having been downloaded from the back end.

Note that if any of the functionality related to one or more of the above procedures is not to be a part of the mobile application it is not necessary to perform that procedure. For example, if users will not be modifying the files downloaded from the back end on the client, it is unnecessary to define the logic and transactions to check for such changes.

Defining the File Object

Prerequisites

Prior to performing this procedure, the following items must be addressed:

- The storage location of the files on the client device must be determined and noted for reference in this procedure.
- The parent object in which the file object will be stored must already be defined.

- The full path to the storage location of the files must be known

Task

In this procedure an example of defining an object to encapsulate the files being transferred and attached to other objects within the mobile application is provided. The primary focus of this procedure is on the external data property that references the file. Additionally, the overall architecture and components of the file object are explained during this procedure, including the reasons for their implementation and how they may be used in related functionality.

The sample application to be used in this procedure is Mobile Northwind, which contains the module data structure of Customers -> Orders -> OrderItems. Added to this structure will be a collection property for the Document object that is defined. This will be added as child to the Customer object, making it a sibling of the Orders in the aforementioned module data structure.

1. Begin by defining a new object within the module of the Agentry application project. For this example we name the object definition Document.
2. Next add a string property to this object that will contain the name of the file the object encapsulates. Note that this property need not be a string, but can be any other data type desired. It's value is converted to a string when used to name the file being saved. It is recommended it be defined as a string initially to allow for more variability in the file name. In this example we name the property FileName and has the other following attributes. (Attributes not listed here should be left set to their defaults for this example. They can be set as needed for implementation specific requirements):

- **Minimum Length:** 1
- **Maximum Length:** none (*note this can be set to a maximum if deemed necessary to the application; however, it should be sufficiently sized to ensure the file name value is not truncated*)
- **Trim:** true (*this can be important to prevent the file name from containing leading or trailing spaces that could cause issues in numerous file related operations*)

3. Now define a property to contain the location of the file on the file system of the Agentry Server's host system. This string property is used when the source location of the file is on a the file system. Define the property to be a string with. In this example we name the property BackEndFile with the following attributes:

- **Minimum Length:** 1
- **Maximum Length:** none
- **Trim:** true

This property is always required regardless of where the files are stored, including in a database system or file control system. It is needed during synchronization, as will be apparent in the procedures on this topic.

4. Define any other properties that may be needed for the file object. These could include values for reference purposes, such as the date and time the file was last modified, a

revision number for the file, or other similar information. Keep in mind any values must be accessible from the back end system in order to be downloaded for the object.

5. Next the external data property is defined. This property provides the reference to the file as stored on the client device. Begin by adding a property to the object with a data type of External Data. In this example the property is named File. Set the attributes of this new property according to the following:

- a) Set the Client File attributes to specify the source for the file name, behaviors related to if/when to delete the file, whether or not it is read-only, and the behavior of the file dialog when attaching files locally:

These attributes are set to reference the property of containing the name of the file as retrieved from the back end system with the File Name attribute set to the string property FileName. We have also defined the file to never be deleted on the client. It is possible to define the file to be deleted only when it was created by the Agentry Client during data synchronization, or to always delete file when the parent object is deleted. Finally, the file has been defined to be read-only, which means users are not able to modify the file on the client device. The File Extension attribute has been left blank, meaning files will keep the extension as returned during synchronization. It is possible specify an extension here, which changes the file extension of all files to the value

provided, regardless of the extension of the file when it is downloaded or attached locally.

- b) Next the attributes specifying where the file is to be stored on the Agentry Client are set:

The screenshot shows the 'Add External Data Property' dialog box. It contains the following fields and settings:

- Name:** File
- Display Name:** File
- Client File**
 - File Name:** :> "FileName" Property
 - File Extension:** (empty)
 - When Object is Deleted:** Never delete file
 - Read Only:** ☒ Make file read-only
 - Use Most Recent Location:** ☐ Open the file dialog in the last used folder
- Windows 9.x/NT/2000/XP**
 - Base Path:** Absolute Path
 - Relative Path:** C:\MobileNorthwindFiles\CustomerFiles
- Windows CE**
 - Use Path:** ☒ Use same path as Windows 9.x/NT/2000/XP
 - Base Path:** My Documents
 - Relative Path:** (empty)

Buttons at the bottom: < Back, Finish, Cancel.

The settings defined here result in a client device storage location specific to the mobile application and the parent object to which the files are attached. The base path under the Windows 9.x/NT/2000/XP section is set to "Absolute Path." The Relative Path attribute then contains the full path to the location where the files are to be stored. Other options include the various standard windows locations, such as My documents, My Pictures, etc. For the Windows CE section, the attribute Use Path is selected, which will replicate the path for Windows desktops on the Windows Mobile devices. The drive letter is removed from the path.

- Complete the creation of this external data property clicking the **[Finish]** button. Then, review the properties for the document object. For this example, the following now exist (others may be included in file objects like this based on need, with these being considered the bare minimum:
 - FileName:** A string property that contains the name of the file as it will be stored on the client device. Note that this may different from the file name in the back end system and is set during synchronization, which can include both the back end name and/or other

values available at run time. The synchronization procedures provided for this topic address this behavior.

- **BackEndFile:** This property contains the full path and file of the file to be downloaded from the back end system. This is referenced by the server processing for synchronization.
 - **File:** External data property that references the full path and file name for the file stored on the client device.
7. The next step for the file object is to set the key property of the file. View the Object tab in the Properties view of the Agentry Editor. Change the Key Property attribute here to the property that stores the name of the file (in our example the FileName property).
 8. Finally, create navigate in the Agentry Editor to the object that will contain the instances of the file object. In our example, this is the Customer object. Add a property to this parent object of type collection and define it to stored instances of the file object just created.

With the complete of this procedure the object to encapsulate the files associated with (or “attached to”) some other object has been defined. The parent object has been defined to contain a collection of these objects, which is to be populated during synchronization and also when files are attached on the client device.

Next

The next area of functionality to implement is the downstream synchronization processing. See the procedure on this topic for guidance on this procedure.

Defining the Download Logic for File Transfer

Prerequisites

The following items must be addressed prior to performing this procedure:

- The object definition encapsulating the files to be downloaded (hereafter referred to as the file object) must be defined. The collection property to store the file object instances must already be defined in the parent object.
- This procedure assumes a fetch exists within the application project to retrieve the parent objects of the file object.
- The location of the files in the back end system must be known and the manner in which they are to be retrieved from that location should be determined. This is especially true if a version control system is to be accessed or if the files are stored in a database.
- Agentry Client Agentry Server

Task

This procedure describes how to define the steps and step usage definitions to retrieve files from the back end system to be transferred to the Agentry Client. In the example used here the files are stored in the version control system Subversion. As a part of downloading the files,

the command line process `export` is used. This process extracts a revision of a specific file from the repository and copies it to a designated location on the file system. This copy is not maintained by the version control system, which is a desirable behavior for this logic as the copy we create will be temporary and will be removed when the Agentry Server has finished transferring it.

This procedure describes what information is needed prior to the retrieval of the actual steps. How this information is retrieved will vary from one system and one back end type to the next. However, the general information required for this processing is the same.

As will be illustrated in the portion of this procedure related to the object read steps, one of the key items is to define the proper run behavior for the Document Management step that actually retrieves the files. Files are retrieved one at a time. Therefore, the document management step must be defined to run once for each file to be retrieved. This is controlled by the Run attribute of the object read step definition that uses the document management step. Note that only the object read step definition type contains the proper setting for the Run attribute. This step cannot be run to retrieve files in a fetch step usage definition.

In order for the document management step to run properly, therefore, the files to be retrieved must be known in advance. This is a part of the general information retrieved from the back end, where presumably the information relating the files to the business objects is located. this information is used to instantiate the file objects and, therefore, must include the value of the key property for each of those objects. In the architecture recommended here, this is the name of the file to be retrieved as it will be stored on the Agentry Client.

In this example, the Customer object contains a collection of file objects named Documents. A fetch already exists to synchronize the Customers collection, including it's nested collections Orders and Products. To this processing we will add the steps to get the document objects and the actual files to be referenced by those objects.

1. First, define the step to retrieve the property values for the file object, other than the actual external data property. In our example the file object includes the values of the FileName and BackEndFile properties. As with the synchronization of any nested collection, this processing must also include the key property of the parent object, which in this case is the CustomerID. For the sample Mobile Northwind application a SQL statement is written that selects the BackEndFile location and the FileName values from the table CustomerDocuments in the database. The step definition is a SQL step named GetDocumentInfo.
2. Next the document management step must be defined to retrieve the actual files to be transferred. This consists of several pieces, which are broken down over the next steps of this procedure. Begin by adding a step to the module of type File Document Management that uses the file system connection within the project. In the second wizard screen, select the attributes to specify which object is to contain the files it retrieves. In our example application the selection for this attribute is "Object - Document." Steps of this type need to know for which definition they will be retrieving data to provide access to the proper

values to the logic they contain, as will be demonstrated in the following steps. Set the name and group attributes and then finish the wizard.

3. Now that the step is created, the next task is to define the document mapping(s) for the step. Document mappings define the relationship between the properties of the definition selected when the step was created, and the command the step will execute at run time. In the Properties view of the Editor, select the tab Document Mappings. To add a new mapping, select the add button above the empty list. The Add Document Mapping wizard displays.

Set the attributes of Property, which should be the external data property for the file object. Then, set the Output Type, which is how the file to be transferred is output by the command we will write to retrieve the file from the back end. In our case, the file is retrieved from the version control system and saved to the local file system of the Agentry Server, so the proper selection is “File created by command.” Other options are to capture the processes output to standard error or standard out, or the processes exit code. These are discussed in the later, optional steps to this procedure. Leave this wizard open to set the next attribute, File Name, as discussed in the next step of this procedure.

4. First, check the box below the File Name attribute marked Delete File. This setting removes this temporary file from the file system for the Agentry Server once the Server has completed processing it. The File Name attribute specifies the name of the file as it will exist on the file system of the Agentry Server once it has been created by the document management step. This file name can be set to any value, as it is only temporary and does not need to match the file name as it will be stored in the Agentry Client. This attribute can, and likely will include SDML data tags. In our example, the file name is a combination of the fileName property of the file object, the CustomerID of the parent Customer object, and the user’s login value. This combination guarantees a unique file name for the file on the Server. This attribute can, and in most cases should include the path information to where the file is to be stored. This location should be one to which the Agentry Server has read-write access and is typically one created specifically for use by the Agentry Server. The full value of this attribute in our example, then, is:

```
C:\MobileNorthwndFileTransfer  
\<<document.fileName>><<customer.CustomerID>><<user.agentryID>>.t  
mp
```

Note that any property value from the object is valid with the exception of the external data property. This value, then creates a file in the specified location, with a name guaranteed to be unique for each user and customer object combination. As demonstrated shortly, the values used here to create the file name should be noted, as they will be needed in the creation of the command for the document management step. Once the File Name attribute has been set, finish the wizard.

5. **OPTIONAL:** Additional Document Mappings can be defined for this document management to capture other information from the command it executes. The Output Type setting controls this behavior, with the options Command Exit Code, STDERR, and STDOUT. Each of these values produced by the command can be mapped to a property

within the definition for which the document management step was defined, e.g., the Document object in our example. The Command Exit Code and STDERR options are typically selected to capture any errors to support processing them accordingly. STDOUT is selected when the process executed by the document management step streams the file data to standard output rather than creating a file on the file system. STDOUT can be mapped to the external data property when the process behaves in this manner.

Create any additional document mappings, as needed. The attributes File Name and Delete File are not available when these options are selected, as there is no physical file included in the processing.

6. Next the command to retrieve the file from the back end is written. In the Properties View select the Document Management Script tab. The command can be written either directly in the Command attribute (set to `<<script>>` by default) or in a separate script file. To run a separate script, the default `<<script>>` data tag should be left as the value for the Command attribute. The separate script file is the default behavior, and the Command field is typically only used when a single short command is needed. In most real-world applications the command is stored in the separate script file. Click the edit button to the right of the File attribute on this screen to display this script, which is shown in the text editor view.
7. The command executed can be any series of one or more command line utilities that may be needed to retrieve the file from the back end storage system. In our example we assume the version control Subversion contains a repository of files for customers in the Northwind system. We extract the files, therefore, from this system. We use the `export` command in Subversion for this processing, writing the file to the file system for the Agentry Server. Regardless of the tools used, the command written here must extract the file from the back end system, making use of the information retrieved previously regarding which file and where it is located. the command can then either write the file to the file system, or stream it to standard output for the Agentry Server to capture. The proper behavior must be matched to the defined document mappings for the Document Management step. The following is the example command used for the Mobile Northwind application:

```
export <<document.backEndFile>> C:\MobileNorthwindFiles
\<<document.fileName>><<Customer.CustomerID>><<user.agentryID>>.t
mp
```

The above command executes the `export` utility for the Subversion system. It extracts the file from the location where it is stored in the repository, which is the value returned by `<<document.backEndFile>>`. The file is then written to the location where the Agentry Server expects it. The file name is then set to match the values of the name of the file in the back end (`<<document.fileName>>`), the parent CustomerID property values (`<<customer.CustomerID>>`), and the user's client login (`<<user.agentryID>>`).

8. Now that steps are defined, they must be used. For file transfer functionality this requires the use of object read steps. In our sample Mobile Northwind application there already exists a fetch named GetCustomers, which targets the top-level Customers collection. Therefore, the read steps will be added to the Customer object and read into the Documents collection property of that object. First, the GetDocumentInfo step must be run. Remember that this is the first step we defined in this procedure and is the one to retrieve the information about the file to be transferred. Create a read step within the object to run this step. The Run attribute for the mobile application is Run one time, as all information about the files can be retrieved in a single query. For other systems, determine if the logic can perform this type of batch processing and set the Run attribute accordingly.
9. Next we define the read step for the document management step. In the Mobile Northwind application this is the GetCustomerFiles step. When defining this read step, the attributes should be set to reference the document management step and to read into the collection of file objects. The Run attribute must then be set to Run Once per Collection Object. This will execute the document management step once for each of the file objects contained in the collection, returning the file based on the other property values of that object.
10. Once these read steps are defined, verify the proper order. The first step run should be the one that retrieves the information about the files to be transferred. The result of this step's execution is the creation of the file object instances to be stored in the collection. This should then be followed by the document management step that retrieves the actual files. Since this step is executed once per collection object, the step to create the file objects must come first, followed by the document management step.

With the completion of this procedure the file transfer functionality is added to the mobile application. If following the steps and general architecture of this procedure, the following is the overall processing and data flow of this change:

1. A transmit is initiated by the Agentry Client. The fetch for the top level collection is processed as it was prior to this modification.
2. The object read steps of the object type being targeted by the fetch are processed by the Agentry Server. When the step to retrieve the information about the files to be transferred is run, the Agentry Server processes the results of that step and instantiates the file objects, storing them in the collections of the parent object instances. These file objects include the information about where the file is located in the back end, and the name of that file.
3. The document management step is run next. It is executed once for each object instance in the collection of file objects. For each execution, the file is retrieved from the back end system and stored in a temporary location for the Agentry Server to access it. This location is determined by the command in the document management step.
4. When the command completes execution, the Agentry Server uses the information in the document management step's document mappings to read the file from the location the mapping specifies. It sets the property the mapping specifies within the file object to reference the file it reads in.
5. The file is transferred by the Agentry Server to the Agentry Client, where the file is stored on the client device according to the storage location defined in the external data property.

6. The above steps concerning the execution of the document management step are repeated once for each file object until all have been processed.

Next

If not already accomplished, the user interface to display a list of the files downloaded, as well as to possibly allow the user to display those files and edit them needs to be implemented. See the procedures in this tutorial on the user interface and actions related to the files downloaded to the Agentry Client.

Defining the User Interface for Attached Documents

Prerequisites

The following items must be addressed prior to performing this procedure:

- The object that encapsulates the files stored on the Agentry Client (hereafter referred to as the file object) must already exist and all properties it is to contain must be defined.
- Though not a technical requirement, it is generally good form to define the synchronization logic for the objects prior to defining the user interface.
- A part of the functionality implemented in this procedure includes the ability for users to select a file from a list of those on the Agentry Client and to display that file in the application associated with the file type. This functionality can be implemented in the Agentry application project at any time. However, it will not function properly unless the application is installed to the client device and is associated with the file type. This requirement must be met before deployment of this functionality.

Task

This procedure described how to build user interface definitions and functionality around files that have been transferred to the Agentry Client and/or attached to objects locally. The Agentry Client does not display files directly within the client interface, nor does it provide a means to edit these files to the user. However, the files can be displayed in an application for the file type, provided that application has been installed and is associated with the file type on the client device.

To list the files attached to a parent object, a list screen or one of the list types of detail screen fields can be used. The definition of these lists is the same as for the display of any other collection property. The collection is selected to be listed, and the properties from the object type within the collection are selected for display in columns for fields.

In this procedure a list is defined to display a collection of file objects. An action is then defined and a button is added to execute that action that displays the file currently selected in the list. This is accomplished with the definition of an action step of type Windows Command. This type of action step is used to execute commands on the client device. If the path to a non-executable file is defined as the command for the step, the Windows OS will execute the application associated with that file type and open the specified file.

In the example used in this procedure, the module data structure is Customers -> Documents, where Documents is a collection of Document objects to encapsulate the files attached to Customer objects. A screen set to display details of a selected Customer object is already defined and includes detail screens for the other properties of the Customer object. It is to this screen set, named ShowCustomerDetails, that the list screen will be added. The functionality to be implemented includes a list screen with columns displaying the name of the file stored on the client, and another to display the full file path. On this list screen an action can be executed via either a double-click of an item in the list, or via a button click that displays the selected file in the application associated with its type.

Note that similar behavior can be defined to list the Documents object using other list controls available in Agentry, including List Tile View fields and List View fields, or any display definition that lists collections.

1. We will begin by defining the action to display the selected file in its associated application on the client device. Begin by adding a new action to the module. Set the Name and Display Name as deemed appropriate. The For Object attribute should be set to the file object containing the external data property. In the sample Mobile Northwind application this is the Document object definition.
2. Once the action is defined, add a new step to it of type Windows Command. Again set the Name as Desired. Then set the attributes of this action step according to the guidance provided below this example:

Add Windows Command Step to Action: OpenFile

Step Name: OpenFile

Command Line: %File

Wait: ☐ Wait until the command returns to continue with the action

Wait Period Limit: 0 : 00 : 30

Error Dialog to Display When the Command Returns Error

Error Message: Unable to open file. It may not have an associated application, or it may not be accessible at the specified location.

Timeout Message:

Continue Label: ☐ Not Allowed ☒ OK

Cancel Label: ☒ Not Allowed ☐ Cancel

< Back Finish Cancel

- **Command:** To display a file, the value here must be the full path and file name of the file to open. A format string can be used here for the External Data property of the file object (In the Mobile Northwind this is the File property), e.g. %File
- **Wait:** This attribute controls whether or not to wait for the executed process to complete before continuing execution of the action. When selected, a timeout value is required. For this operation such behavior does not make sense, as there is no way to know how long the user would view the file and setting a wait behavior would prevent the user from accessing the Agentry Client while the external application is still open. Therefore, leave this attribute set to false.
- **Error Message:** This is displayed when an error occurs executing the command. For this step, the message can indicate there was a problem opening the file. Note that the operating system may also return an error message should this operation fail and both will be displayed to the user.
- **Timeout Message:** This message has no affect on this step definition and can be left blank, as the Wait attribute is set to false.
- **OK and Cancel Labels:** These are the buttons displayed in the error message dialog. In most cases either the OK or Cancel would be enabled, but no both. In the Mobile

Northwind application this action contains only the single Windows Command step, with no additional steps being executed. Therefore, the OK is sufficient to acknowledge the error message on the Client. If additional steps are included in the action after the Windows Command step, then it should be determined if the user should be allowed to continue the action or not, and to enable the buttons appropriately.

This step will now pass the full path and file name to the operating system, which will then result in the OS attempting to open the file in the application associated with the file type.

3. Now the list to display the file objects stored in a collection can be defined. This can include a list screen, a List View detail screen field, or a List Tile View detail screen field. The collection should be the collection property of file objects. In the Mobile Northwind application project this is the Documents property of the Customer object. Typically the file name and file path are displayed. For the Document object this would be the FileName string property and the File external data property, respectively. The external data property type always displays the full path and file name of the file being referenced when a UI definition targets it for display.
4. Finally the control(s) are defined to execute the new action. This can include a button definition on the list screen, as defined in this example, that targets the selected Document object in the list. Additionally, or in place of the button, a double-click on-item action can be defined for the list screen that executes the OpenFile action as well, targeting the Document object. If a List View or List Tile View field is defined to list the Documents collection, a button can be defined on the parent detail screen that targets the selected object in the list field.

Once this procedure is complete, a list of the file objects for a given parent object is defined. An action also exists that allows the user to display the selected file in the application on the client device associated with that file type.

Defining Locally Attached Documents Functionality

Prerequisites

The following items must be addressed prior to performing this procedure:

- An object must exist in the application project to encapsulate the files to be attached to a parent object (hereafter referred to as the file object).
- Options exist to specify the location in which files can be selected on the client device. If this behavior is desired, this location should be determined and noted for this procedure.
- Options exist to restrict the type of file a user can attach on the client device. This is controlled by the file extension. If this is desired behavior, the file type and extension should be noted for this procedure.
- The manner in which files should be stored in the back end system should be researched, including details on how to add the files to that storage location or repository. This processing must be a part of the logic defined to upload the file to the back end.

Task

In this procedure the tasks necessary to allow users to attach files to an object on the client, selecting that file from the client's file system, are provided. This functionality includes the use of an Add transaction to create a new file object and to capture the properties of that object, including the file to be attached, from the user. A transaction screen set is defined that includes a field to display the external data property. The field for this property type displays a file dialog to the user when it is selected in the wizard, allowing the user to navigate to the file to be attached.

The synchronization of this data includes steps that update the back end system with the information about the file, including its name, the parent object it is attached to, and other information as may be required by the back end system. Typically this information is updated to the back end using a step definition of the type matching that back end, i.e., a Java, SQL, or HTTP-XML step. A document management step is defined to perform the actual file upload and back end processing necessary to store that file in the location or repository where files are to be stored.

1. The first task is to define an Add transaction for the file object. As with most add transactions, this one should include transaction properties matching all defined object properties. In the Mobile Northwind example application, the Document object encapsulates the file attached to a parent object. An Add transaction named AttachDoc is defined that targets this object type. It contains the following properties, matching those in the object:
 - **File:** External data property, initialized to empty, i.e. Initial Value: "Auto-Initialize." There are some optional changes to the default attribute settings for this property, discussed in the next step of this procedure.
 - **FileName:** String property. This property is modified from the default initialization to a rule after data entry. A subsequent step in this procedure describes this modification.
 - **BackEndFile:** String property, initialized to an empty string. This value is not needed for transaction processing, as it only relates to downstream synchronization (e.g., fetch processing). However, it is good form to initialize the value in the add transaction.

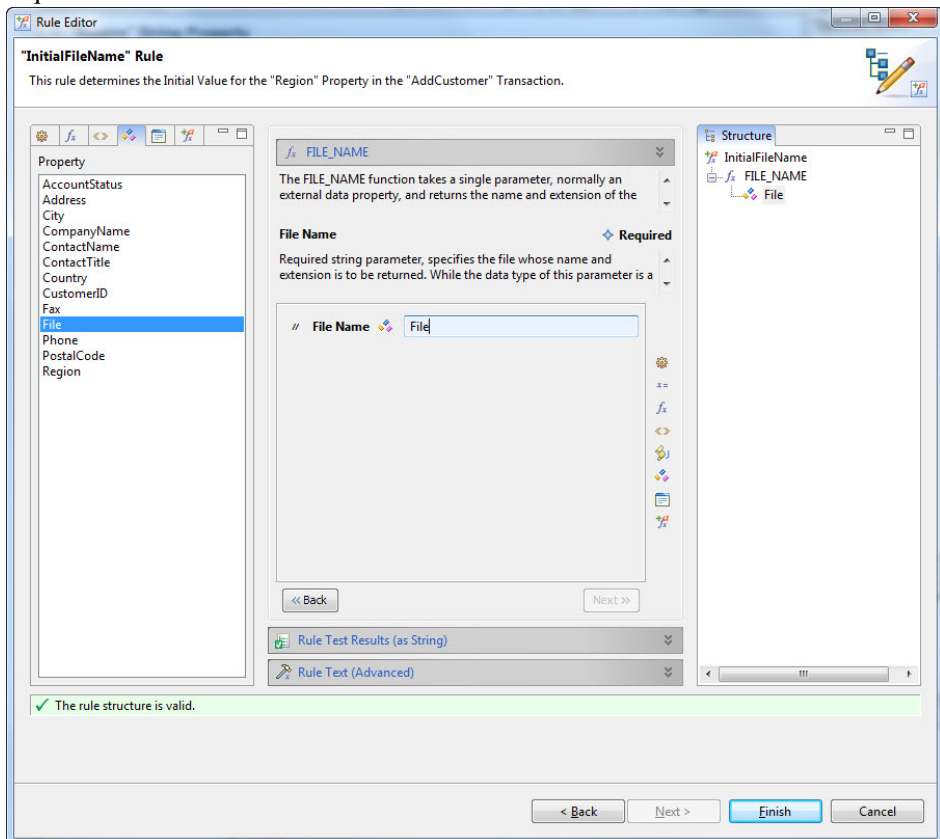
When the transaction is created, assuming the properties from the object were all selected in the add transaction wizard, there will be matching properties in the new transaction definition. In subsequent steps, a property to capture the parent object's key property value is added to this transaction. First, the initial value of the FileName property must be defined to capture the name of the file on the client device.

2. **OPTIONAL:** It is possible to provide a white list of files for file types the user can select, and to also provide a black list of specific files or locations from which the user may not make a selection. Once the external data property has been defined, viewing it in the Properties View of the Agentry Editor displays the three attributes of **File Filters**, **Filter Description**, and **Restricted Files**. **File Filters** can be set to one or more file types the user can select. Only files of that type will be displayed in the file dialog, and wild cards can be used (e.x., *.doc; *.jpg - Only MS Word Documents and JPEG images). The

Restricted Files attribute must be set to an absolute path to the directories, or specific files, that the user will be allowed to select. Multiple paths can be provided and must be pipe delimited. (e.g. \\Windows | \\Program Files - No files can be selected from any path under these two locations.)

If these attributes are set, the file dialog on the Agency Client will present only those files and file types defined in the **File Filters** list, and will prevent users from selecting the files found in the **Restricted Files**. This is optional behavior and is not a requirement of this functionality. However, it is recommended that options for these attributes be at least considered to prevent users from attaching files that should not be transferred, e.g., executables, resource files, etc.

3. The FileName property must contain the name of the file on the client device's file system. This value is not needed during synchronization, but is used for display and reference purposes on the Agency Client. The property is therefore modified by changing the **Initial Value** attribute to "Rule - after data entry." The rule is then defined for this initialization to use the FILE_NAME function, taking the external data property in the transaction as its sole parameter. It returns just the name of the file, to store in the FileName property as required:

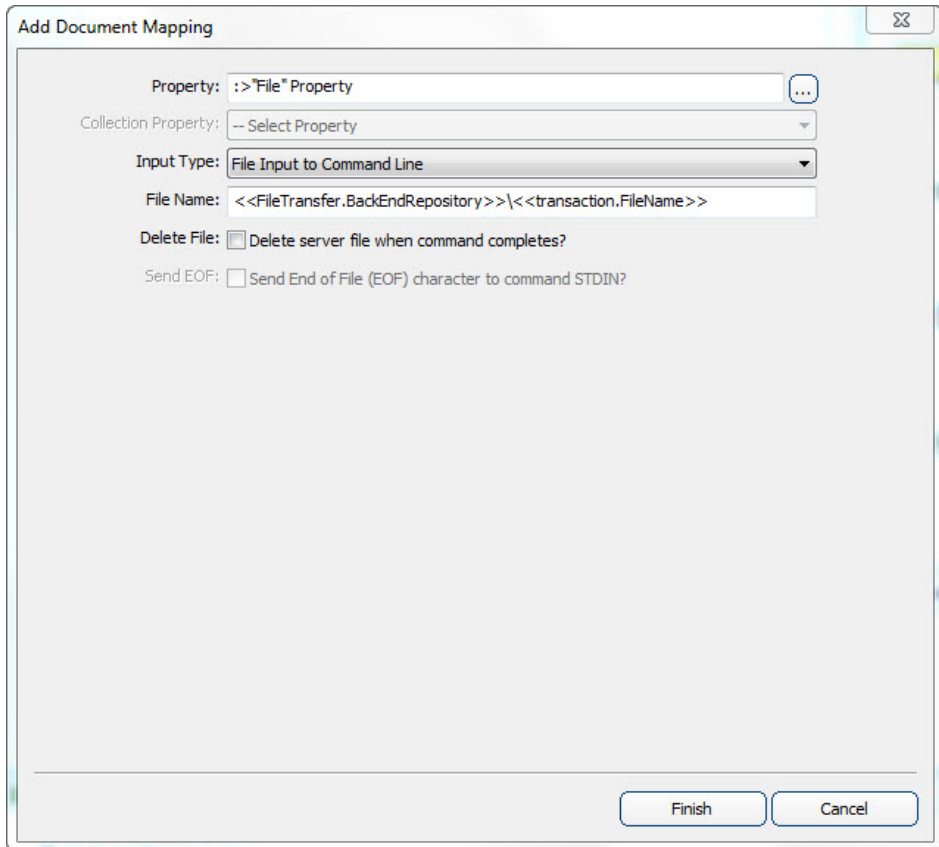


With the change to the property's initial value attributes, and the definition of the rule, the FileName property within the transaction is now defined to capture the name and extension of the selected file as stored on the client device. The rule contains a simple structure consisting of the FILE_NAME function that takes the external data property (File in the Mobile Northwind example) as it's single parameter, returning the name of the file it references.

4. Now that the transaction has been defined, the synchronization logic to update the captured information can be defined. We begin with the document management step responsible for updating the actual file to the back end system. Create a step of type File Document Management. Use the defined File system connection. Advance the wizard and on the enxt screen, set the **Used By** attribute to "Transaction - *AddDocTransactionName*", where the second portion is the name of the transaction defined to add a file to a parent object. Set the **Name** and **Group** as desired and finish the wizard.

In the Mobile Northwind example application the step is defined to be used by the transaction AttachDoc. The name of this step is CommitCustomerDoc. Next document mappings for this new step must be defined.

5. The document mappings for a document management step used by a transaction define the source property in the transaction referencing the file being transferred, and how that file is to be provided to the command run by the document management step. For the Mobile Northwind example application, the File property contains the reference to the file being transferred. The command being run includes command line processes within the Subversion version control system. These commands expect the file to be stored on the file system. Therefore, the document mapping is defined as follows:



The screenshot shows a dialog box titled "Add Document Mapping". It contains the following fields and options:

- Property:** A text box containing the value `:>"File" Property` with a browse button (three dots) to its right.
- Collection Property:** A dropdown menu currently showing `-- Select Property`.
- Input Type:** A dropdown menu currently showing `File Input to Command Line`.
- File Name:** A text box containing the value `<<FileTransfer.BackEndRepository>>\<<transaction.FileName>>`.
- Delete File:** A checkbox labeled `Delete server file when command completes?` which is currently unchecked.
- Send EOF:** A checkbox labeled `Send End of File (EOF) character to command STDIN?` which is currently unchecked.

At the bottom right of the dialog are two buttons: **Finish** and **Cancel**.

- **Property:** The external data property of the transaction.
- **Input Type:** How the file is to be provided to the command run by the step; for this example this is set to "File Input to Command Line," meaning the file is stored on the file system by the Server prior to executing the command for this step. The other option is to stream the file data from the Agentry Server to a command being executed.
- **File Name:** Where the Agentry Server should write the file to on the file system. This is disabled if streaming the file to the command. Note that the FileName property is provided for this attribute as a matter of convenience. This attribute specifies the name the file should be given when written to the file system by the Agentry Server. The value specified here can be any desired using any values available to the document management step. Like other steps, this is any value in the transaction, any globals defined in the application project, and any of the other available values for back end processing.
- **Delete File:** Whether or not the file should be deleted. In this example the file is being written to the working copy location of the Subversion repository and therefore should not be deleted. In other cases, if this location is a temporary storage to allow the file to

be processed and moved by the command, then this option should be selected to remove the file when its processing has been completed.

Once the document mapping is defined, the Agentry Server will process the file as that mapping dictates, including which property contains the file, how to provide the file to the document management step's command, and whether or not to clear the file from the file system when processing is completed.

6. The command for the document management step must now be written. This command processes the file provided by the Agentry Server, either via a file written to the file system, or via a stream to standard input of the command being executed. The command written for the Mobile Northwind application is stored in a batch script and contains the Subversion commands add and commit. The add command adds a new file to the Subversion repository and the commit command commits that addition. The command is then written as follows:

```
add <<FileTransfer.BackEndRepository>>/
<<transaction.FileName>>
commit <<FileTransfer.BackEndRepository>>/
<<transaction.FileName>>
```

When this command is defined, the file is first added to the repository and then subsequently committed to the repository. This is a required set of operations for Subversion. Other version control systems will handle new files differently. This processing assumes the file did not exist in the repository prior to this operation. If files are to be updated in any system, Subversion or others, the order of operations would be different. In general this command should be written only with a full understanding of how the back end stores files and how different situations are handled, e.g., new files vs. updates, etc.

7. In most systems it is necessary to create the logical link between the file and the business entity to which it is attached. For example, in the Mobile Northwind application the table Customer Files exists in the Northwind database and contains the path to the file's location in the version control repository and the Customer ID of the customer with which the file is associated. For this application a SQL step is defined containing an insert statement that adds the needed record to the table for a new file. Other information can be included, such as date and time information, file size, or any other meta data that may be needed.
8. Once the steps have been defined, they must be used by the transaction. A server update step is added for each step involved in the processing of an attached file. In the Mobile Northwind application this includes the document management step and the SQL step discussed in this procedure. In most cases the recommendation is to first execute the Document Management step, followed by any steps that provide the link or other information about that file. This order of operations prevents there being any "empty links" should an error occur with processing the file.
9. The final step in this procedure is to define the user interface for this functionality. This procedure is the same as defining the wizard for most other transactions. The screen set is defined, including the proper platforms, and finally the detail screen(s) and fields to

capture the data from the user. The one item of note is the definition of the field for the external data property. The edit type of this field can be set to “External Data”, or left set to “-- Default --”. At run time, the Agentry Client recognizes that the target property of the field is an external data property and automatically displays a field that includes a control to launch a file dialog to allow the user to select a file from the file system. However, selecting the External Data field type explicitly can make it clearer when returning to this definition in the future. Once the wizard has been defined, the action for it and the transaction can be defined, followed by the control(s) to execute that action.

When this procedure is complete, users will have the ability to navigate to and select files on the client device. These files are then attached to the parent of the file object. During synchronization, the transaction processing will include uploading the file to the Agentry Server and then storing that file in the back end system

Next

A variation on the above functionality is the ability to allow users to select multiple files from the file dialog and attach them in one operation to a parent object. This requires the following differences in the definition of the transaction:

- Define a collection property of file objects in the transaction in place of the external data property
- When the detail screen field is defined for the wizard where files are selected, its edit type must be set to “External Data” before the collection property in the transaction can be selected.
- The Document Mapping for the document management step must be defined for the collection of file objects in the transaction. An additional attribute, Collection Property, is enabled where the external data property from the object in that collection is selected.
- The Document Management Script’s Command (or more likely the script containing the commands to be executed) must include iterative processing for each object in the collection. This processing can be provided using the SDML function tag `<<foreach...>>`. The commands execute within this loop then would operate on a single file. `<<foreach...>>` provides the context of each object instance in the collection so that each file can be processed individually. see the example below for the pseudocode representing how this command script would be written.

```
<<foreach Documents
```

```
  copy <<my.FileName>> to <,FileTransfer.RepositoryDirectory>>
```

```
  add <,FileTransfer.Repository>>\<<my.FileName>> to repository
```

```
  commit <<FileTransfer.Repository>>\<<my.FileName>> to  
repository
```


>>

The Agentry SDK

The Agentry SDK is a collection of resources provided to developers to support inter process communications between the Agentry Client and another process or application running the same client device.

Note: The ActiveX API and Agentry Client API should both be considered deprecated in the SAP Mobile Platform 3.0 release. They are provided solely for backwards compatibility in support of existing implementations in which either of these resources were used. With the release of the SAP Mobile Platform 3.0, the OpenUI SDK should be used for all new development work.

There are two options available for interacting with external processes. These options consist of ActiveX, or using the Client API. The ActiveX interface to the Agentry Client has been available for quite some time and has been expanded and augmented with continuing increases in the exposure of the Agentry Client functionality and data. The Agentry Client API exposes the ability to execute actions, instantiate and apply transactions, and request a rule evaluation, all from an external process running on the same device as the Agentry Client.

Included in the Agentry SDK, and in addition to the resources needed to make use of the API's it provides, are a handful of samples. It may be beneficial for the uninitiated to review these samples, and possibly even compile and build one or more of them in order to become more familiar with the overall structure and logic involved in building processes or controls that interact with the Agentry Client.

Agentry ActiveX Client API

The Agentry ActiveX Client API (ActiveX API) provides numerous resources for creating an ActiveX control that is displayed on the Agentry Client's user interface and that can interact with the Agentry Client in several different ways, including passing various types of data between the Agentry Client and ActiveX control, requests made by the control of the client to execute actions, and for the Agentry Client to be aware of various control-related events such as data entry and changes in focus.

The ActiveX API provided by Syco includes numerous resources needed by the ActiveX control that must be included in the build and compile stages. There are methods within the Agentry Client that are exposed to the ActiveX control, as well as methods that are expected to exist within the control that will be called at various times by the Agentry Client to notify the control of certain events related to the control and/or the user's interaction with it.

Agentry Client API

The Agentry Client API has been provided to expose certain functionality within the Agentry Client to external processes. Similar to the ActiveX API, the external process can request the Agentry Client to execute actions. In addition, the process can also request the Agentry Client to instantiate a transaction using values provided by the external process, and to then apply that

transaction. The external process can also call through the Client API to request a rule evaluation and to receive the value returned by that rule.

A significant difference between the Client API and the ActiveX API is how the communications are supported. When using the ActiveX API it is a requirement that an ActiveX control be used and displayed on the Agentry Client's user interface. In some situations this is either not practical, or such a control makes no sense for the intended purpose. The Agentry Client API allows for interaction between an external process and the Agentry Client with such a control. The external process must be built using the provided resources in the Agentry SDK, and call the methods provided by the Client API to perform the desired processing.

System Support, Usage, and API Differences

Both the Agentry ActiveX Client API and the Agentry Client API are available for use with Agentry Client's running on Windows devices, desktops, and laptops. ActiveX is a Microsoft protocol provided exclusively for their family of Windows operating systems, and therefore cannot be used in conjunction with Agentry Clients for platforms other than Windows. The Agentry Client API is at this time available only for Windows platforms as well.

The primary driver for selecting the Agentry Client API for external processes or the Agentry ActiveX Client API is whether or not a control should be or is needed to be displayed on the Agentry Client's user interface. If there is no need for such a control, or if such a control does not make sense in the context in which the processing or work flow is performed, then the developer should investigate using the Agentry Client API for external processes. If, however, a control is needed, then the Agentry ActiveX Client API must be used.

Other differences include the functionality exposed and available by each API. The Agentry Client API for external processes allows for the execution of actions, the evaluation of rules, and the instantiation and application of transactions. All three of these processes are performed in the context of the module MainObject.

A Note on Changes to the Agentry SDK

For developers familiar with the Agentry ActiveX Client API, it is important to note some items related to the recent addition of the Agentry Client API for external processes. The most important difference is that the Client API does not utilize any of the resources provided by the SDK for ActiveX, nor does it use the methods, field types, action types, or other components provided to support ActiveX controls. The Agentry Client API for external processes is a separate entity and all support and resources related to its usage are mutually exclusive from the Agentry ActiveX Client API related resources.

No changes were made to the ActiveX API related to the addition of the Agentry Client API. Any existing ActiveX controls built using the ActiveX API provided previously are still supported and no change to them is needed.

Technical Overview - ActiveX Controls and the Agentry Client

The Agentry Client is capable of interfacing with an external ActiveX control installed to the same host device. This functionality is supported through the implementation of several separate but tightly coupled components within the Agentry architecture:

- The **External Field - ActiveX** Control detail screen field edit type
- The Agentry Client API containing methods that can be called by the ActiveX control
- The ActiveX control's proper implementation of the interface points (methods) expected by the Agentry Client
- The client action step type **External Field Command**

Using the above components together, it is possible for the Agentry Client to display an ActiveX control within the Client's user interface, to pass data from the Client to the Control, to pass data from the ActiveX control to the Agentry Client, to execute actions on the Agentry Client at the request of the ActiveX control, and to issue commands from the Agentry Client to the ActiveX control.

Provide Data to the ActiveX Control

The External Field - ActiveX Control screen field edit type includes in its definition a list of objects and properties, known as the Agentry Values for that field. Each of these properties is selected and added as an Agentry Value for the screen field. When a value is added, the specific property or object is selected and given an arbitrary name.

The ActiveX control can call into methods provided in the Agentry Client API to retrieve these values. The value to retrieve is specified via the name given to it in the Agentry Values list. All property values, regardless of the data type within the Agentry application project, are provided to the ActiveX control as strings.

Pass Data to the Agentry Client

The ActiveX control can call methods in the Agentry Client API to notify it of a change in its current value, or to indicate that the value has been fully entered. A call to these methods results in an immediate call by the Agentry Client back to the ActiveX control to retrieve the current value. The value returned by this subsequent call is then set as the current value of the External Field - ActiveX Control screen field. If no further changes are made, this value will set the value of the property target by the External Field if that field is displayed in a wizard screen for a transaction or fetch.

Execute Actions in the Agentry Client

The ActiveX control can call into the Agentry Client API to execute actions defined within the mobile application. This behavior requires the action or actions the control may execute to be listed within the External Field - ActiveX Control screen field. Only those actions listed within the detail screen field can be executed by the ActiveX control. The ActiveX control calls the appropriate method within the Agentry Client API, passing the name of the Action to be executed.

Issue Commands to the ActiveX Control

The Agentry Client can issue a command to the ActiveX Control. An action step of type External Field Command can be defined to pass a string value to the ActiveX control. The ActiveX control receives this command string via a method called by the Agentry Client when the action step is executed. The action step defines the External Field - ActiveX Control screen field that references the ActiveX control. This screen field must reside on a detail screen within a screen set defined to display an Object. Screen sets displaying transactions and fetches will not be valid options in the action step when it is defined.

External Field - ActiveX Control

The external field-ActiveX control edit type is defined to call out from a field to an ActiveX control. Values may be passed to this control from the Agentry Client.

Use of this field requires an ActiveX control exist on the client devices and that control be built using the Agentry ActiveX Control API, including the implementation of all Expected Methods.

Using the **Agentry Data** and **Actions** tabs allows an ActiveX control to query Agentry for data and for an ActiveX control to call for Agentry to execute actions. Agentry can also query the ActiveX control for any values listed in the **External Values** tab.

External Field - Active X Control Attributes

The following attributes are specific to the External Field - ActiveX control field edit type. These are in addition to the common field attributes:

- **ActiveX Class Name (Prog ID):** This attribute contains the class name that the Agentry Client will interface with for the ActiveX control.
- **Allow Scanning as Input:** This attribute specifies whether or not the field displayed will accept barcode scan values as input. This attribute will only impact fields displayed on detail screens used by a platform that supports scanner behavior and on client devices equipped with a barcode scanner. When value is scanned for the field, the ActiveX control expected method `AgentryUpdateScanData` to pass the barcode value to the ActiveX control.
- **External Values Tab:** The External Values tab is a list of values provided by the ActiveX Control. This will allow the Agentry Client to query the control for data. From the tab, you can add and delete value names from the list. The ActiveX control referenced by the detail screen field must include the proper processing within the `AgentryGetSpecificValue` method to return the value(s) associated with each of the External Values listed in this tab.
- **Agentry Values Tab:** The Agentry Values tab is a List of names and target paths for values within Agentry, made available to the ActiveX Control. From the tab, you can link Agentry data with the external values for the ActiveX Control. Both primitive data types as well as object instances and collection properties can be made available to the ActiveX control. The name associated with the selected data item is the identifier exposed to the ActiveX

control, which can call the `GetPropertyFromMappings` Agentry Client-Side API method, passing the name to retrieve the desired value.

- **Actions:** Allows the ActiveX control to call for Agentry to execute actions. The Properties tab gives you a list of Actions and target paths. Within this list actions can be added and deleted. When an action is added it must also specify a target object for the action. The ActiveX control can call the `ExecuteAgentryAction` Agentry Client-Side API method, passing the name of the action to be executed.

Action Step Type: External Field Command

The External Field Command action step issues a command to an ActiveX control when executed. It references the External Field - ActiveX Control field to specify the control to which the command is to be issued. The action step passes the value of the defined command string to the ActiveX control, which is then responsible for receiving and processing the string command accordingly.

The defined command string within this action step type is passed by the Agentry Client to the ActiveX control through the expected method `AgentryExecuteCommand`. This method should be implemented to process the provided command string in the manner deemed appropriate for that control.

External Field Command Step Attributes

- **Step Name:** This attribute contains the unique internal name of the action step definition. This must be unique among all steps within the same parent action.
- **Screen Set:** This attribute specifies the screen set containing the detail screen within which the External Field - ActiveX Control field is defined. Valid selections for this attribute include any screen set defined to display an object definition. Screen sets for transactions and fetches are not valid.
- **Screen:** This attribute specifies the detail screen containing the External Field - ActiveX Control field.
- **External Control:** The External Field - ActiveX Control detail screen field that references the ActiveX control to which the command string is to be issued.
- **Command:** The string to be passed to the ActiveX control's `AgentryExecuteCommand` method. This attribute value can be entered into the attribute field directly, or can be set to the return from a rule definition. A rule referenced by this attribute is evaluated in a string context and in the context of the action to which the action step is being added and the object for which that action is defined.

ActiveX Control - Features Log

The following provides a quick overview of the progression of the ActiveX control functionality supported by the Agentry Mobile Platform, and which release these features were implemented. Features and behaviors related to the ActiveX control are available beginning with the release in which it is listed, and in all subsequent releases unless otherwise noted.

Agentry Mobile Platform 5.2.8

The following features were added and changes were made to the 5.2.8 service pack release of the Agentry Mobile Platform in relation to the ActiveX control functionality:

- The ability to pass objects and object collections to the ActiveX Control. This is supported with the addition of the following client-side API methods:
 - AgentryActiveXPropertyType
 - GetPropertyFromMappings
 - GetPropertyFromObject
 - GetPropertyType
 - PropertyAsString*
 - NextCollectionProperty
 - CollectionHasNextProperty
 - RewindCollection
- The ability to issue a command to the ActiveX Control. This is supported with the addition of the following:
 - New Action Step Type: **External Field Command**
 - New Expected ActiveX Control Method: AgentryExecuteCommand

** - The PropertyAsString method is a replacement for the client-side API method GetAgentryString, which has been deprecated. This method is still supported for backwards compatibility, but should not be used in new development. Where possible, it is recommended that existing implementations are modified to use the PropertyAsString method in place of GetAgentryString.*

Agentry Mobile Platform 5.1

The following features were added and changes were made to the 5.1 minor release of the Agentry Mobile Platform in relation to the ActiveX control functionality:

- Implementation of the Agentry Client-Side API to expose various aspects of the Agentry Client to the ActiveX control. Many of the following new features are supported with the implementation of this new API for the Agentry Client. The API itself is made available to the ActiveX control via the following:
 - IAgentryActiveXControlHost COM interface
 - AgentrySetActiveXControlHost - ActiveX control expected method (provides IAgentryActiveXControlHost COM interface to the ActiveX control)
- The ability to pass data from the Agentry Client to the ActiveX control, based on a request by the control. Support for this functionality is provided with the following additions:
 - Agentry Data List added to the **External Field - ActiveX Control** detail screen field edit type.
 - GetAgentryString - Agentry Client-Side API method

- Support for the ActiveX control to request an action be executed on the Agentry Client. This functionality is provided with the following additions:
 - Agentry Actions List added to the **External Field - ActiveX Control** detail screen field edit type.
 - `ExecuteAgentryAction` - Agentry Client-Side API method
- Support for the ActiveX control to notify the Agentry Client when its value changes or is fully entered. This functionality is supported with the following additions:
 - `ActiveXControlValueChanged` - Agentry Client-Side API method
 - `ActiveXControlValueEntered` - Agentry Client-Side API method
- Support for the Agentry Client to request values from the ActiveX Control via target paths. This functionality is supported with the following additions:
 - External Data List added to the **External Field - ActiveX Control** detail screen field edit type.
 - `AgentryGetSpecificValue` - ActiveX control expected method

Agentry Mobile Platform 5.0

The following features were added and changes were made to the 5.0 major release of the Agentry Mobile Platform in relation to the ActiveX control functionality:

- Support for the Agentry Test Script functionality, including recording and playback features provided in the Agentry Test Environment. This includes the following client-side API methods:
 - `AgentrySetScriptValue`
 - `AgentryGetScriptValue`

Agentry Mobile Platform 4.3

In the 4.3 minor release of the Agentry Mobile Platform, the ability to pass values read into the Agentry Client by the device's barcode scanner to the ActiveX was added. This feature is exposed in the External Field - ActiveX Control detail screen field type. This definition type was modified to include the Scanning attribute. When set to true, barcode values read in by the Agentry Client are passed to the ActiveX control referenced by the detail screen field.

Agentry Client ActiveX API Methods

The following methods are available within the Agentry Client and can be called from an ActiveX control installed to the same client device. This assumes the ActiveX control has been referenced and loaded by an External Field - ActiveX Control detail screen field within the mobile application running on the Agentry Client.

The information provided for each method includes its intended purpose and the description of the method parameters.

Included in these methods are those called to retrieve values from the mobile application, execute actions defined within the mobile application, and to notify the application that the value of the ActiveX control has changed or has been fully entered.

In order to retrieve data values from the application and to execute actions within it, these items must be listed in the External Field - ActiveX Control detail screen field definition within the application project. Only those values and actions listed as available are accessible to the ActiveX control at run time.

ActiveXControlValueChanged

This method should be called to notify the Agentry Client that the value of the ActiveX control has changed. The Agentry Client will evaluate any update rules currently in context for the detail screen. Note that this differs from the `ActiveXControlValueEntered` method, which should be called when the value has been completely entered in the ActiveX control. Rather, `ActiveXControlValueChanged` is called for each value change that Agentry Client should be aware of in order to process the changed value within update rules defined for other fields on the same detail screen.

Parameters

None

ActiveXControlValueEntered

This method should be called to notify the Agentry Client the value of the ActiveX control has been fully entered. The Agentry Client will evaluate any update rules currently in context for the detail screen, and will perform any additional operations based on the auto-next or auto-focus behaviors defined for the detail screen fields.

Parameters

None

ExecuteAgentryAction

This method can be called to execute an action on the Agentry Client. This action must be listed in the Actions list for the External Field – ActiveX Control detail screen field. This method blocks until:

- A wizard screen is displayed
- The Action completes execution
- The Action is canceled by an action step of type message
- The Agentry Client reports that it cannot execute the action

Note that the method will **not** wait for the completion of a wizard screen set. When such a screen set is displayed, the `ActionResult` parameter will contain a value of `Action_Pending`.

Parameters

- `ActionName` - Contains the definition name of the Action to be executed

- **ActionResult** - This value is set after the action is executed and indicates the status of the action execution. This will be one of the following enumerated values:
 - **Action_BackUp**: Reserved for future use.
 - **Action_Error**: Returned when the action could not be executed for any reason. Common causes for this return include if the action named is not one defined for the screen field, if another action is currently being executed on the Client, or if the defined target object for the action cannot be resolved.
 - **Action_Cancel**: Returned if the action is cancelled. This will only be returned if the action is canceled in an action step of type Message. This method does not block when wizard screens are displayed and therefore will not capture the cancellation of a wizard screen set by the user.
 - **Action_Pending**: Returned if the action executed successfully and displayed a wizard screen set.
 - **Action_Complete**: Returned if the Action executes and completes successfully without having displayed a wizard screen set.

GetPropertyFromMappings

This method retrieves the property named in the name parameter. The property is returned in the `property` parameter. This method can return any value listed in the Agentry Values of the External Field - ActiveX Control field within the mobile application. The name parameter is set to the name of the value as defined in the Agentry Values list. As of version 5.2.8 of the Agentry Mobile Platform, these properties can be of type collection and object in addition to the other property types support in previous versions.

Prototype

```
void GetPropertyFromMappings(BSTR name, VARIANT* property)
```

Parameters

- `name` - The name of the object or property to be retrieved, as defined in the External Field - ActiveX Control's Agentry Values list.
- `property` - The property retrieved by the method.

Return Value

None (*see the property parameter description*)

GetPropertyFromObject

This method returns the named property from the previously retrieved object. This method is called after `GetPropertyFromMappings`, with the `property` parameter to that method passed to `GetPropertyFromObject` as the `object` parameter. Note that this method will return an invalid result if the object parameter is provided any value other than an object.

The name of the property to be returned is provided as the definition name of that property definition within the application project. The property is returned in the `property` parameter.

Prototype

```
void GetPropertyFromObject
```

```
(VARIANT const object, BSTR propertyName, VARIANT* property)
```

Parameters

- `object` - The object from which the property is to be retrieved. This parameter should be the value returned in the `property` parameter of the `GetPropertyFromMappings` method when that parameter references an object; or from a previous call to `GetPropertyFromObject` when the property it returns is an object property.
- `propertyName` - The name of the property definition to be retrieved within the object. This is the definition name of the property within the Agentry application project.
- `property` - This parameter will be set to the property referenced by the `propertyName` parameter. The type will be set to `AXPT_Invalid` if the `propertyName` parameter contains a name not found in the object's properties.

Return Value

None (see *property parameter description*)

GetPropertyType

This method returns the property type of the item provided in the `property` parameter. Note that this can include objects if a single object instance is defined as a property to another object. To avoid confusion, a collection property containing objects will return the collection property type, not object.

The types returned by this method will be one of the values in the enumerated list `AgentryActiveXPropertyType`. See the description of this enumerated list for details of the values it defines.

Prototype

```
void GetPropertyType
```

```
(VARIANT const property, enum AgentryActiveXPropertyType* type)
```

Parameters

- `property` - The property for which the type is to be determined.

- `type` - The enumerated value of the property type, as defined in the `ActiveXPropertyType` enumerated list.

Return Value

None (*see type parameter description for method return*)

PropertyAsString

This method takes the `property` parameter and returns the value of the property it references as a string. This string is assigned to the `value` parameter. Providing an invalid property type, including an object or collection property, returns an empty string in the `value` parameter.

The `GetPropertyFromMappings` or `GetPropertyFromObject` methods must be called prior to calling `PropertyAsString`. The `property` parameter set by the `GetPropertyFromMappings` or `GetPropertyFromObject` methods should be passed as the `property` parameter to `PropertyAsString`.

Prototype

```
void PropertyAsString (VARIANT const property, BSTR* value)
```

Parameters

- `property` - The property for which the value is to be retrieved.
- `value` - The string representation of the property value.

Return Value

None (*see the value parameter description for this function's return value*)

NextCollectionProperty

This method returns the next member of the collection property referenced in the `collection` parameter. The method also updates the position pointer within the `collection` parameter to point to the next member of the collection property. The returned member is referenced in the `property` parameter to this method. If there is no next member, the type for this property is set to `AXPT_Invalid`. This can be checked using the `GetPropertyType` method, passing the `property` parameter to it.

The `GetMappedProperty` method should be called prior to this method. When the property returned in the `property` parameter is a collection, `NextCollectionProperty` is called to retrieve the members of that collection. When these members are object instances, the `GetPropertyFromObject` method is called subsequent to `NextCollectionProperty` to retrieve the property values found within the object instance.

Prototype

```
BOOL NextCollectionProperty (VARIANT* collection, VARIANT* property)
```

Parameters

- `collection` - This parameter is the reference to the property returned by the `GetMappedProperty` method when the property it returns is a collection.
- `property` - The next member of the collection property referenced by `collection`. This property type should always be checked with the `GetPropertyType` method before attempting to reference it to ensure it is not set to `AXPT_Invalid`. This type is returned when there is no next member in the collection.

Return Value

Boolean value indicating whether the position pointer references a valid collection member, or the end of the collection:

- `true` - Returned if the position pointer of the collection parameter points to a valid member of the collection.
- `false` - Returned if the position pointer of the collection parameter indicates the end of the collection has been reached.

CollectionHasNextProperty

This method returns `true` if the current position pointer within the `collection` parameter is pointing to a valid member of the collection; that is, if a call to `NextCollectionProperty` will return an actual instance from the collection. If the position pointer is at the end of the collection, this method returns `false`.

Prototype

```
BOOL CollectionHasNextProperty (VARIANT const collection)
```

Parameters

- `collection` - The collection property to be evaluated for a valid next member based on the collection's position pointer.

Return Value

Boolean value indicating whether the position pointer references a valid collection member, or the end of the collection:

- `true` - Returned if the position pointer of the collection parameter points to a valid member of the collection.

- `false` - Returned if the position pointer of the collection parameter indicates the end of the collection has been reached.

RewindCollection

This method resets the `collection` parameter's internal position pointer to the first member of the collection. The `NextCollectionProperty` method will return the first member of the collection property in the next subsequent call.

Prototype

```
void RewindCollection (VARIANT* collection)
```

Parameters

- `collection` - The collection whose internal position pointer should be reset to the first member of that collection.

Return Value

None

GetAgentryString

Note: This method has been deprecated with the 5.2.8 service pack release of the Agentry Mobile Platform. The method `PropertyAsString` should be used in its place. This method is supported for backwards compatibility only. New development should use the `PropertyAsString` method in all cases, as `GetAgentryString` may be removed at a future time.

This method can be called to access a value on the Agentry Client. This value must be defined as an Agentry Data item in the External Field - ActiveX Control detail screen field. The name of the value, as defined in the Agentry Data list, is passed to the methods `DataItem` parameter. The value of that item is returned in the `agentryString` parameter as a string value regardless of the property's data type within the mobile application.

Prototype

```
HRESULT GetAgentryString (BSTR DataItem, BSTR agentryString)
```

Return Value

The `HRESULT` return indicates the status of the method call.

Enumerated List: AgentryActiveXPropertyType

The following list contains the members of the enumerated list `AgentryActiveXPropertyType`, along with the corresponding property type in the

Agentry application project. One of these values is returned by the `GetPropertyType` method within the Agentry Client ActiveX API indicating the data type of the referenced property.

AgentryActiveXPropertyType Members

- `AXPT_Invalid` - Returned when the property parameter does not reference a valid property
- `AXPT_Collection` - Property is a collection property
- `AXPT_ComplexTableSelection` - Property is a complex table selection
- `AXPT_Boolean` - Property is a Boolean
- `AXPT_DataTableSelection` - Property is a data table selection
- `AXPT_Date` - Property is a date
- `AXPT_DateAndTime` - Property is a date and time
- `AXPT_DecimalNumber` - Property is a decimal
- `AXPT_Duration` - Property is a duration
- `AXPT_ExternalData` - Property is an external data
- `AXPT_Identifier` - Property is an identifier
- `AXPT_Image` - Property is an image
- `AXPT_IntegerNumber` - Property is an integral number
- `AXPT_Location` - Property is a GPS location
- `AXPT_Object` - Property is an object instance
- `AXPT_Signature` - Property is a signature
- `AXPT_String` - Property is a string
- `AXPT_Time` - Property is a time

Expected Methods Implemented in ActiveX Control

In order for the Agentry Client to interface with an ActiveX control, it is a requirement of that control that it implements certain methods with the proper prototypes. Following is a description of each of these methods, their prototypes, and when the method is called by the Agentry Client at run time.

It is important that each of these methods is implemented, even those that are provided for functionality not currently implemented. Those methods not expected to be used should include at least a stub implementation within the ActiveX control.

Currently the Agentry Client can directly integrate with ActiveX controls built using C++ and Visual Basic. Included in the following sections are two lists of method prototypes for each of the expected methods. The first is for C++ implementations, and second is for Visual Basic.

Note: See the section at the end of this technical bulletin on integrating ActiveX controls built on .NET.

ActiveX Expected Method Declarations - eMbedded Visual C++

When using the Visual C++ wizard to add methods, create the methods with the parameter and return types exactly the same as shown below. The method declarations in the control class header should appear identical to the following, with the exception of any word wrapping resulting from this publication.

```
afx_msg BOOL AgentryInitialize(LPCTSTR initialValue, LPCTSTR
formatString,
    BOOL readOnly, BOOL autoChangeFocus, long parentHwnd, VARIANT
messageIDs);
afx_msg void AgentryDestroy();
afx_msg void AgentryEnable(BOOL state);
afx_msg BSTR AgentryGetValue();
afx_msg void AgentrySetFocus(long type);
afx_msg void AgentryShow(BOOL state);
afx_msg void AgentryUpdateRuleEvaluated(LPCTSTR ruleResult);
afx_msg void AgentryUpdateScanData(BSTR scanResult);
afx_msg BSTR AgentryGetSpecificValue(LONG opcode, VARIANT
specificValue);
afx_msg BSTR AgentryGetScriptValue();
afx_msg void AgentrySetScriptValue(BSTR str);
afx_msg void AgentrySetActiveXControlHost(IUnknown* host);
afx_msg LONG AgentryExecuteCommand(LPCTSTR str);
```

The methods section of the IDL file for the Agentry ActiveX interface should appear identical to the following:

```
methods:
[id(1)] boolean AgentryInitialize(BSTR initialValue, BSTR
formatString,
    boolean readOnly, boolean autoChangeFocus, long parentHwnd,
    VARIANT messageIDs);
[id(2)] void AgentryDestroy();
[id(3)] void AgentryEnable(boolean state);
[id(4)] BSTR AgentryGetValue();
[id(5)] void AgentrySetFocus(long type);
[id(6)] void AgentryShow(boolean state);
[id(7)] void AgentryUpdateRuleEvaluated(BSTR ruleResult);
[id(8)] void AgentryUpdateScanData(BSTR scanResult);
[id(9)] BSTR AgentryGetSpecificValue(LONG opcode, VARIANT
specificValue);
[id(10)] BSTR AgentryGetScriptValue();
[id(11)] void AgentrySetScriptValue(BSTR str);
[id(12)] void AgentrySetActiveXControlHost(IUnknown* host);
[id(13)] LONG AgentryExecuteCommand(BSTR str)
```

ActiveX Expected Method Declarations - MS Visual Basic

The methods expected by the Agentry Client should be declared exactly as listed below, with the exception of any word wrapping resulting from this publication.

```
Public Function AgentryInitialize(initialValue As String,
    formatString As String, readOnly As Boolean, autoChangeFocus As
Boolean,
    parentHwnd As Long, VARIANT messageIDs) As BooleanPublic Function
AgentryDestroy()
Public Function AgentryEnable(state As Boolean)
Public Function AgentryGetValue() As String
Public Function AgentrySetFocus(focusType As Long)
Public Function AgentryShow(state As Boolean)
Public Function AgentryUpdateRuleEvaluated(ruleResult As String)
Public Function AgentryUpdateScanData(scanResult As String);
Public Function AgentryGetSpecificValue(opcode As Long,
    VARIANT specificValue);Public Function AgentryGetScriptValue();
Public Function AgentrySetScriptValue(str As String);
Public Function AgentrySetActiveXControlHost(IUnknown* host);
```

AgentryInitialize

This method initializes the ActiveX control. It is called by the Agentry Client immediately after the External Field - ActiveX Control detail screen field is created. If this method returns false, indicating the control failed to initialize, the Agentry Client will not display the ActiveX control.

Parameters

- `initialValue` - The value of the property targeted by the External Field - ActiveX Control field in the Agentry Client.
- `formatString` - The value of the **Format** attribute defined in the External Field - ActiveX Control field in the Agentry Client.
- `readOnly` - The value of the **Read Only** attribute defined in the External Field - ActiveX Control field in the Agentry Client. `true` indicates the field is defined to be read-only.
- `autoChangeFocus` - The value of the **Automatically change focus to next control** attribute of the External Field - ActiveX Control field in the Agentry Client. `true` indicates this attribute has been set.
- `parentHwnd` - The HWND that corresponds to the parent window of the ActiveX control The ActiveX control should use this to send messages to the Agentry Client.
- `messageIDs` - **NOTE:** *This value, while still provided, should be considered deprecated See the Agentry Client ActiveX API methods `ActiveXControlValueChanged` and `ActiveXControlValueEntered` for the current manner of performing these operations.* A safe array stored within a VARIANT. The safe array contains an array of long values that correspond to each message ID for each message that may be sent to the Agentry Client. Within C++ the array index begins at zero and within the Visual Basic the array index begins with one.
 - *First index position:* Send this message to the `parentHwnd` to notify the Agentry Client a value has changed within the control and it is time for the Agentry Client to

evaluate the field update rules and enable rules defined for all fields on the current detail screen.

- *Second index position:* Sends this message to the `parentHwnd` to notify the Agentry Client a value has been completely entered in the control and it is time to automatically change focus to the next control.

Return Value

- `true` - This method should be implemented to return `true` when the ActiveX control has been successfully initialized.
- `false` - This method should be implemented to return `false` when the ActiveX control has failed to initialize. The Agentry Client will not display the control on the screen and will not call any other methods within the ActiveX control.

AgentrySetActiveXControlHost

This method provides the pointer to the `IAgentryActiveXControlHost` object to the ActiveX control. This pointer is passed over as an `IUnknown` pointer for the control host interface and should be queried to obtain the `IAgentryActiveXControlHost` object. This object provides the interface to the Agentry Client. It contains the methods that make up the Agentry Client-Side ActiveX API.

Parameters

- `host` - `IUnknown` pointer to the `IAgentryActiveXControlHost` object. Query this pointer to obtain the control host object.

Return Value

This method should be implemented with a void return.

AgentryDestroy

This method is called by the Agentry Client just before the External Field - ActiveX Control detail screen field is destroyed. This method should be implemented to perform any cleanup that may be necessary before the control is deleted.

Parameters

None

AgentryGetValue

This method is called by the Agentry Client to retrieve the current value of the ActiveX Control. This method is called by the Agentry Client either when the user advances past the screen displaying the External Field - ActiveX Control field in order to obtain the value to set to the target property of the field definition; or when a rule is evaluated by the Agentry Client that references the External Field - ActiveX Control.

Parameters

None

Return Value

- **BSTR** - String to be returned to the Agentry Client as the ActiveX control's value.

AgentrySetFocus

This method is called by the Agentry Client when the focus is set to the External Field - ActiveX Control detail screen field due to one of the following events:

- **Auto Focus:** The parent screen has just been displayed, that screen's **Focus Field** attribute is defined as Auto, and the External Field - ActiveX Control detail screen field is in the position to receive the focus.
- **Initial Focus:** The parent screen has just been displayed and that screen's **Focus Field** attribute is defined to set the focus explicitly to the External Field - ActiveX Control detail screen field.
- **Auto Change Focus:** The previous detail screen field is defined to automatically change focus to the next field, the External Field - ActiveX Control field is the next field, and the user has just entered a value in the previous field.
- **OS Focus:** The OS has sent a message that the External Field - ActiveX Control should receive the focus. This can occur when the user tabs to the field, selects the field's hot key, and other similar situations.

The value of the type parameter to this method indicates which of the above is the reason for the External Field - ActiveX Control to have received the focus should it be necessary to perform different processing based on the focus event.

Parameters

- **type** - The value indicating why the field has received the focus. These are numeric values corresponding to one of the above described events:
 - **Auto Focus:** 1
 - **Initial Focus:** 2
 - **Auto Change Focus:** 3
 - **OS Focus:** 4

Return Value

None

AgentryGetSpecificValue

This method is called by the Agentry Client to retrieve a value by name from the ActiveX control. The name passed will be one of those values contained in the External Field - Active X Control definition's External Data list. This method should be implemented to receive any of

the names as defined in the field definition, and to return the appropriate string value represented by that value name.

The External Data values listed in the External Field - ActiveX Control field's definition are available for reference in target paths within the Agentry application project. This method is called by the Agentry Client whenever one of these values is so referenced.

Parameters

- `opcode` - deprecated value that should be a part of the method's prototype but not used within the method's implementation.
- `specificValue` - The string name of the value to be returned by the method, as listed in the External Field - ActiveX Control's External Data list.

Return Value

- `BSTR` - The string representation of the named value requested by the Agentry Client.

AgentryUpdateScanData

This method is called by the Agentry Client immediately after a value has been scanned in by the client device for the External Field - ActiveX Control screen field. This method passes the scanned value to the ActiveX control in the `scanResult` parameter.

Parameters

- `scanResult` - This is the string value of the barcode value scanned in on the client device for the External Field - ActiveX Control screen field.

AgentryEnable

This method is called by the Agentry Client immediately after the enable rule for the External Field - ActiveX Control field has been evaluated. The state parameter to this method accepts the result of the enable rule's evaluation, which is a Boolean value. This method should be implemented to perform whatever processing may be necessary when the field is enabled and when it is disabled.

Parameters

- `state` - This parameter contains the Boolean value of the enable rule's return. `true` indicates the field is enabled, `false` indicates it has been disabled.

AgentryShow

This method is called by the Agentry Client immediately after the Hidden Rule is evaluated for the External Field - ActiveX Control field. This rule returns a Boolean value indicating whether or not the field should be displayed or hidden on the screen. The `state` parameter to this method indicates whether or not the field is shown.

Note that the Hidden Rule evaluated by the Agentry Client returns `true` when the field should be hidden, and `false` when it should be displayed. The value passed to the `AgentryShow` method is the inverse of the rule's return, meaning a state parameter value of `true` indicates the field is displayed, and `false` indicates it is hidden on the Client.

Parameters

- `state` - This parameter indicates whether the field is shown on the client screen. `true` indicates the field is currently displayed, `false` indicates it has been hidden.

AgentryUpdateRuleEvaluated

This method is called by the Agentry Client immediately after the update rule for the External Field - ActiveX Control detail screen field has been evaluated. The return from the field's update rule is passed to this method in the `ruleResult` parameter. The `AgentryUpdateRuleEvaluated` method should be implemented to process this value as the one currently displayed in the field on the client.

Parameters

- `ruleResult` - The string result of the External Field - ActiveX Control's update rule evaluation.

AgentryGetScriptValue

This method is provided exclusively for support of the Agentry Test Script functionality available in the Agentry Test Environment. This method is called by the Agentry Test Script Recorder when a `<field-expect>` method is recorded for the test script. It is also called during script playback when the `<field-expect>` element is processed for an External Field - ActiveX Control detail screen field. The method takes no parameters and is expected to return the value of the ActiveX control to be evaluated by the `<field-expect>` element.

Parameters

None

Return Value

The value of the ActiveX control as a string (BSTR) to be provided to the test script currently being recorded or executed by the Agentry Test Environment.

AgentrySetScriptValue

This method is provided exclusively to support the Agentry Test Script functionality available within the Agentry Test Environment. This method is called by the Agentry Test Environment during script playback when a `<field-set>` element is executed. The value for the ActiveX control is passed to this method's `str` parameter. The

`AgentrySetScriptValue` method should be implemented to process this parameter value such that it is set as the current value of the control as entered by a user.

Parameters

- `str` - The value to be set as the current value of the ActiveX control, provided as a string.

Return Value

None

Agentry Client API for External Processes Technical Overview

The Agentry Client API for external processes provides four methods that may be called by an external process to request information and data from, and to invoke transactions and execute actions on the Agentry Client. In order to use this API the external process must be built using the resources provided by the Agentry Client SDK for this API. The resources provided were built and are maintained using Visual Studio 2008 in the Visual C++ language. These same tools must be used to build the external process that is to make calls into the API.

The Agentry Client API for external processes does not include any corresponding controls or other similar components within the Agentry Client. It is limited to the methods made available to external processes to call.

Each of these methods includes a parameter containing the Agentry Client context object. This object is provided by the `AgentryInitialilze` method, which must be called prior to calling any of the other methods, and this object is then passed to each of the other methods when called.

Retrieving Data from the Agentry Client

Data is returned to the external process via the `EvaluateAgentryRule` method within the Agentry Client API. The rule to be evaluated and the module in which it has been defined are passed to the method parameters, along with a string variable in which the return value of the rule is provided. Using this method rules can be called within the Agentry Client from the external process to retrieve values from the Client. The returned values can be calculated or conditional values based on the structure of the rule definition, or they can simply be property values or other similar data items, again based on the rule structure.

Executing Actions on the Agentry Client

Actions can be executed by the external process via the `ExecuteAgentryAction` method within the Agentry Client API. The action to be executed and the module in which it has been defined are passed to the method. The method returns a Boolean indicator of success or failure to execute the action.

Transaction Processing on the Agentry Client from External Processes

Edit transactions can be instantiated, properties within them populated with values, and subsequently applied on the Agentry Client as a result of a request from an external process via

the `ExecuteAgentryTransaction` method. The edit transaction to be processed, the module in which it is defined, and values for one or more of its properties are passed to the method as parameters. Not all properties within the transaction need to be populated by the method call, and any not provided are initialized according to the property definitions just as if the transaction were instantiated via standard Agentry Client processing.

AgentryInitialize

The `AgentryInitialize` method is called to initialize a pointer to an `AgentryClientContext` object. This pointer is a required parameter to all other methods within the Agentry Client API for external processes. The single parameter to this method is a pointer to an object pointer of type `AgentryClientContext`. The pointer should be declared prior to calling the method and initialized to `NULL`. The address of the `AgentryClientContext` object is provided with the call to the `AgentryInitialize` method.

This method must be called when the external process is executed. The handle it returns (`AgentryClientContext` object pointer) should be preserved and passed to any subsequent Agentry Client API for external processes method calls. This handle should be passed to the `AgentryUnInitialize` method as a part of the external processes's shutdown procedures. See information provided on this method for details.

Prototype

```
bool AgentryInitialize(AgentryClientContext** ppCtx)
```

Parameters

- `ppCtx` - The `AgentryClientContext` object initialized by this method and passed to all other method calls within this API. A `NULL` pointer should initially be created to such an object, and the pointer should then be passed to this method, as in:

```
AgentryClientContext *ctx = NULL;  
AgentryInitialize(&ctx);
```

Return Value

The Boolean return value indicates whether or not the `AgentryClientContext` object was successfully initialized. The method returns `false` in the event of failure, which can occur if the Agentry Client is not currently running, as well as under other conditions. The return from this method should always be checked prior to using the `AgentryClientContext` object pointer it initializes. The external process should include processing to account for a false return indicating a failed initialization of this handle.

AgentryUnInitialize

the `AgentryUnInitialize` method is provided to allow the `AgentryClientContext` handle object to be properly cleaned up when the external process is exiting. this method should be called as a part of the processes's shutdown routines. It's only parameter is the handle, which should be the same as the one passed to a previous call to the `AgentryInitialize` method.

Prototype

```
bool AgentryUnInitialize (AgentryClientContext* pCtx)
```

Parameters

- `AgentryClientContext` - This parameter is passed to the method so that the handle for the `AgentryClientContext` object can be properly cleaned up when the external process is shutting down and the handle is no longer needed.

Return Value

The Boolean return value from the method indicates the success or failure of the uninitialize processing.

EvaluateAgentryRule

The `EvaluateAgentryRule` method can be called by the external process to request a named rule be evaluated by the Agentry Client. Included in the parameters to this method are the internal names of the module in which the rule is contained and the name of the rule to be evaluated. Also included are the `AgentryContext` and a string parameter in which the return value of the rule will be captured.

Rules evaluated by the `EvaluateAgentryRule` method are evaluated in the context of the module `MainObject` of the same module in which the rule is defined. Any rule in the module may be evaluated via this method, with the rule's return value provided as a string. This value can then be converted to other data types as needed within the external process.

Prototype

```
bool EvaluateAgentryRule (AgentryContext* pCtx,
                        const std::tstring& ModuleName,
                        const std::tstring& RuleName,
                        std::tstring& Value)
```

Parameters

- `pCtx` - Pointer to the `AgentryContext` object returned by a call made to the `AgentryInitialize()` method.
- `ModuleName` - The name of the module definition within the Agentry application project in which the rule to be evaluated is defined.
- `RuleName` - The name of the rule to be evaluated and whose return value is to be captured in the `Value` parameter.
- `Value` - Reference to a string value within the external process in which the return value of the rule definition will be contained. Regardless of the rule context or structure, the return value is always provided as a string value and can be cast to other data types within the external process.

Return Value

The Boolean return of this method indicates whether or not the rule was found and evaluated. If this fails for any reason the function returns false and the value of the `Value` parameter is a null string. The return value should always be checked before attempting to use the `Value` parameter and the external process should include logic to account for a failed rule evaluation.

ExecuteAgentryAction

The `ExecuteAgentryAction` method is called to request the Agentry Client execute an action. In addition to the `AgentryContext`, the method takes parameters specifying the name of the module in which the action to be executed is defined, as well as the name of the action itself. The action is always executed in the context of the module `MainObject`. The action being executed, therefore, must be defined for the `MainObject` or for no object. SubAction steps executing actions for other objects can be defined within the action executed by the method should it be necessary to execute an action for a different object type.

Actions may not be executed immediately under certain conditions; specifically, if another action is currently being executed. In such cases the action is queued by the Agentry Client to be executed as soon as it is able. The method will return true in such a case and the action will be executed when the first opportunity arises. The Agentry Client contains only a single action queue in which all queued actions are stored until executed. The other primary situation in which actions can be queued relates to Push Actions. Actions are executed from this queue in a first in-first out order.

Actions will not be executed and the method will return false if the Agentry Client is currently running, but the user has not yet completed the login process successfully, e.g., the login screen is currently displayed, the server selection screen is displayed, etc.; or if the named action or module cannot be found within the business logic currently running on the Agentry Client.

Prototype

```
bool
ExecuteAgentryAction(
    AgentryContext* pCtx,
    const std::tstring& ModuleName,
    const std::tstring&
    ActionName)
```

Parameters

- `pCtx` - Pointer to the `AgentryContext` object returned by a call made to the `AgentryInitialize()` method.
- `ModuleName` - The name of the module definition within the Agentry application project in which the action to be executed is defined.
- `ActionName` - The name of the action to be executed by the Agentry Client.

Return Value

The Boolean return of this method indicates whether or not the action was found and either executed or placed in the pending actions queue to be executed when possible. If this fails for any reason the function returns false and the named action will not be executed by the Agentry Client. The return value should always be checked and the external process should include logic to account for a failed action execution.

ExecuteAgentryTransaction

The `ExecuteAgentryTransaction` method is called by the external process to request an edit transaction be instantiated and applied, i.e., to be processed, by the Agentry Client. In addition to the `AgentryContext` object, the method takes the name of the module in which the transaction is defined, the name of the transaction itself, and a reference to an `AgentryPropertyVector` containing the transaction property values to be set within the transaction.

The transaction to be processed must be an edit transaction and must be defined for the module `MainObject`, as it is instantiated in the context of that object. Add and delete transactions are not supported.

When a transaction is processed via as a result of a call to this method, the properties of that transaction are first initialized according to the initial value attributes of those properties, with the exception of “Rule - After Data Entry.” Next, any values passed to the method call are copied to the transaction properties, which will replace any initialization values that may be present. The transaction is then processed by the Agentry Client. Property values are then set for any properties which are initialized to “Rule - After data entry.” The current value of such properties are overwritten with the value returned by the rule. Finally the transaction is applied, which includes setting the values of the object properties targeted by the transaction properties, and the transaction itself is saved to the client device as a pending transaction.

Prototype

```
bool
ExecuteAgentryTransaction(
    AgentryContext* pCtx,
    const std::tstring&
ModuleName,
    const std::tstring&
TransactionName,
    const
AgentryPropertyVector& properties)
```

Parameters

- `pCtx` - Pointer to the `AgentryContext` object returned by a call made to the `AgentryInitialize()` method.
- `ModuleName` - The name of the module definition within the Agentry application project in which the transaction to be processed is defined.
- `TransactionName` - The name of the transaction to be processed by the Agentry Client.
- `properties` - Reference to an `AgentryPropertyVector` containing the property values to be set when the transaction is instantiated. See the section on the API data types for more details on `AgentryPropertyVectors`.

Return Value

The Boolean return of this method indicates whether or not the transaction was found and processed. If this fails for any reason the function returns false and the named transaction will not be processed by the Agentry Client. The return value should always be checked and the external process should include logic to account for failed transaction processing.

Data Types Defined in the Agentry Client API for External Processes

Within the Agentry Client API for external processes there are certain data types defined: `AgentryContext`, which is an object obtained using the `AgentryInitialize()` method, and `AgentryPropertiesVector`, which is established via a type definition as a vector of `AgentryAttrPair` items. `AgentryAttrPair` is a standard pair of strings.

AgentryAttrPair and AgentryPropertiesVector

The `AgentryPropertiesVector` is provided to allow for property values of a transaction to be set by the external process and passed to the Agentry Client via the `ExecuteAgentryTransaction` method. This data type is declared in the include file `AgentryExternal.h`, which should be included in the project containing the external process logic.

This data type is declared by the following `typedef` statements:

```
typedef std::pair<std::tstring, std::tstring> AgentryAttrPair
typedef std::vector<AgentryAttrPair> AgentryPropertiesVector
```

The first typedef statement creates a standard pair of string values identified as `AgentryAttrPair`. This type is then the member type for the vector declared by the second statement, which is identified as the type `AgentryPropertiesVector`.

Within the elements of an `AgentryAttrPair` are stored the name and value of a property within the transaction definition, with the first element of the pair containing the property definition name, and the second containing the value. All values are stored as strings within a given pair and the second element is converted, when necessary, by the Agentry Client to the property data type before assigning the value to the specified property within the transaction. This behavior negates the need to perform any data type conversion within the external process as it would relate to property data types.

AgentryClientContext

This object type is internal to the Agentry Client. A declaration is provided for this object in the `AgentryExternal.h` header file. A handle to this object is provided by the `AgentryInitialize` method, which should be called by the external process during startup. The handle is then a required parameter to all API method calls. The handle should be passed to the `AgentryUnInitialize` method by the external process during shutdown.

Agentry Language Reference

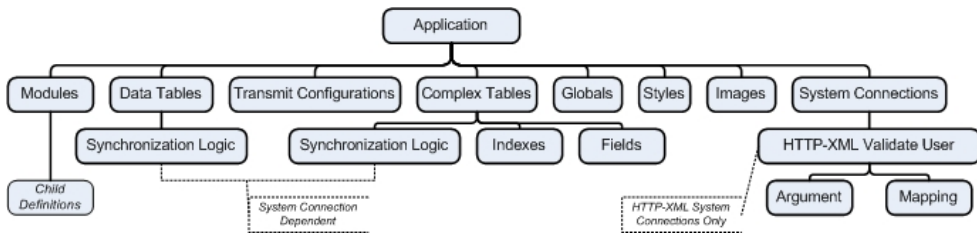
Use the Agentry Language Reference to learn about the following.

Application Level Definitions Overview

Within the application project structure in Agentry the definitions at the application level are at the top of the hierarchy. These definitions affect the application as a whole. The definitions that are direct children of the application are those that affect communications behavior, globally available constant values used for configuration and other purposes, and also include data storage on the client in the form of tables and records accessible to the entire application. In addition are the definitions that can affect the appearance of the user interface.

The application itself is represented as a definition type within the application project. Within a project there is only one application definition. The child definitions to the application are then referred to as the application-level definitions. Regardless of functionality, most of the application-level definitions will be used in a given application.

Following is the structure of the application level definitions within the application project. For all definitions in this graphic the child definitions are also shown, with the exception of the module. Modules are a robust definition type and the structure of the module is provided with the module-level discussions.



As illustrated in this graphic, the child definitions to data tables and complex tables related to synchronization are dependent on the type of system connection for which those definitions were created. The synchronization logic will be encapsulated in the language or methodology matching that back end system type.

As denoted in the illustration the system connections of type HTTP-XML include child definitions related to user validation. The user validation request is sent according to these definitions, including arguments to that request. Responses from the request are then mapped to the data components of the mobile application.

The module definition type contains numerous child definitions not represented in this graphic. These are illustrated in the sections covering the module-level definition types.

In general when working within the Agentry Editor to either develop an application or to modify an existing one, the application-level definitions dictate and control aspects of the application behavior overall, rather than within a given module or lower-level of granularity.

Application Definition

The application definition type represents the mobile application within the project and all definitions for the application are its descendents. The attributes of the application definition are those that affect application-level behaviors. These include the application name and version, the appearance of built-in Client screens, login and password settings, application-wide screen and user interface behaviors, and other similar items. The application definition is the single root definition in the application hierarchy and as such has no parent definition.

When a new application project is created in the Agentry Editor, an application definition is automatically created. Its attributes are set to defaults that should be reviewed thoroughly early in the development process. These attributes can affect security, appearance, and numerous other behaviors of the application.

General Setting Attributes

The general setting attributes for the application provide project name, the application's display name, and a version value.

- **Name:** This is the internal name of the application. This value is used for certain checks during publish.
- **Display Name:** This is the name of the application as displayed on the Agentry Client. This value appears in the title bar of the application and in the About dialog displayed from

the Agentry Client's Help menu. For any newly created application project this is set to a default of Agentry and should be changed.

- **Version:** This value is also displayed to the user in the About screen. This value is not related in any way to the application's publish version number. This attribute value is provided for branding purposes only and will not impact any aspect of the application's behavior. Typically this reflects the application's release version.

Application Setting Attributes

Table Settings - These attributes affect the behavior of data synchronization related to the two table definition types, data tables and complex tables.

- **Check Data Tables:** Specifies how often the application will check for new or changed data for the application's data tables. The choices are "Every Transmission", "Once per day", and "Once per week". In between the specified intervals, no synchronization components of the application's data tables will be processed during a transmit. A published change to the data table definition will override this attribute, forcing a reload of the data table during the next Agentry Client transmit.
- **Check Complex Tables:** Specifies how often the application will check for new or changed data for the application's complex tables. The choices are "Every Transmission", "Once per day", and "Once per week". In between the specified intervals, no synchronization components of the application's complex tables will be processed during a transmit. A published change to the complex table definition will override this attribute, forcing a reload of the complex table during the next Agentry Client transmit.
- **User Request:** This setting specifies whether or not users can explicitly check for changes to the application's complex tables and data tables. This is a means of providing users with a manual override for the **Check Data Tables** and **Check Complex Tables** attribute settings. When **User Request** is enabled, users will be able to force the synchronization process to include the processing of the data table and complex table definitions' synchronization components. Users will be able to force this behavior by selecting the menu item "Check for Table Updates" in the Agentry Client's Off-Line menu. This attribute will have no effect when the **Check Data Tables** and **Check Complex Tables** attributes are set to "Every Transmission".

Client Settings - Client Settings affect various behaviors of the client application at runtime.

- **When Exiting Client:** This enables a warning message displayed to the user if there are pending transactions stored on the Agentry Client when they exit the application.
- **Prompt on User Change:** This enables a prompt when a user change occurs, informing the user that a synchronization with the Agentry Server must take place to change users and gives the user the option to cancel the user change. If this is disabled, the synchronization will still occur to complete a user change, but no prompt will be displayed.
- **Module Menu Item:** This attribute specifies whether or not the menu item for the current module is enabled or disabled in the Agentry Client's View menu at runtime. Selecting the current module from the View menu will return the user to the module's main screen set, regardless of where they may be in the navigation. When disabled, the menu item for the

current module is disabled. Users can always select other module items in this menu for applications with multiple modules regardless of this setting.

- **Synchronize Clocks:** This attribute specifies whether or not the system time on the client device will be reset to that of Agentry Server's host system time during each transmit. Note that this time is not the time of the back end system with which the Agentry Server communicates. It is the system time as reported by the operating system of the Agentry Server's host system. This is typically disabled in deployments involving multiple time zones.
- **Screen Size:** This setting specifies the size of all screens displayed to the user on the Agentry Client. This attribute will only effect Agentry Client applications running on a Windows PC platform capable of full VGA screen resolution. The screen sizes available for this setting range from 240 x 320 (1/4 VGA) to 1366 x 768. The Screen Size value will override the screen size attribute for all platform definitions. There is also the available setting "Allow Resize". If any selection other than Allow Resize is made, users will not be able to resize Agentry Client screens. The screen size for all mobile devices, including smart phones tablets, and other devices, is always full resolution of those devices and users can never resize the screens.
- **Battery Status:** For mobile devices, the status of the battery can be displayed on the Agentry Client. This will appear in either the upper or lower portion of the screen, depending on the device. Note that this setting has no effect on the Windows PC builds of the Agentry Client.
- **WinCE Navigation:** This setting enables support for the arrow keys of a device's hardware keyboard. When a user clicks one of the arrow keys, the focus of the screen will be changed to the next or previous control on that screen. This attribute has no effect on the Windows PC builds of the Agentry Client, where full keyboard navigation is always enabled.
- **Scan Trigger Shortcut Key:** For devices equipped with a scanner, this attribute allows for the specification of a shortcut key to activate the scanner. This key will be universal to the application, and will activate both socket and built-in scanners. This attribute will have no affect on Agentry Clients running on devices not equipped with a scanner.
- **Voice Support:** Enables voice support for devices that support this feature.
- **Title Bar Buttons:** This attribute specifies whether or not the close buttons (either an X or an OK button) are displayed on the title bar of screens within the mobile application. Due to the behavior of Pocket PC devices, it is recommended that these buttons not be displayed and that actions are defined within the Agentry application project to close screen sets within the application, and that users close the application itself using either the **File | Exit** menu item, or through an action containing an Exit Application action step. Note that on Pocket PC devices, screens closed with the title bar's OK button are not destroyed, but rather only moved to the end of the "Z" order, hiding them from view. Applications closed with the X button of the title bar are not actually exited. Any defined behaviors for exiting an application will not be exhibited. Furthermore, the application itself will still be running. The behaviors described here are not present on a Windows PC platform.
- **Theme Selection:** This attribute specifies whether or not the Theme menu item within the Preference menu of the Agentry Client is displayed. When true (checked) the user can

change the Agentry Client theme using this menu. When false, the user cannot change the Agentry Client theme and the theme displayed is always the one selected in the **Default Theme** attribute. Allowing user's to select a different theme can have unexpected impact on the UI of the Agentry Client if styles are defined and in use.

- **Default Theme:** The theme selected is the default theme displayed on the Agentry Client at run-time. If **Theme Selection** is disabled, the selected Default Theme is always displayed and the user cannot change the theme selection. If **Theme Selection** is enabled, the selected **Default Theme** will be the initially applied theme on the Agentry Client, but the user can select a different theme at any time.
- **Win32 Buttons - Use large buttons:** This attribute specifies whether or not to use large sized buttons. This attribute affects only the Windows PC platform types. When true, screen buttons are displayed in a large size, generally intended for touch screen support. This setting affects button definitions for list and detail screens. Built-in buttons, such as ellipses buttons, icon buttons, and similar controls are not affected. Note that large buttons are not displayed in the Agentry Editor's layout view or visual screen editor for screen definitions. They are displayed in the Agentry Test Environment when the selected platform is Windows.

Application Styles Attributes

The attributes listed in the Application Styles tab define how styles are to be applied to all components of the application's user interface. These style settings may be overridden at lower levels in the application's structure. The style settings here also impact what styles are applied to the Agentry Client's built in screens and dialogs, such as those for complex table searches, the transmit dialog, and others. For all style attributes, the option "--Default--" will default to the operating system's default font and color options.

Screen Styles

- **Tabs:** The style to apply to the tab controls representing each screen within an object screen set. This attribute has no effect on screens within a transaction or fetch screen set.
- **Buttons:** The style to apply to all button definitions on all application screens. This includes buttons displayed on built-in Agentry Client screens as well as buttons within screen definitions.
- **Focused Buttons:** The style to apply to the button that currently has the focus. This includes buttons displayed on built-in Agentry Client screens as well as buttons within screen definitions.

Detail Screen Styles

- **Screen:** The style to apply to all detail screens defined within the application. This will affect all portions of the screen not displaying a field or button.
- **Fields:** The style to apply to all fields displayed on a detail screen.
- **Focused Fields:** The style to apply to the detail screen field that currently has the focus.
- **Read-Only Fields:** The style to apply to a detail screen field defined to be read-only. If not specified, the Fields style is applied.

- **Hyperlinks:** The style to apply to detail screen field labels defined to be hyperlinks.
- **In Progress Edit Screens:** The style to apply to screens in which changes are currently being made and have not yet been applied. This affects screens displayed in List Tile View and Tile Edit fields.

List Screen Styles

- **Screen:** The style to apply to all list screens as a whole. This will affect all portions of the screen not displaying a list, header label, detail pane, or button.
- **Header Label:** The style to apply to all list screen header labels. If no header label is defined this attribute has no effect on the screen.
- **Column Labels:** The style to apply to the column labels on the screen's list control. This style is applied to the column labels of any screen containing a list control, including both built-in Agentry Client screens as well as list screen definitions, and list view field definitions.
- **Rows:** The style to apply to all rows on a list screen. The **Hyperlinks** optional style will override the **Rows** style for cells with hyperlinks. This style is applied to the list items of any screen containing a list control, including built-in Agentry Client screens, list screen definitions, and list view field definitions.
- **Alternate Rows:** The style to apply to every other row in a list, beginning with the second row. The **Hyperlinks** optional style will override the **Alternate Rows** style for every other row where there are cells containing hyperlinks.
- **Highlight Rows:** The style to apply to a row for the purposes of drawing attention to that row. This style attribute should not be set at the application level. The platform and list screen definition types both contain a Highlight Rows attribute that should be used.
- **Selected Rows:** The style to apply to the row or rows currently selected by the user in the list control. The optional **Hyperlink** style will be applied to any cells within the selected row containing a hyperlink.
- **Selected No Focus Row:** The style to apply to the row or rows currently selected by the user in the list control, when the input focus is set to some control other than the list control. The optional **Hyperlink** style will be applied to any cells within the selected row containing a hyperlink.
- **Detail Pane:** The style to apply to both the foreground (text) and background of a list screen's detail pane. If no detail pane is defined this attribute has no effect on the screen.

Application Images

- **About Box Dialog Image:** This attribute specifies an image definition to display in the Agentry Clients' about box.
- **Login Dialog Image:** This attribute specifies an image definition to display in the Agentry Clients' login screen.
- **Module Menu Dialog Image:** This attribute specifies an image to display in the Agentry Clients' module selection dialog. This dialog is displayed after users log into Agentry Client applications with multiple modules. Note that within this same module selection dialog, each module may also display an image. The image defined at the application level

is separate from the module images. For applications with a single module, this attribute has no effect as the module selection dialog is never displayed.

Application Security Attributes

The attributes in this section control overall security related to items such as failed login attempts, locking the user out of the client application after failing validation, user ID and password rules, and idle timeout settings.

User Settings

- **Login:** This attribute specifies whether or not users are required to perform a transmit every time the Agentry Client application is started and the user logs in. Note that setting this option to true requires an available network connection for the client device and users will be required to perform a successful transmit before the user can use the client application.
- **Login Attempts:** This attribute specifies the maximum number of failed login attempts that may occur before locking the user out of the application. “Unlimited” will never lock the user out. A failed login will occur if the user enters an incorrect password for the entered User ID when not required to connect to the Server during login; or if the user fails user authentication when the Login attribute is set to true and the number of failed attempts exceeds the value entered here. The resulting behavior of locking out a user is defined in the Lockout Level attribute described below.
- **Lockout Level:** This attribute specifies the action to take when a user is to be locked out of the Agentry Client. This may occur as a result of exceeding the defined **Login Attempts**, or based on failed attempts to authenticate the user against the back end system. The four possible settings for this attribute are:
 - Critical: This lockout level specifies that the entire Agentry Client be reset. This includes the removal of all production data and all application data, as well as removing the stored user ID and password. Users will be required to log into the Agentry Client application and perform a successful transmit with the Agentry Server before being allowed to access the Agentry Client application.
 - Severe: The severe lockout level specifies that all module-level production data, i.e. object instances and pending transactions, be removed from the Agentry Client application. Complex table and data table records will not be removed. The user will be required to perform a successful transmit with the Agentry Server before being allowed to access the client application.
 - Medium: This lockout level specifies the Agentry Client will exit and the user will be required to log into the Agentry Client and perform a successful transmit with the Agentry Server before being allowed to access the client application. No data is removed from the application.
 - None: This setting indicates that no lockout behavior should take place. This setting will result in any lockout events being ignored by the Agentry Client.

- **Client Database will be encrypted:** When true, this attribute specifies the database in which all client data is stored on the client device, including both production data and application data, will be encrypted.

User ID

- **Case:** This attribute specifies the case in which the user ID should be entered and will be stored. The options are mixed case, uppercase, and lowercase. Note that mixed case does not require a mixed case user ID, but rather allows for variable case. User IDs may then be in all upper, all lower, or mixed case.
- **Scan User ID:** This attribute specifies whether or not users can enter user IDs via the device's barcode scanner. This attribute does not require the ID to be scanned, but only allows for the option. This attribute has no effect on Agentry Clients running on devices not equipped with a barcode scanner.

Password

- **Password Change:** This attribute specifies whether or not users can initiate password changes on the Agentry Client. When true, the users will be able to change the password based on responses from the back end system indicating their passwords are about to expire or have expired. Users are required to enter the old password and the new one to change passwords. Note that enabling this behavior requires the implementation of logic to process a password change for the user in the back end system.
- **Scan Password:** This attribute specifies whether or not users can enter passwords via the device's barcode scanner. This attribute does not require the password to be scanned, but only allows for the option. This attribute has no effect on Agentry Clients running on devices not equipped with a barcode scanner.

Idle Timeout

- **Timeout:** This attribute specifies whether or not to require users to re-enter their user ID and password if the device is left idle for a defined duration of time. The duration is set as a part of this attribute. Also an option is whether or not the user ID should be populated automatically.

Password Rules

- **Minimum Length:** This attribute specifies the minimum number of characters of the password entered on the Agentry Client. The minimum length must be at least 1 to enable the **First Character** attribute. This value must be at least 2 to enable the **Character Mix** attribute. This value must be equal to or less than the **Maximum Length** attribute. The default minimum is none, which does not require a password to be entered on the Agentry Client.
- **Maximum Length:** This attribute specifies the maximum number of characters of the password entered on the Agentry Client. This value must be equal to or greater than the **Minimum Length** attribute, or be set to default, which is no maximum length.

- **Password Case:** This attribute specifies the case in which the user's password is stored on the Agentry Client and will be sent to the Server. This may be set to Mixed Case, Lowercase Only, or Uppercase Only. Note that Mixed Case does not enforce a requirement of a mixed case password. Rather it merely specifies that the case of the password characters will not be changed from how they are entered by the user.
- **Character Mix:** This attribute requires the **Minimum Length** attribute to be set to at least 2. **Character Mix** requires passwords entered on the Agentry Client must contain at least one alphabetical character and one non-alphabetical character. Non-alphabetical characters exclude non-printable characters.
- **First Character:** This attribute requires the **Minimum Length** attribute to be set to at least 1. **First Character** specifies that the first character of the password must be an alphabetical character.
- **New vs. Old:** This attribute specifies that a new password entered by the users on the Agentry Clients must be different from the previous password. A difference is based on the change of at least one character from the previous password to the new one. This attribute may be impacted by the **Password Case** attribute. **Mixed Case** will treat the same letters in the old and new password as different if at least one letter is entered in a different case. For Uppercase Only and Lowercase Only **Password Case** settings, case is ignored and the same letters entered in a different case will not be treated as a different password.

Module

The module definition is a grouping of definitions providing functionality that logically belongs together. The module's attributes and child definitions define the majority of the behavior and functionality exhibited on the Agentry Client at runtime.

The modules of an application contain the functionality related to the user interface on the Agentry Client, data storage and structures, data synchronization, and data capture. The child definitions of a module also have access to all application-level definitions.

An application project must contain at least one module. When multiple modules are defined for an application, users will be required to select which module to work with when logging into the Agentry Client application. They will be able to switch from one module to another using the Agentry Client's View menu, which will list the defined display name for each module within the application.

The module's child definitions are primarily intended to work with other definitions within the same module. Cross-module functionality can be defined using actions within one module that may execute actions of another module within the same application.

Module Child Definitions

- **Action** - An action defines navigation and user interaction for the Agentry Client, bringing the other components of the Client's UI together.
- **Fetch** - A fetch defines how the Agentry Server synchronizes data for a target object collection by referencing the step definitions to perform this task.

- **Object** - An object definition encapsulates a business entity and its related data.
- **Push** - A push defines when it is necessary to push an object in real time from the back end system to the Agentry Client and how that object's data is retrieved.
- **Report** - A report defines a printed tabular report format for the contents of an object collection on the Agentry Client.
- **Rule** - A rule defines evaluation logic processed on the Agentry Client that returns a single value to the caller of the rule.
- **Screen Set** - The screen set is the main Client user interface definition and defines what definition type its child screens display.
- **Service Event** - A service event defines how the Agentry Server synchronizes data between two back end systems, usually based on a change or "event" occurring in one of the systems.
- **Step** - A step defines a piece of processing to be performed by the Agentry Server with a specific back end system.
- **Transaction** - A transaction definition defines what data is captured on the Client, how that data affects a target object instance on the Client, and how the captured data is processed by the Agentry Server.

Module Attributes

- **Name** - This is the unique name of the module. This value must be unique among all modules defined within the application.
- **Display Name** - This is the text displayed to the users on the Agentry Client application at runtime. This value appears in the Agentry Client's Module Selection Screen to represent the module and also appears in the View menu of the Agentry Client as a menu item.
- **Preserve Objects** - This attribute specifies whether or not the objects within the module will be preserved when a new user logs into the Agentry Client on the same device as a previous user. If checked, the objects will be preserved from one user to the next. If left unchecked, a user change will result in the objects being removed prior to synchronizing object data for the new user.
- **Image:** Specifies the image definition to associate with the module definition. This image is then displayed for the module in the Agentry Client's Module Selection Screen displayed after login for multi-module applications.
- **Successful Login Action:** Specifies an action defined within the module to be executed after a user successfully logs into the application. The action executed here targets the module main object. For multi-module applications where more than one module defines a Successful Login Action, the order in which those actions are executed is undefined.
- **Application Exit Action:** Specifies an action defined within the module to be executed just prior to exiting the application. The action executed here targets the module main object. For multi-module applications where more than one module defines an Application Exit Action, the order in which those actions are executed is undefined.

Data Table

A data table definition defines a set of records stored on the Client. Each record consists of two fields containing a key and value. A data table is intended to contain a small number of records (less than 100) that may be displayed to users in drop-down lists and other uses. A data table is defined at the application level and is available to all modules of the application. Its structure also defines how its data is synchronized.

The intended purpose of a data table is to provide short lists of records that can be created quickly and with little overhead related to maintaining the data. A data table has no built-in search support and if searching is necessary it is performed row-by-row (e.g., no binary or other search algorithms are employed).

As a part of its definition, the data table contains the components to synchronize data. This includes determining if new data is needed for the table as well as the processes to retrieve the records for the data table. The definition of a data table requires the selection of an existing system connection. The type of synchronization components a data table contains is based on the type of the selected system connection.

Though the synchronization components will differ in form and structure related to the type of back end system for which they are intended, they are required to always return two general categories of data to the Agentry Server. The first is a date and time value retrieved from the back end that indicates the last time when the data source for the data table was last modified in the back end system. The second is the actual data for the data table's records.

The date and time value is compared to a date and time value stored internally on the Agentry Client for each data table instance. This internal value is called the data table's last update value. This last update value indicates when the data table was downloaded to the Agentry Client. When the date and time retrieved from the back end is newer than the Agentry Client's last update value for the data table it is the indication that the records for the data table must be retrieved. The existing records on the Agentry Client will be deleted and replaced with the new data retrieved for the data table. This is an all-or-none operation and individual records cannot be selectively replaced.

The specifics of how the date and time values for the data tables are retrieved, and how the records are retrieved for the data table are provided in the discussions specific to each of the possible system connection types that may be selected for the data table definition when initially defined.

Data Table Attributes

The following attributes are applicable to all data tables, regardless of the system connection a given data table may be using.

- **Name:** This is the unique name of the data table. This value must be unique among all data tables defined for the application.

- **Display Name:** This is the default text displayed to the user on the Agentry Server for the data table.
- **Connection:** This is the system connection defined for the back end system containing the data source for the data table. This attribute is set when the table is initially created. It cannot be edited for an existing data table definition. The system connection to be used must exist prior to defining the data table.
- **Reload:** This attribute specifies whether or not the records of the data table should be reloaded when a user change occurs on the Agentry Client. When true, all records in the data table are deleted and completely reloaded during the first transmit of the new user. Otherwise the records will remain on the Agentry Client during the user change. This attribute should be set to true when the data table contains records that are user-specific.

SQL Data Table Synchronization Components

When a data table is defined to use a SQL Database system connection, the synchronization components include a Sync Query and a Data Query.

The Sync Query is expected to return a value identified as `LastUpdate`. This value should indicate the date and time the source table in the database was last modified. This value is then compared to the last update value for the data table provided from the Agentry Client. If the date and time value returned by the Sync Query is not newer than the one for the data table, no further processing for the data table occurs.

If the Sync Query `LastUpdate` value is newer than the Agentry Client's last update value for the data table, the Data Query is run. This query is expected to return all records for the data table, whether or not an individual record is different in the database. This query is expected to return two columns identified as `CODE` and `VAL` to the Agentry Server. The value of the `CODE` column must be unique within the return set provided by the Data Query.

Sync Query and Data Query Attributes

- **Sync Query - File:** Specifies the name and location of the text file (`.sql` extension) containing the SQL statement for the Sync Query.
- **Data Query - File:** Specifies the name and location of the text file (`.sql` extension) containing the SQL statement for the Data Query.

HTTP-XML Data Table Synchronization Components

When a data table is defined to use an HTTP-XML system connection, the synchronization component it contains is an HTTP request.

This request is a child definition to the data table. It can be defined to make a request to a specified URL and may use the request methods GET, POST, HEAD, or PUT. The HTTP request itself contains two types of child definitions: Request Arguments and Response Mappings.

The request arguments contain the data values passed as arguments to the back end system as a part of the request being made. The response mappings are defined using XPaths to retrieve

data from structured XML return values provided by the back end system as a result of the HTTP request. These mappings can include the back end system's last update value for the data table's data source, the data values for the records to be stored in the data table, and other types of data.

There is a single request made to synchronize the data table, with the value mapped to the `LastUpdate` value determining whether or not the data values returned should be used to replace the data table on the Agentry Client.

Data Table HTTP Request Child Definitions

Following is a list of the child definitions for the HTTP Request within an HTTP-XML data table.

- **Request Arguments:** This definition encapsulates an argument to be passed with the request to the back end system. These arguments can include data contained within the mobile application.
- **Response Mappings:** This definition encapsulates the XML data returned by the HTTP Request. The specific values are extracted from the XML return data using XPath's defined within each response mapping. The mapping "maps" the extracted values to values within the mobile application.

Data Table HTTP Request Attributes

The following attributes pertain the HTTP Data Request of a data table defined to use an HTTP-XML system connection.

- **Name:** The name of the request, set automatically based on the parent data table name. May be modified if desired.
- **URL:** The URL to the specific CGI or other process being called by the HTTP request to synchronize the data table.
- **Method:** The HTTP request method for the request. May be one of GET, POST, HEAD, or PUT.

HTTP Request Argument

The request argument definition encapsulates a data value to be passed from the mobile application to the process being called by the parent HTTP request definition. The request argument specifies the argument type, which may be CGI Argument, Cookie, HTTP Header, or XML Body. The request argument also specifies the data or data source within the mobile application to pass as the argument to the process or service being called by the parent HTTP request definition.

For a data table, the data value may be the user ID, the name of the data table, a fixed string whose value is defined as a constant within the request argument, or markup text. A given parent HTTP request may contain multiple request arguments. The order in which they are passed to the process or service when called is defined in the parent HTTP request's list of request arguments.

HTTP Request Argument Attributes

The attributes of a request argument depend in part on the data type of the argument (Data Type attribute). The following list makes note of those attributes specific to a certain argument data type.

- **Argument Type:** This attribute specifies the type of argument the definition contains. This may be one of CGI, Cookie, HTTP Header, or XML Body.
- **Name:** Alternately displayed as Argument Name, Cookie Name, Header Name, or Name depending on the **Argument Type** selection. This value must be unique among all request arguments defined within the same parent HTTP request definition.
- **Data Type:** Specifies the data value or source for the data value for the request argument. For a data table this may be the Table Name, User ID, Small or Large Markup, or a Fixed String value.
- **String:** This attribute is available only when the **Data Type** attribute is set to Fixed String. String contains the constant string value that is the request argument's data.
- **Markup Text:** This attribute is available only when the **Data Type** attribute is set to Small Markup or Large Markup. **Markup Text** contains the single line (Small Markup) of markup text or the contents of the Markup File (Large Markup) that is the data for the request argument.
- **Markup File:** This attribute is available only when the **Data Type** attribute is set to Large Markup. **Markup File** contains a reference to the text file containing the multi-line markup text. This file is displayed in the **Markup Text** field directly below the file name in the Editor and can be authored or modified directly in this multi-line field.

HTTP Request Response Mapping

The response mapping definition is a child to an HTTP request definition. This definition maps a data value returned from the process called by the HTTP request to a value within the mobile application. This value may be extracted from structured XML using XPath or XSL. It may also be a Cookie value or the HTTP Header.

For a data table the values may be mapped to the code or value fields in a data table record, an error message, the last update value to be compared against the data table's last update, a local data tag or local XML value, or to the user ID value that may be used in place of the ID entered to log into the Agentry Client.

HTTP Request Response Mapping Attributes

The response mapping attributes are in part dependent on the selection made in the Mapping Type attribute. Those specific to a certain type are denoted in the following list.

- **Mapping Type:** This attribute specifies the mapping type. This may be Cookie, HTTP Header, XPath Expression, or XML Transformation.
- **Base XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression or XSL Transformation. This attribute is optional and should be used when returning multiple instances of the same data element in the XML content. When a **Base**

XPath is defined for a response mapping, the same value will be set by default in the add wizard for subsequent response mappings within the same parent HTTP request definition.

- **XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression. This attribute contains the XPath markup to extract the desired value from structured XML data returned from the HTTP Request.
- **XSL:** This attribute is only available when the **Mapping Type** is set to XML Transformation. It contains the XSL expression to transform the XML data to be mapped to a value for the request.
- **Cookie Name:** This attribute is only available when the Mapping Type is set to Cookie. It contains the name of the cookie for the response mapping.
- **Header Name:** This attribute is only available when the Mapping Type is set to HTTP Header. It contains the name of the HTTP header for the response mapping.
- **Maps To:** This attribute specifies where the value extracted by the response mapping is stored in the mobile application. This may be one of the following values for a data table:
 - Data Table Key: This selection specifies the value extracted by the mapping contains the key or code field value for each data table record.
 - Data Table Value: This selection specifies the value extracted by the mapping contains the value field value for each data table record.
 - Error Message: This selection will map the data to error text displayed by the mobile application.
 - Last Update: This selection specifies the extracted value is a date and time indicating when the data table's source in the back end system was last modified. This value is compared against the internal last update value for the data table as provided by the Agentry Client.
 - Local String (<<local>>): This selection will create a local data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping. When selected, the attribute **String Name** will be available to name the new local data tag. This is the equivalent to calling the SDML function tag <<local ...>>.
 - Local XML (<<localXML>>): This selection will create a local XML data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping. When selected, the attribute **XML Name** will be available to name the new local data tag. This is the equivalent to calling the SDML function tag <<localXML ...>>.
 - User ID: This selection will map the value extracted by the response mapping to the user's ID. This value is the equivalent to the SDML data tag <<user.id>>. If a previous response mapping in any HTTP Request processed by the Agentry Server set the user ID, setting it here will override that value. This will then be the value available for all HTTP-XML system connection processing where the data tag <<user.id>> is referenced.

- **String Value:** This attribute is available when the map type is set to Local String. It contains the name of the data tag being created by the mapping. This is the name by which the data tag is referenced in subsequent references.
- **XML Name:** This attribute is available the map type is set to Local XML. It contains the name of the data tag being created by the mapping. This is the name by which the data tag is referenced in subsequent references.

Java Virtual Machine Data Table Synchronization Components

When a data table is defined to use a Java Virtual Machine system connection, its synchronization component is a single Java source file. This file contains the skeleton structure for a Java class that extends the Agentry Java API class `DataTable`. The name given to this class matches the name of the data table definition itself and should not be changed.

When the table is defined, the wizard for creating Java classes provided with the Eclipse Java perspective is used. This allows the developer to select the package to which the new class will be added. The source `.java` file created will then be stored according to the configuration of the project and package selected for the new class. Alternately an existing class in a package within the Java perspective may be selected. This class must extend the Agentry Java API class `DataTable`.

This skeleton class declaration includes three methods:

- The Constructor method.
- An override method for the data table `iterator()` method. This method is intended to contain the logic to retrieve the data from the back end system via the Java interface it provides. It is then intended to return an iterator for the data table object.
- An override method of `isOutOfDate()`. This method is expected to return true or false based on whether or not the data for the table is out of date. When true is returned by this method, the `iterator()` method will be called. When false, processing for the data table by the Agentry Server will be complete.

In versions of the Agentry Mobile Platform prior to 5.1, the source class was stored on the Agentry Server's file system. This behavior is deprecated in versions 5.1 and later. Agentry application projects created prior to this release are still supported and the Java logic will still be processed correctly. New data tables for Java Virtual Machine system connections should use the new procedure for defining the Java synchronization component.

Complex Table

The complex table definition defines a table of records containing multiple fields stored on the Agentry Client in a structured and searchable format. A complex table can contain large amounts of data with records numbering in the thousands. Included in the complex table are the fields for its records and indexes on fields to provide search functionality and structure to the overall data in the table. The complex table definition also defines how its data is synchronized.

The fields and indexes of a complex table define the structure of the records. A complex table must have a minimum of one index definition, which is the primary index. This index is defined for the field containing the unique identifier for each record. This field and index are then used during synchronization to identify records for addition, replacement, or removal.

The synchronization components of a complex table depend on the system connection the table definition uses for its data source. The synchronization components will match the system connection type. Independent of the system connection type, the synchronization logic for a complex table should account for retrieving all records when the table is in a rebuild state, retrieving just new or modified records during normal synchronization, as well as determining which records should be removed from the complex table.

The rebuild state of a complex table is set under various conditions. These include a published modification to the complex table definition, a user change occurring on the Agentry Client, and optionally based on the rebuild state being forced via administrator actions. During synchronization between the Agentry Client and Agentry Server, the Agentry Server will indicate if the complex table is in a rebuild state to the Agentry Client. The Agentry Client will remove all records for the complex table from the client device. The synchronization processing will retrieve all current records for the complex table and send them to the Agentry Client, rebuilding the table. This synchronization processing requires the developer to account for this situation.

When not in a rebuild state, the complex table can be updated selectively. Using an exchange data model for processing, only those records to be added, those records that need to be replaced, or those that need to be removed from the Agentry Client are retrieved by the Agentry Server from the back end system for the complex table. Any unchanged records will be left unmodified.

Complex Table Child Definitions

- **Field:** A complex table field definition defines a single piece of data for a complex table record, including its data type and size.
- **Index:** A complex table index definition orders the table's records by a field, making the table searchable by that field.

Complex Table Attributes

- **Name:** This is the unique name of the complex table. This value must be unique among all complex tables defined within the application.
- **Display Name:** This is the default text displayed to the user on the Agentry Client identifying the complex table.
- **Connection:** This is the system connection used by the complex table's synchronization components to synchronize the records of the complex table on the Agentry Client.
- **Reload:** This attribute specifies whether or not the records of the complex table should be fully reloaded when a user change occurs on the Agentry Client. When true, all records in the complex table are deleted and completely reloaded during the first transmit of a new

user. When false, the records downloaded by the previous user are kept. This attribute should be set when the records of the complex table are user-specific.

Complex Table Fields

A complex table field definition defines a field in each record of the table, including the data type of the field and the size of the data the field can store. A record within the table can consist of multiple fields of varying types and sizes.

A Complex Table is made up of records on the Agentry Client. Each record in the table is made up of Fields. Within a Complex Table definition in the Agentry Editor, you define the fields that make up the table's records.

Complex Table Field Settings

- **Name:** This is the name used to uniquely identify the field within the Complex Table.
- **Display Name:** This is the text value displayed on the Agentry Client for the field. This includes the column headers in a Complex Table Search screen, as well as other places.
- **Type:** This attribute specifies the data type of the field. This is discussed further shortly.
- **No. of Characters:** This attribute is available only for one of the String type of fields and specifies the maximum number of characters the field can hold. Note that this is not necessarily the same as the number of bytes, as is explained in the section of the field data types. When setting this attribute, the value should be large enough to accommodate the strings the records will contain. However, it should not be simply set to an overly large value, as this will waste significant resources on the Client, both in storage and memory.

Field Data Types

There are six data types possible for a Complex Table Field definition. Each controls, first, the type of data that can be stored in the field and, second, how that data is sorted within the table. This last aspect can have a significant impact on how a user can search the complex table on a particular field. Following, each of the data types for a field are listed, along with a description of the impact each type has.

- **ASCII String (case-insensitive):** This field type specifies that the field will contain string characters, each one byte in length. This will support the standard ASCII characters. The case-insensitive portion indicates that, when the table is searched or sorted on this field, the case of the characters is not considered. That is, the lett 'A' is treated as equal to the letter 'a'.
- **ASCII String (case-sensitive):** This field type specifies that the field will contain string characters, each one byte in length. This will support the standard ASCII characters. The case-sensitive portion indicates that, when the table is searched or sorted on this field, the case of the characters is considered. That is, the letter 'A' will be sorted after the lett 'a'.
- **International String (case-insensitive):** This field type specifies that the field will contain string characters in the UNICODE format. This supports the non-english language characters, such as those in Hebrew or Chinese. Note that this also includes characters with an accent mark. This field type will still include the ASCII characters, as well. The case-

insensitive portion indicates that, when the table is searched or sorted on this field, the case of the characters is not considered. That is, the letter ‘A’ is treated as equal to the letter ‘a’.

- **International String (case-sensitive):** This field type specifies that the field will contain string characters in the UNICODE format. This supports the non-english language characters, such as those in Hebrew or Chinese. Note that this also characters with an accent mark. This field type will still include the ASCII characters, as well.
- **Number:** This field type specifies that the field will contain numeric values only. These values may be whole numbers or decimal values and may be positive or negative. If you wish to index a field containing numerical values for the purpose of providing search functionality to the user on the Agentry Client, that field type should be string. Currently, Agentry does not contain the control types on the Agentry Client to support searching numerical values in a complex table.
- **Identifier:** This field type specifies that the field will contain numeric values only. These values may only be whole, positive values. Decimal and negative values are not supported. The purpose of this field type is to explicitly support an identifier field for each record. Note that this is not a requirement, as the other field types can also be used as the identifier value for a record. This is covered in detail in the section in Indexes later in this chapter.

Complex Table Indexes

A complex table index definition orders the records of that table by the field for which the index is created. A field must be indexed to allow for the table to be searched on that field. A complex table can have multiple indexes. Indexes can be defined to have a parent-child relationship to give structure to the table’s records.

The Index definition is the most important of the Complex Table. It is this definition type that makes the Complex Table so useful. When an index is defined, you specify the field to be indexed. When the Complex Table is downloaded to the Agentry Client, its records will be sorted by the fields you have indexed. Only those fields that have been indexed can be searched by the user on the Agentry Client.

Additionally, all Complex Tables must contain at least one index. This is the primary index of the table. The field for this index must contain the unique value for each record in the table. Whenever you define the indexes for a Complex Table, the first index defined is the one treated as the primary index. This cannot be changed once set, so be sure to determine which field should contain the Primary index beforehand. Also, any complex table definition that does not contain a primary index cannot be selected for use by any other definition in the application.

Though only those fields which contain an index can be searched on the Agentry Client, do not simply define one index for each field in the table. There is a certain amount of overhead that goes into each index definition. Also, whenever the records in the complex table are changed, each index must be resorted for each new, updated, or deleted record in the table. This also takes a certain amount of time and resources during a transmit. In a table with a large number of records, superfluous indexes can result in an unnecessary delay for users during transmit.

Complex Table Index Attributes

- **Name:** The internal name of the index. This value must be unique among all index definitions within the same parent complex table.
- **Display Name:** The value displayed for the index definition on the Agentry Client's user interface. In most contexts the index is, to the user, the same as the field and it is a common practice to set the display name of the index to match the display name of the field for which it is defined.
- **Field:** The field for which the index is being created and by which the complex table records will be sorted.
- **Parent Index:** The parent index, set to create parent-child indexes within the complex table.
- **Order:** This attribute specifies the order in which records should be sorted; either ascending (default) or descending.

Parent-Child Indexes

In addition to index a field within the complex table, indexes can also be defined to have parent indexes. This can allow you to create a parent-child relationship among the records of a complex table. The Primary index cannot be defined to have a parent index.

This structure can be very useful when the records of the table support this kind of relationship. One example of such data would a complex table containing locations, with each record representing one location within an industrial park. These locations can be structure to have parent-child relationships and the indexes for the complex table can be created to support this. In this case, a parent location could be a building. Within this building there may be five child locations, one for each floor. Within the first floor of the building, there may be 20 child locations, one for each office suite. Within the first suite, there may be 15 child locations as well, one for each room within the suite. Within the Complex Table, each record would contain, among the other fields, one for the location's ID and one for its parent location, named LocationID and ParentID, respectively.

When defining the indexes for the complex table, an index could be defined on the ParentID field, named ParentIDIdx. Then, a second index definition can be defined for the LocationID field, and this index would have a parent index of the ParentIDIdx index. Within the user interface definitions in Agentry, there are the field types used to create a Cascade. If a cascade were defined for the Locations complex table, the user would first be required to select the Parent ID. Then, they would be presented with a list of just those records in the table with a parent ID equal to the one selected. In the Agentry Editor, these controls are defined to use these parent and child indexes.

SQL Complex Table Synchronization Components

When a complex table is defined to use a SQL Database system connection type, the synchronization components consist of three SQL statements: Reload State Query, Deleted Query, and Data Query.

The reload state query can be enabled or disabled based on preference. When enabled, this query is expected to return the text values “true” or “false.” When the query returns true, the complex table will set to its rebuild state. The condition under which this query returns true is completely dependent on the need of the application or implementation. Its intent is to select from the back end system based on some value or condition that an administrator can easily set when it is desirable to force the complex table to be fully reloaded on the Agentry Client. When this query is disabled, it will not be run by the Agentry Server during synchronization for the complex table.

The data query is always run during synchronization and should include two separate select statements. Both statements are expected to return records from the database to the Agentry Server containing the field values for the complex table records. The columns of this return set must be named to match the names of the complex table fields. The difference between the two statements contained in the data query is the logic related to which records they will select. One statement should be written to select all records to be stored in the complex table on the Agentry Client and under the assumption that the Agentry Client currently contains no records. This statement will then be run for only the rebuild state. The second statement should include logic in support of the exchange data model of synchronization, and should retrieve only new or modified records from the database that will be updated to the records stored on the Agentry Client. To determine if the complex table is in a rebuild state, the SDML data tag `<<rebuild>>` is used. This tag will return true when the rebuild state is set, and false when it is not. The data query will likely check this data tag using the `<<if . . .>>` function tag, which should then return the appropriate statement.

The deleted query is only run when the complex table is not in a rebuild state. This query is expected to return a single column identified as the key field in the complex table. Any values returned by this query will be sent to the Agentry Client so that the Agentry Client will delete the records with the matching key field value from the complex table.

Reload State, Data, and Deleted Query Attributes

- **Enabled:** This attribute is only found for the reload state query. It specifies whether the reload state query is enabled or disabled. The reload state query is only run during synchronization when it is enabled.
- **File:** All three query components contain the File attribute. It specifies the location of the text file (.sql file extension) relative to the Agentry Development Server’s installation location.

Java Complex Table Synchronization Components

When a complex table is defined to use a Java Virtual Machine system connection type, its synchronization component consists of a Java source file. This file contains a skeleton class declaration. This class is created specific to the complex table definition and extends the Agentry Java API class `ComplexTable`.

When the table is defined, the wizard for creating Java classes provided with the Eclipse Java perspective is used. This allows the developer to select the package to which the new class will

be added. The source . java file created will then be stored according to the configuration of the project and package selected for the new class. Alternately an existing class in a package within the Java perspective may be selected. This class must extend the Agentry Java API class `ComplexTable`.

This skeleton class includes the following methods:

- The Constructor method for the class
- An override implementation of `dataIterator()`. This method is intended to contain the logic to retrieve the data from the back end system for the complex table records. It returns an iterator to the data object created to store this returned data. The Agentry Server calls this method during synchronization and uses the returned iterator to extract the data for the records from the array of data objects. Records returned by this method are sent to the Client to be added to the complex table, or to replace those records with matching key field values.
- An override implementation of `deleteIterator()`. This method is intended to contain the logic to retrieve the key field values from the back end system for the complex table records to be deleted from the client application. It returns an iterator to an array of the data object created to store this returned data. The Agentry Server calls this method during synchronization and uses the returned iterator to extract the key field data for the records from the array of data objects. Records returned by this method are deleted from the Client.
- The method `willRebuildTable()` can be created within the complex table class if needed. This method is called by the Agentry Server after the constructor method has been called. Its logic should check for any administrator defined conditions within the back end system to force a complex table rebuild. This method is expected to return a Boolean value. True will set the rebuild state for the complex table.
- The method `build()` can be created within the complex table class if needed. This method is provided to allow for a single call to the back end system to retrieve new, updated, and deleted records. If a build method is present it will be called by the Agentry Server before the iterator methods. In this scenario, the iterator methods are still expected to provide access to the returned data. However, the build method will have already retrieved it. If two separate calls are needed to retrieve updates to the table and deletions from the table, the `build()` method should not be used. The logic for those separate calls should be contained in the iterator methods, which are always called by the Agentry Server.

In versions of the Agentry Mobile Platform prior to 5.1, the source class was stored on the Agentry Server's file system. This behavior is deprecated in versions 5.1 and later. Agentry application projects created prior to this release are still supported and the Java logic will still be processed correctly. New complex tables for Java Virtual Machine system connections should use the new procedure for defining the Java synchronization component.

HTTP-XML Complex Table Synchronization Components

When a complex table is defined to use an HTTP-XML system connection, its synchronization components consist of three HTTP request child definitions: **Update Request**, **Rebuild Request**, and **Deleted Request**.

Each request has the same overall structure and attributes, which includes the URL for the request and the request method. Likewise, the request argument and response mapping child definitions also contain the same attributes. The difference between these requests is when they are sent to the back end system, and what the data they are expected to return is used for in relation to the complex table.

The update request definition is sent to the back end during normal synchronization. This request is expected to return data for the complex table representing records to be added or replaced on the Agentry Client. Therefore it should contain one child data mapping definition for each field in the complex table.

The rebuild request is sent to the back end system when the complex table is in a rebuild state. This request is expected to return the data for all records that should be stored in the complex table on the Agentry Client. The rebuild state means the complex table on the Agentry Client is going to be cleared of all records before the request is sent. This request should contain child data mapping definitions for each field in the complex table.

Complex Table HTTP Request Child Definitions

Following is a list of the child definitions for each of the HTTP Requests within an HTTP-XML complex table.

- **Request Arguments:** This definition encapsulates an argument to be passed with the request to the back end system. Includes the ability to use data within the mobile application with the argument.
- **Response Mappings:** This definition encapsulates the XML data returned by the HTTP request. The specific values are extracted from the XML return data using XPath's defined within each response mapping. Attributes are also set to map the extracted values to data structures within the mobile application.

Complex Table HTTP Request Attributes

The following attributes are set in all three HTTP request definitions within an HTTP-XML Complex Table.

- **Name:** The name of the request, set automatically based on the parent complex table name and the request type. May be modified as needed.
- **URL:** The URL to the specific CGI or other process being called by the HTTP request to synchronize the complex table.
- **Method:** The HTTP request method for the request. May be one of GET, POST, HEAD, or PUT.

HTTP Request Argument

The request argument definition encapsulates a data value to be passed from the mobile application to the process being called by the parent HTTP request definition. The request argument specifies the argument type, which may be CGI Argument, Cookie, HTTP Header, or XML Body. The request argument also specifies the data or data source within the mobile application to pass as the argument to the process or service being called by the parent HTTP request definition.

For a complex table, the data value may be the user ID, the name of the complex table, a fixed string whose value is defined as a constant within the request argument, or markup text. A given parent HTTP request may contain multiple request arguments. The order in which they are passed to the process or service when called is defined in the parent HTTP request's list of request arguments.

HTTP Request Argument Attributes

The attributes of a request argument depend in part on the data type of the argument (Data Type attribute). The following list makes note of those attributes specific to a certain argument data type.

- **Argument Type:** This attribute specifies the type of argument the definition contains. This may be one of CGI, Cookie, HTTP Header, or XML Body.
- **Name:** Alternately displayed as Argument Name, Cookie Name, Header Name, or Name depending on the Argument Type selection. This value must be unique among all request arguments defined within the same parent HTTP request definition.
- **Data Type:** Specifies the data value or source for the data value for the request argument. For a complex table this may be the Table Name, User ID, Small or Large Markup, or Fixed String.
- **String:** This attribute is available only when the **Data Type** attribute is set to Fixed String. String contains the constant string value that is the request argument's data.
- **Markup Text:** This attribute is available only when the **Data Type** attribute is set to Small Markup or Large Markup. **Markup Text** contains the single line (Small Markup) of markup text or the contents of the Markup File (Large Markup) that is the data for the request argument.
- **Markup File:** This attribute is available only when the **Data Type** attribute is set to Large Markup. Markup File contains a reference to the text file containing the multi-line markup text. This file is displayed in the **Markup Text** field directly below the file name in the Editor.

HTTP Request Response Mapping

The response mapping definition is a child to an HTTP request definition. This definition maps a data value returned from the process called by the HTTP request to a value within the mobile application. This value may be extracted from structured XML using XPaths or XSL. It may also be a Cookie value or the HTTP Header.

For a complex table the values may be mapped to the fields in a complex table record, an error message, the last update value to be compared against the complex table's last update, a local data tag or local XML value, or to the user ID value that may be used in place of the ID entered to log into the Agentry Client.

HTTP Request Response Mapping Attributes

The response mapping attributes are in part dependent on the selection made in the Mapping Type attribute. Those specific to a certain type are denoted in the following list.

- **Mapping Type:** This attribute specifies the mapping type. This may be one of Cookie, HTTP Header, XPath Expression, or XML Transformation.
- **Base XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression or XSL Transformation. This attribute is optional and should be used when returning multiple instances of the same data element in the XML content. When a **Base XPath** is defined for a response mapping, the same value will be set by default in the add wizard for subsequent response mappings within the same parent HTTP request definition.
- **XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression. This attribute contains the XPath value to extract the desired value from structured XML data returned from the HTTP Request.
- **XSL:** This attribute is only available when the **Mapping Type** is set to XML Transformation. It contains the XSL expression to transform the XML data to be mapped to a value for the request.
- **Cookie Name:** This attribute is only available when the Mapping Type is set to Cookie. It contains the name of the cookie for the response mapping.
- **Header Name:** This attribute is only available when the Mapping Type is set to HTTP Header. It contains the name of the HTTP header for the response mapping.
- **Maps To:** This attribute specifies where the value extracted by the response mapping is stored in the mobile application. This may be one of the following values for a complex table:
 - **Complex Table Field:** This is the default selection and will result in the value being mapped to the selected complex table field in the table records. This enables the field Field Name, where the complex table field to which the return value is mapped.
 - **Error Message:** This selection will map the data to error text display by the mobile application.
 - **Last Update:** This selection specifies the extracted value is a date and time indicating when the complex table's source in the back end system was last modified. This value is mapped to each record. However, the latest date and time value for all records is the one stored with the complex table on the Client.
 - **Local String (<<local>>):** This selection will create a local data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping. When selected, the attribute String

Name will be available to name the local data tag created. This is the equivalent to calling the SDML function tag `<<local ...>>`.

- **Local XML (<<localXML>>):** This selection will create a local XML data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping.
- **User ID:** This selection will map the value extracted by the response mapping to the user's ID. This value is the equivalent to the SDML data tag `<<user.id>>`. If a previous response mapping in any HTTP Request processed by the Server set the user ID, setting it here will override that value. This will then be the value available for all HTTP-XML system connection processing where the data tag `<<user.id>>` is referenced.
- **Field Name:** This attribute is available when the map type is set to Complex Table Field. This attribute specifies the complex table field to which the values extracted by the mapping is assigned in the complex table records.
- **String Value:** This attribute is available when the map type is set to Local String or Local XML. It contains the name of the data tag being created by the mapping. This is the name by which the data tag is referenced in subsequent references.

Transmit Configuration

The transmit configuration defines how the application on the Agentry Clients can communicate with the Agentry Server. It can define what application-level data definitions to synchronize and the address and port number of an Agentry Server. It also defines whether to log a user out of the Server when a transmit has completed, or to keep them connected to provide real-time communications functionality.

The areas of the communications behavior include:

- The Agentry communications protocol to be used.
- What actions to take, if any, in the event of a communications error.
- Whether or not data tables and complex tables should be synchronized.
- Address and port numbers for the Agentry Server and Midstation.
- Whether or not the client should remain logged in for real-time communications.
- Various aspects of using a modem, such as the Windows Network connection to use, what to do if not currently connected, and other related behaviors.

A transmit configuration is defined for an available communications method on the client device. As an example, if a wireless LAN connection will be available to client devices in the deployment, a transmit configuration for this connection should be defined. If a wireless WAN connection is also available, a transmit configuration should also be defined for this connection type.

Each transmit configuration defined in the application will be listed in the built-in screen called the Transmit Dialog on the Agentry Client. For this reason it is important to consider the proper setting for each transmit dialog's Display Name, as it is this value that will be listed to the user.

Within the transmit configuration's attributes is the Connect Type. This attribute can be set to one of three options:

- Agentry Next Generation Encryption Layer, or ANGEL.
- Midstation
- Unencrypted Network Connection

Each of these connection types perform communications using the TCP/IP protocol. The connection type refers to the type of connection in the context of the application to be used when the transmit configuration is selected by the user on the client application.

Syclo recommends the ANGEL connection for all applications developed going forward. Applications developed using versions of the Agentry Mobile Platform prior to 4.4 being upgraded to the latest platform should be modified so that all transmit configurations use the ANGEL connect type. The Midstation and unencrypted Network Connection types are still available for the purposes of backwards compatibility and will be deprecated in a future release of the platform.

Transmit Configuration Attributes

General Settings:

- **Name:** This is the unique name of the transmit configuration. This value must be unique among all transmit configurations defined for the application.
- **Display Name:** This is the text displayed to the user on the Agentry Client for the transmit configuration in the Client's Transmit Dialog.
- **Connect Type:** This is the communications protocol the Agentry Client is to use when synchronizing with the Agentry Server. The options here are ANGEL or Unencrypted Network Connection. This attribute will not be definable in a future release of Agentry and all transmit configurations will use the ANGEL protocol.
- **Group:** This is the group into which the transmit configuration is organized within the application. This designation is provided for organizational purposes, and there are two options available by default: Fast and Slow. A new group can be created by entering a name in this field. It will then be available in this same drop down for all transmit configurations within the same application project.
- **Failover to:** This attribute can be set to any other transmit configuration within the application. When set the Agentry Client will switch to the selected transmit configuration if it is unable to connect the Agentry Server using the first transmit configuration.
- **Check Data Tables:** This attribute specifies whether or not the data tables within the application should be synchronized when the transmit configuration is used. This is normally unchecked for transmit configurations intended for slower connection types.
- **Check Complex Tables:** This attribute specifies whether or not the complex tables within the application should be synchronized when the transmit configuration is used. This is normally unchecked for transmit configurations intended for slower connection types.

Server Address Settings:

- **Address:** This attribute can be set to the IP address or network name of the host system for the Agentry Server. When set to default, the host will be the one entered by the user on the Client during the initial transmit.
- **Port:** This attribute can be set to the port number of the Agentry Server with which the Client is to connect using the transmit configuration. This is normally set to allow for multiple Agentry Servers running on the same host system, or to allow access through a firewall between the Client and Server.

Transmit Configuration - Session Attributes

General Settings:

- **Stay Logged In:** This attribute controls whether the client user will remain logged in and the Agentry Client will remain connected to the Agentry Server. The purpose of setting this attribute is to support real-time communications within the mobile application, which includes Background Sending and Push behaviors. This requires a constant network connection be available to the Client's. This attribute must be set for any of the other Session attributes to be enabled for the transmit configuration.
- **Prompt on Log In:** This attribute applies when a the connection between the Agentry Client and Agentry Server is lost, and when the transmit configuration is defined to attempt to reconnect. A prompt can be displayed to the user in this situation when the connection is re-established, or hidden from the user based on this attribute setting.
- **Prompt on Log Out:** This attribute controls whether or not the user is prompted when the Agentry Client is logged out of the Agentry Server. When set to false, no prompt is displayed. This only applies when the connection for the transmit configuration is lost and the transmit configuration is set to stay logged in.
- **Inactive Timeout:** This attribute specifies the time limit, in hours, minutes, and seconds, the Agentry Client should remain connected to the Agentry Server with no activity. Activity is defined as the transmission of data between the Client and Server.
- **When Off-line:** This attribute controls whether or not the Agentry Client should attempt to reconnect to the Agentry Server when the connection has been lost. If the Client should reconnect, the duration of time to wait before attempting to reconnect is set in minutes and seconds.
- **Attempts:** This numeric attribute is set only when the **When Off-line** attribute is set to reconnect. The attempts attribute defines how many attempts to make before failing. If the number of attempts is tried without success, the behavior of the Agentry Client is dictated by the transmit configuration's **Failover** to attribute, as well as the **Prompt on Log In/Log Out** attributes.

Background Sending:

- **Allow:** This attribute enables background sending of pending transactions on the Agentry Client. When this attribute is enabled the Client will attempt to send transactions to the Agentry Server in the background as soon as they are applied.

- **Retry Period:** This attribute specifies the amount of time in hours, minutes, and seconds to wait between failed attempts to send a transaction in the background.
- **Attempts:** This specifies the number of attempts to make at sending a transaction in the background before failing.

Push Session:

- **Allow:** This attribute enables Server Push functionality. This functionality also requires the definition of a push within a module of the application. When enabled, users connecting the server using the transmit configuration will be logged in to the Agentry Server as a Push User. This also opens the Agentry Client to receiving push data.
- **Retry Period:** This attribute specifies how long the Agentry Server should wait before attempting to re-send data for a push when a failure occurs.
- **Attempts:** This attribute specifies how many attempts the Agentry Server should make to re-push data before failing.
- **Client Port:** This attribute specifies the port upon which the Agentry Client listens for push communications from the Server. The default port is 7001.

Transmit Configuration - Modem Connection Attributes

- **Check for Modem Connection:** This attribute specifies whether or not the Agentry Client should check for a modem connection when the transmit configuration is used. If true, this check is made prior to beginning the transmit. This attribute must be true for any of the remaining modem attributes to be enabled.
- **Connection Name:** This attribute can be set to the name of any Windows network connection configured on the client device. It can also be set to Any Dial-Up Connection. In the case of the former, it will use the settings of the named connection to establish the modem connection to the network. If set to “Any Dial-Up Connection,” the user will be required to establish the network connection manually outside the mobile application before beginning the transmit. In this case, the remaining modem connection attributes are not enabled.
- **If Not Connected:** This attribute specifies whether the Agentry Client should attempt to create a connection using the Windows network connection named in Connection Name when there is no current connection. If this is set to false, the remaining modem connection attributes are disabled.
- **Connect Prompt:** This attribute is set to the message to display to the user prior to the Agentry Client attempting to create a modem connection. If this attribute is left blank, no message will be displayed to the user prior to creating the connection. Normally the contents of this message prompt the user to connect a phone line or perform similar actions in order for the connection to be made.
- **Username:** This attribute prompts the user to enter a user name for the network connection. This will be used as the login name for the network connection once the modem’s hand shaking processes are successful. If this is set to false, the users Agentry Client login will be used.

- **Password** - This attribute prompts the user to enter a password for the network connection. If this attribute is set to false, the users Agentry Client password will be used.
- **Modem Init Wait:** This attribute specifies the amount of time in milliseconds the Agentry Client should wait for the client device's modem to initialize before beginning the dial-up process.
- **Post-connect Wait:** This attribute specifies the amount of time in milliseconds the Agentry Client should wait after the network connection has been made before beginning the transmit process between the Client and Agentry Server.
- **Close Connection:** This attribute specifies whether the connection made by the transmit configuration should be closed if no data has been transmitted between the Agentry Client and Agentry Server for the specified amount of time. The attribute can be set to Never, meaning the connection will not be closed, or to the minutes and seconds to wait before closing the connection.

System Connection

A system connection sets the connection type the Agentry Server will use to synchronize data with a back end system. A system connection specifies what type of system the Agentry Server is communicating with: SQL Database, Java Virtual Machine, HTTP-XML Server, or File System.

An Agentry application project must have at least one system connection. More system connections can be added if the application requires the Agentry Server to communicate with multiple back end systems. Each system connection may be of different types, or multiple connections for the same type can be defined, depending on the environment in which the mobile application will run.

There are four supported System Connection types:

- **SQL Database** - This system type is used when the Agentry Server needs to communicate with a database system using the Structured Query Language, or SQL. This includes database types such as Oracle or SQL Server.
- **Java Virtual Machine** - This system type is used when the Agentry Server needs to communicate with an interface using the Java Virtual Machine. This logic is implemented using the Java development language and includes usage of the Agentry Java API.
- **HTTP/XML Server** - This system type is used when the Agentry Server needs to communicate with an HTTP server by making HTTP requests that will return structured XML data.
- **File System** - This system type is used when the Agentry Server needs to communicate with the host system upon which the Server has been installed, specifically for file access or command-line processing.

The SQL Database and File System connection types have only the two attributes of Name and the ID number. The name is set by the developer when the system connection is defined. The ID number is generated automatically by the Agentry Editor. This ID number ties the definition to the set of configuration options, configured in the Agentry Server.

The Java Virtual Machine connection type contains the additional attribute API Version. This attribute specifies the version of the Agentry Java API to be used by the mobile application. For all new development, version 5 of this API should be used. Version 4 is available for existing applications developed on versions of the Agentry Mobile Platform prior to the version 5.0 release.

A system connection defined for the HTTP-XML connection type contains the child definition type Validate User Request. This is an HTTP Request definition intended to validate the client user, as well as to capture user information to be stored in the <<user.info>> SDML data tag.

Validate User Request

When a system connection is defined for an HTTP-XML connection type, it can contain one or more HTTP Request child definitions called Validate User Requests. These requests can be made to validate the client user during transmit. This request can also be used to create one or more <<user.info>> SDML data tags.

The validate user request is sent to the back end system at the beginning of the transmit process as a part of the user validation behavior. Each validate user request definition includes child definitions to encapsulate the request arguments, as well as those to map any data returned by the request to structures within the mobile application.

Validate User Request Child Definitions

- **Request Arguments:** This definition encapsulates an argument to be passed with the request to the back end system. Includes the ability to use data within the mobile application with the argument.
- **Response Mappings:** This definition encapsulates the XML data returned by the HTTP Request. The specific values are extracted from the XML return data using XPath's defined within each response mapping. The mapping "maps" the extracted values to values within the mobile application.

Validate User Request Attributes

- **Name:** The name of the request, set by default to ValidateUser. May be modified if desired.
- **URL:** The URL to the specific CGI or other process being called by the HTTP request.
- **Method:** The HTTP request method for the request. May be one of GET, POST, HEAD, or PUT.

Validate User Request Argument

The request argument definition encapsulates a data value to be passed from the mobile application to the process being called by the parent validate user request definition. The request argument specifies the argument type, which may be CGI Argument, Cookie, HTTP Header, or XML Body. The request argument also specifies the data or data source within the

mobile application to pass as the argument to the process or service being called by the parent validate user request definition.

For an HTTP-XML system connection, the data value may be the user ID, the user's password, a fixed string whose value is defined as a constant within the request argument, or markup text. A given parent validate user request may contain multiple request arguments. The order in which they are passed to the process or service when called is defined in the parent validate user request's list of request arguments.

HTTP Request Argument Attributes

The attributes of a request argument depend in part on the data type of the argument (Data Type attribute). The following list makes note of those attributes specific to a certain argument type.

- **Argument Type:** This attribute specifies the type of argument the definition contains. This may be one of CGI, Cookie, HTTP Header, or XML Body.
- **Name:** Alternately displayed as Argument Name, Cookie Name, Header Name, or Name depending on the **Argument Type** selection. This value must be unique among all request arguments defined within the same parent validate user request definition.
- **Data Type:** Specifies the data value or source for the data value for the request argument. For a complex table this may be the User ID, user's password, Small or Large Markup, or Fixed String.
- **String:** This attribute is available only when the **Data Type** attribute is set to Fixed String. String contains the constant string value that is the request argument's data.
- **Markup Text:** This attribute is available only when the **Data Type** attribute is set to Small Markup or Large Markup. **Markup Text** contains the single line (Small Markup) of markup text or the contents of the Markup File (Large Markup) that is the data for the request argument.
- **Markup File:** This attribute is available only when the **Data Type** attribute is set to Large Markup. Markup File contains a reference to the text file containing the multi-line markup text. This file is displayed in the **Markup Text** field directly below the file name in the Editor.

Validate User Request Response Mapping

The response mapping definition is a child to a validate user request definition. This definition maps a data value returned from the process called by the HTTP request to a value within the mobile application. This value may be extracted from structured XML using XPath or XSL. It may also be a Cookie value or the HTTP Header.

For an HTTP system connection the values may be mapped to the user ID, validation, partial validation, the <<user.info>> set of SDML data tags, an error message, a local data tag, or a local XML data tag.

HTTP Request Response Mapping Attributes

The response mapping attributes are in part dependent on the selection made in the Mapping Type attribute. Those specific to a certain type are denoted in the following list.

- **Mapping Type:** This attribute specifies the mapping type. This may be one of Cookie, HTTP Header, XPath Expression, or XML Transformation.
- **Base XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression or XSL Transformation. This attribute is optional and should be used when returning multiple instances of the same data element in the XML content. When a **Base XPath** is defined for a response mapping, the same value will be set by default in the add wizard for subsequent response mappings within the same parent HTTP request definition.
- **XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression. This attribute contains the XPath value to extract the desired value from structured XML data returned from the HTTP Request.
- **XSL:** This attribute is only available when the **Mapping Type** is set to XML Transformation. It contains the XSL expression to transform the XML data to be mapped to a value for the request.
- **Cookie Name:** This attribute is only available when the Mapping Type is set to Cookie. It contains the name of the cookie for the response mapping.
- **Header Name:** This attribute is only available when the Mapping Type is set to HTTP Header. It contains the name of the HTTP header for the response mapping.
- **Maps To:** This attribute specifies where the value extracted by the response mapping is stored in the mobile application. This may be one of the following values for a complex table:
 - **User ID:** This selection will map the value extracted by the response mapping to the user's ID. This value is the equivalent to the SDML data tag <<user.id>>. If a previous response mapping in any HTTP Request processed by the Agentry Server set the user ID, setting it here will override that value. This will then be the value available for all HTTP-XML system connection processing where the data tag <<user.id>> is referenced.
 - **Validation:** This selection will map the value extracted by the response mapping to the validation structures for the Agentry Server. This value is used to indicate whether or not the user passed validation.
 - **Partial Validation:** This selection will map the value extracted by the response mapping to the validation structures for the Agentry Server. This differs from the Validation selection in that mapping the validation result to partial validation can fail user validation with a false response, just as the validation response will, but true for Partial Validation will not fully validate the user. This is intended to provide support for validation using multiple system connections.
 - **Error Message:** This selection will map the data to error text display by the mobile application.

- **Local String (<<local>>):** This selection will create a local data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping. When selected, the attribute String Name will be available to name the local data tag created. This is the equivalent to calling the SDML function tag <<local ...>>.
- **Local XML (<<localXML>>):** This selection will create a local XML data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping.
- **Save to User info:** This selection will map the value extracted by the response mapping to the set of data tags in the <<user.info>> group. When this selection is made, you will also be required to enter a name for the data tag. Referencing these values is then accomplished via the syntax <<user.info.name>>.
- **String Value:** This attribute is available when the map type is set to Local String or Local XML. It contains the name of the data tag being created by the mapping. This is the name by which the data tag is referenced in subsequent references.
- **With Name:** This attribute is available when the map type is set to Save to User Info. it contains the name of the data tag being created by the mapping. This is the name by which the data tag is referenced in subsequent references.

Global

A global definition defines a constant value, including data type, for the application. This value can be referenced throughout the application, both by the attributes of other definition types and for use in synchronization components. A global value cannot be changed on the Client at run-time but can be overridden during synchronization.

A global's value is constant and cannot be modified on the Agentry Client. It can be overridden at run time during synchronization.

The value of a global definition is dependent on the global's data type. Following is a list of the global data types:

- **Boolean:** A value that may be either true or false.
- **Date:** A value representing a calendar date.
- **Date and Time:** A value containing a calendar date and time of day.
- **Decimal Number:** A numeric value that contains a fractional portion and that may be positive or negative.
- **Duration:** A value containing a duration of time in hours, minutes, and seconds.
- **Identifier:** A numeric value that is primarily used to represent an identifying value. Can contain whole, positive numeric values.
- **Integral Number:** A numeric value containing whole numbers that may be positive or negative.
- **Selection:** A special data type for a global definition that represents an attribute setting that is selected from a list. This data type cannot be selected when defining a global, but rather is the automatic data type of the global when it is created specifically for an attribute whose

setting is selected from a drop-down list in the Editor. Valid values for this type of global are those found in the specific list for the attribute.

- **String:** A value containing alphanumeric or other printable characters.
- **Time:** A value containing a time of day.

The data type of a global is important as it will determine where in the application the Global can be used. The data type of the attribute and the global definition used to set it must be the same. For example, string properties contain attributes for their size, i.e., the number of characters they can contain. This size attribute is an integral number. This then requires the user of a global with a data type of integral number.

A global definition may be added to the application project from either the list of globals for the application, or at the point where it will be referenced by another attribute that may be set via a global. In the latter case, the data type of the global is set automatically based on the data type for that attribute.

Global Attributes

- **Global Type:** The data type of the global, selected when the global is added to the application, or set automatically by the Editor based on the attribute to use the global for its setting.
- **Group:** The group into which the global definition will be organized. Unlike the group setting for other definitions, a global's group is a required attribute. References to the global definition throughout the application must include its group as well as its name.
- **Name:** The unique identifier for the global definition. This value must be unique among all global definitions within the same group.
- **Value:** The value of the global definition returned when the global is referenced. Valid values for a global depend on its defined **Global Type**.

Style

A style definition defines a set of style elements that can be applied to the Agentry Client's user interface to affect its appearance. These elements include text and background colors, font face and size, borders, and other similar UI items. A style may be defined for all supported application platforms or for a single platform.

The Agentry Editor allows the developer to create display styles for screens, buttons, text, fields and list controls. A style is defined as a collection of display elements combined to provide an overall look and feel to the application.

Styles exist at the application level in a project. They are then available to be used, or "applied" at the application, module, platform, screen and control levels. Each attribute, or "style element" of a style definition may be set to a specific value or default. Default results in the system default being used for that aspect of the user interface.

If styles are applied at multiple levels within the application they are merged at run time before being applied to the user interface. The style definition applied at a lower level in the application hierarchy will override the settings of a style applied at a higher level. If the lower

level style has an element set to default, the setting for that same element in the higher level style definition will be used. This merge then results in the overall appearance of the user interface component to which the style is applied.

A style may be defined for a specific platform. Multiple styles may be defined with the same name but with different platform selections. When a style is applied to the user interface, only the name is referenced. At run time, a given client device type will receive only the styles with a matching platform. This is optional behavior and a style may be defined for all platforms.

iPhone and iPad/iPod Touch Platform Note

Due to the nature of the iOS devices, the current style support for these device platforms is limited to the specification of the Font Face and the Foreground Color. Styles can only be applied to specific controls and the affects of the two supported style attributes are the font in which text is displayed and the color of that text.

Style Attributes

Following is a list of the attributes for a style definition. In the context of a style definition these attributes are commonly referred to as “style elements” and the terms are interchangeable:

- **Foreground Color:** This attribute specifies the color of any text displayed on the user interface component to which the style is applied. If a particular user interface component has no text, the Foreground Color setting will have no effect on its appearance.
- **Background Color:** This attribute specifies the color of the background of the user interface component. The background of a screen or control is the area that contains no controls, text, or list items.
- **Font Face:** This attribute specifies the font used to display any text on the user interface to which the style is applied. Within the Agentry Editor, the **Font Face** attribute field contains a drop-down list. Its contents will be any fonts installed on the host system of the Editor. The name of a font may also be manually entered if it is one that is known to be available on the client devices, even if it is not available on the Editor’s host system. Any font face entered manually in the list will be available in this same list for all style definitions within the application. If a font name is entered that is not available on a client device, the behavior will be the same as if Default had been selected for the **Font Face** attribute.
- **Point Size:** This attribute specifies the size of the text displayed to the user. If the point size is larger than the viewable area given to that text value, that viewable area will not be increased in size.
- **Font Style:** This attribute specifies whether the text is displayed normally (referred to as the regular font style), or in bold, italics, or bold italics. The Font Style attribute may not have an effect on the appearance of the text based on the selected Font Face. Certain fonts are inherently bold or italicized, or may not support either behavior.
- **Underline:** This attribute specifies whether the text is underlined. This attribute may have no effect on certain Font Face selections, as the selected font may not support underline or may be inherently underlined.

- **Border Style:** This attribute controls how the border around certain UI components will appear. This includes detail screen fields and buttons. The border style can be None, Flat, or 3-D.
- **Text Alignment - Horizontal Alignment:** This attribute specifies the alignment of the text displayed by the UI component to which the style is applied. The options are:
 - “Align Left” - This selection specifies that the text is to be aligned to the left of the viewable area allotted for the text being displayed.
 - “Align Right” - This selection specifies that the text is to be aligned to the right of the viewable area allotted for the text being displayed.
 - “Center” - This selection horizontally centers the text within the viewable area allotted for the text being displayed.
 - “Default” - This selection will horizontally align the text according to the default behavior of the item containing the text.

Image

An image definition incorporates an image file into the application data. This image can be displayed on various components of the Agentry Client’s user interface. An image can be used to add icons and interactive graphics to the UI for branding purposes and to enhance the user experience.

Once an image has been defined it can be referenced in several components of the user interface definitions. This can include button icons, list icons, and detail screen fields, as well as the login dialog, the module selection dialog, and the help dialog. The first group of definition types that may reference an image definition support the use of image lists, which can allow for the display of a different version of the image based on some condition.

When an image definition is created, it must use a file of one of the types:

- Bitmap
- JPEG/JPG
- GIF
- PNG

The selected file must exist prior to creating the image definition. The file is copied within the application project definitions. Modifications to the selected source file after this point will not effect the appearance of the image within the application. This image can be edited from within the Editor if necessary.

An image may be defined for a specific platform. Multiple image definitions may exist with the same name and different selected platforms. References to images from other definitions within the application are made by name only. At run time a given device type will receive the images defined only for the matching platform. This is optional behavior and an image definition can be created for all platforms.

Image Attributes

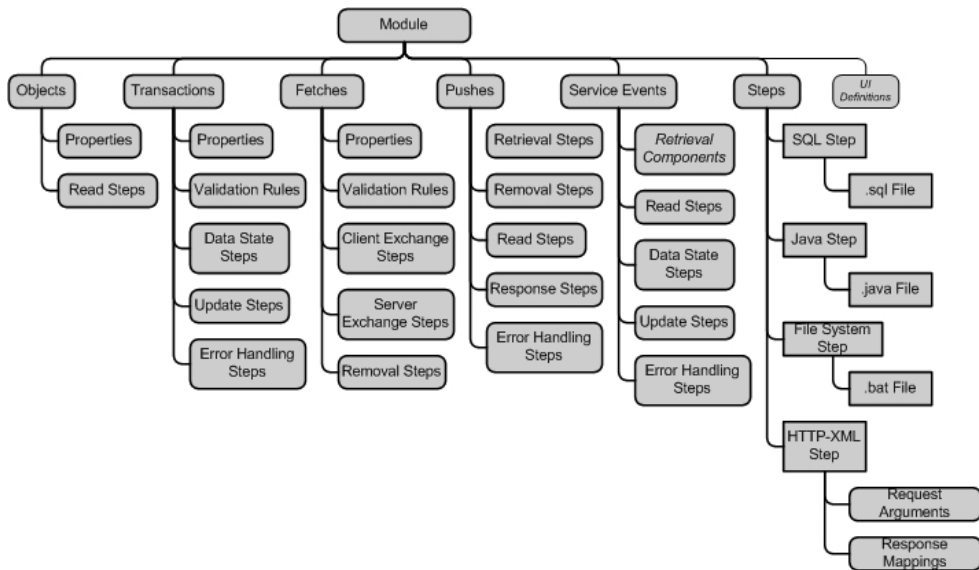
- **Name:** This is the internal name of the image definition. This value must be unique among all Images with the same setting in the Platform attribute.
- **Platform:** The platform attribute specifies the platform of the client devices to which the image will be downloaded. This can be either All, or one of the available platforms listed. Selecting a specific platform will prevent the image from being downloaded to any device of any other platform. This can be used to download images with different file sizes to different client devices while using the same name.
- **Image File:** This value is the file name that will be used to store the image file within the application project, as well as by the Agentry Server and Agentry Client. The default **Image File** value is a combination of the **Name** and **Platform** attribute values. It is rarely necessary to change the **Image File** setting.
- **Mask Color:** This optional attribute can be used to create a mask color for the image, which will be incorporated into the image's display on the Agentry Client. This setting does not affect the image file itself, but rather is applied at run time. It is set using the RGB values, or by selecting the desired color from the Windows color palette. A mask color is used to remove a color from the displayed image, such as the white background of an icon.
- **Image:** This is the actual image file that contains the image. This is selected by clicking the ellipsis button to the far right of the field, which will display the standard Windows File Dialog. From here you can select the file to be used for the image definition. You will only be allowed to select files of the types .bmp, .gif, .jpeg, .jpg, or .png. Once the file has been selected it will be displayed in the image definition within the Agentry Editor.

Module-Level Data Definitions Overview

Within the module level of the application project in Agentry there are definitions for both data and user interface encapsulation. The data-focused definition types include those for business entity encapsulation, data capture, and data synchronization between the Agentry Client and Agentry Server.

Most of the data definitions at the module level have child definitions of their own. Each child definition encapsulates some aspect of the parent's behavior related to the data for which it was defined. This can include the values for the parent definition, or the methodology for data synchronization.

Following is an illustration representing the structure of the module-level data definitions within the application project. This includes the definitions within the module provided to encapsulate data storage or synchronization, as well as the child definitions to each. Excluded from this graphic are the user interface definitions within the module. Note that this separation is for discussion purposes only. Within the application project structure, all child definitions to the module exist at the module level with no distinction made between them in the Agentry Editor in relation to whether they are data or user interface definitions.



A common child definition to objects, transactions, and fetches are the properties. A property is a variable data value stored within the parent definition. The purpose for these values differs depending on the parent definition, but the property definition type itself is the same among all three.

Many of the child definitions to the module-level data definitions are referred to as “step usage definitions.” This term describes a definition that references a step definition within the same module. This reference provides the context to the step, specifying why and when it should be executed by the Agentry Server during synchronization. Any child definition to a module-level definition that includes the term “Step” in its name is a step usage definition. The creation of a step usage definition requires that the step to be used exists first.

As illustrated in this graphic, the step definition itself is defined for different types of processing. Steps are defined for a specific system connection within the application. The step definition has a type that matches the system connection type. The step will then contain a component matching that type, such as a SQL statement or Java logic. HTTP-XML steps include two child definitions that define the arguments passed to the HTTP server with the step request, and mappings between the data returned from that request to the data components of the mobile application.

The data definitions illustrated and described here are displayed, modified, and exposed to the mobile application uses via the module-level user interface definitions. The data definitions must exist before the user interface definitions can be created, as the UI definitions will need to reference the data definitions they display.

Object

An object definition encapsulates a business entity and its related data. An object's child property definitions give that object its characteristics. An object can also define how its data is retrieved from the back end system.

The object definition is in essence a container for the properties defined within it. Objects are defined to encapsulate the different business entities in a module in support of the functionality to be provided in the mobile application. The properties then define the data stored within that object.

A special type of object will exist in every module defined within an application project. This is the module main object, named by default MainObject. The intent of this main object is to be the starting point, or top level of the module's overall object data structure. Via the use of the collection property data type, object instances may be stored within other object instances at run time. This then results in a parent-child relationship within the module's data structure. At the top of this structure is the module main object.

When a new module is defined, the module main object will be added automatically. Additionally, a prompt is displayed in the New Module Wizard for the definition of another object. The object defined in the New Module Wizard is normally the primary object for the module. The primary object is a term of convenience used to denote the object around which most of the module's functionality will revolve. This includes the functionality provided to the end user in the form of information and data capture, as well as synchronization processes for the module. Examples of a primary object include a work order object for a work management module, or a message object for a mail module. The module main object will include a single collection property defined to store instances of the primary object.

Object Child Definitions

- **Property:** An object property defines a single piece of data for the parent object.
- **Object Read Step:** An object read step references a step definition run to retrieve data from a back end system to populate an object's properties.

Object Attributes

- **Name:** This is the unique name of the object. This value must be unique among all objects defined within the same module.
- **Display Name:** This is the default name displayed for the object on the Agentry Client.
- **Key Property:** The key property for an object is used whenever that object is to be a part of a collection. The value of this property must be a value that uniquely identifies the object and in most cases will be the same value as the key value from the back end system. Note that almost all object definitions are stored in collections and therefore must have this attribute set. The property to be used must be defined before setting this attribute.
- **Transmit Display Property:** This attribute specifies the object property value to display to the user on the Agentry Client transmit screen. When an object is being retrieved from

the back end system, the property specified here is displayed to the user during its retrieval. By default the value displayed is the object's key property.

- **Main Object:** This attribute specifies whether or not the object is the main object for the module. Each module contains a main object. This attribute is set to true for that object, and to false for all other objects. This attribute is displayed for reference purposes within the Editor and cannot be modified.

Object Read Step

An object read step references a step definition within the same module. Its purpose is to retrieve data for instances of the object from the back end system. The steps are processed by the Agentry Server during a transmit. The step being referenced can be executed once per transmit or iteratively.

The data returned by the object read step is expected to be identified to match the property values of the object. How this data is identified is dependent on the type of step being executed. A given read step need not return all data values, but must always include the key property of the object type for which it is retrieving data and the key property of any parent objects up to the top-level object in the module's data structure.

Object read steps are executed by the Agentry Server in any of the following situations related to the parent object definition:

- When a fetch is processed that is defined to target a collection of the read step's parent object type.
- When a push defined to target a collection of the read step's parent object type polls the back end for data changes and finds this to be true, and when that push is defined to use the object's read steps to retrieve the data rather than the push read steps.
- When the processing of a transaction targeting the read step's parent object type sends a client response of replace client object.

In any of these situations, the read steps for an object will be run and the data returned will be used to either create new object instances or replace existing object instances that will ultimately be sent to the Client.

It is important to note that the step being executed by the read step must account for which situation it is being run. The read step definition itself is not aware of the synchronization context in which it is being executed.

Object Read Step Attributes

- **Step:** This attribute references the step definition within the same module to run as an object read step for the parent object.
- **Run:** This attribute specifies how to run the read step during a single transmit. This may be set to one of the following values:
 - *Run one Time:* This setting will run the read step a single time for a given synchronization context. This setting assumes the step need be executed only once to return the data for all object instances to be added or replaced during synchronization,

or the step being executed is not returning data but rather is being run in support of synchronization.

- ***Run once per Object:*** This setting will execute the read step once for each object instance in the collection that is being synchronized. This includes both those object instances sent by the Client to the Server, as well as any that may have been added by previous synchronization steps. For push processing the step will be executed once for each object instance created prior to the read step's execution. For transaction processing this setting will have the same behavior as "Run one Time."
- ***Run once per Collection Object:*** This setting will execute the object once for each object instance in the child collection referenced by the Read Into attribute. This child collection is assumed to have been populated with object instances prior to the read step's execution. Note that this setting is primarily intended for file transfer functionality, though it is not limited to this purpose.
- **Read Into:** This attribute specifies the child or descendent object collection property of the read step's parent object into which the data returned by the step should be read. This attribute has a default setting of "None." This default means the data will not be read into a child collection but will instead be used to create object instances of the read step's parent object. Other valid options for this attribute are any child collection properties of the read step's parent object, or any descendent collections (e.g. collections within collections) of the parent object.

Object Property

An object property definition defines a single piece of data and its type. A property can also define minimum and maximum values, a default, or "special value" and other data-related behaviors.

The properties of an object define the aspects of the business entity the object is intended to encapsulate. Each object must include a key property that will uniquely identify each instance of the object at run time. The object key property is important to all aspects of object data synchronization. Both the Agentry Client and the Agentry Server use this value to determine if an object is added to a collection, or if it should be replaced. On the Agentry Client, a new object cannot be added to a collection using an add transaction if the instance it creates has the same key property value as an existing object instance.

The key property is specified in the object definition itself in the Key Property attribute. The property to be used as the key property must be defined first.

The attribute Name is an important one to consider when defining the properties of an object. In addition to uniquely identifying the property definition within the parent object, it also plays a part in the downstream synchronization of objects at run time.

When a step definition returns data for an object, the values returned will be identified in some manner depending on the type of step. For a SQL step this is the column name designated in the SELECT portion of the step's query. In a Java step, it is the name of the members of the `returnData` structure. HTTP-XML and File steps use different mechanisms involving

mapping behaviors. Regardless of the step type, the name used by the step to identify a value must match the name of the property definition. The Agentry Server will populate a property with the value in the data returned by the step with the same name or identifier as that property definition's Name.

Properties are defined to be of a certain data type, of which there are many in Agentry. They are a child definition to the object, transaction, and fetch definitions. Each property data type has its own set of attributes specific to that type. Review the information on property data types for more detailed information on properties.

Transaction

The transaction definition defines data to be captured on the Agentry Client. As a part of its definition, the transaction includes a target object type, data values to be captured, client-side data validation, and processing its data to the back end system by the Agentry Server during synchronization. Transactions can add new object instances, edit an existing object, delete an object, or modify an complex table or data table record. Each of these behaviors is exhibited by a different transaction type, selected during the creation of the transaction.

A transaction definition is created within the application to target a specific object type within the same module. Transactions are instantiated on the Agentry Client one at a time as the result of the execution of a transaction step within a client action. A transaction instance can target only one instance of an object.

The transaction can be displayed to the user in a screen set, which will behave as a wizard allowing the user to enter data in a series of one or more screens.

There are five different types of transactions that can be defined for an application. Each captures data for a specific type of change on the Agentry Client. The transaction types are:

- **Add:** An add transaction type is defined to allow the user to create a new object instance on the Agentry Client.
- **Edit:** An edit transaction is defined to allow the user to edit the property values of an existing object instance on the Agentry Client.
- **Delete:** A delete transaction is defined to remove an object instance from the Agentry Client.
- **Data Table Change:** A data table change transaction is defined to allow the user to add or edit a data table record on the Agentry Client.
- **Complex Table Change:** A complex table change transaction is defined to allow the user to add or edit a complex table record stored on the Agentry Client.

Transaction Child Definitions

All transactions, regardless of type, have the same child definitions. The purpose of these child definitions is the same for all transaction types.

- **Properties:** A transaction property defines a single piece of data a transaction will capture, including its data type and initial value.

- **Validation Rules:** A transaction validation rule defines what rule definition will be used to validate the transaction's data and how failed validation is handled on the Agentry Client.
- **Server Data State Steps:** A transaction server data state step references a step definition within the same module to be run by the Agentry Server to check the back end system for data collisions during transaction processing.
- **Server Update Steps:** A transaction server update step references a step definition that is run during transmit to update the back end system with the data captured by the transaction.
- **Error Handling Steps:** A transaction error handling step references a step definition that is run during transmit if an error occurs while processing the transaction's data state or update steps.

Transaction Attributes

Transaction attributes specify the type of transaction, the object type it targets, the key property of the transaction, and the transaction's name and display name. There are also type-specific attributes for the different transaction types. Review the information on the specific transaction types for details on these attributes.

Transaction Authentication

Transaction authentication is definable behavior for all transaction types and is available to support user authentication during data capture, often referred to as "electronic signatures". To define this behavior, attributes specific to authentication must be set within the transaction definition after it has been defined. These attributes are not displayed in the add transaction wizard.

These attributes are used to define transaction authentication on the Agentry Client. This functionality provides the means to authenticate users when they make data changes. Using transaction authentication you can require users to enter their user ID, password, and other information as may be necessary.

Transaction authentication is defined for each transaction definition. This allows for authentication behavior to be exhibited only for data capture operations that require it.

This information can be captured in properties of the transaction itself, or in an instance of an object defined specifically for this purpose, called the authentication object. A separate screen set defined to display the authentication object to the user must exist prior to defining the authentication within the transaction.

Transaction Authentication Attributes

The following attributes are common to all transaction types. They are set to define the transaction authentication behavior. They can only be modified for existing transactions and are not displayed during the add transaction wizards.

- **Screen Set:** This attribute is set to the screen set to display to the user for the purpose of entering the authentication information you wish to capture. This can include the user ID,

password, and other information as may be necessary. If this is set to “No Authentication” the authentication functionality is disabled for the transaction.

- **Authenticate When:** This attribute determines when the transaction requires authentication. This can be set to: “Do Not Authenticate”, disabling the behavior; “Always authenticate”, or can be based on a rule definition. When a rule is referenced by this attribute, it is evaluated in the context of the transaction and is expected to return a boolean value. A true return will require the user to authenticate. A false return will not and the authentication screen set will not be displayed.
- **Information In:** This attribute is set to either “Properties of this transaction” or to an object type defined within the same module. If set to the former, the properties displayed in the Authentication Screen Set are defined within the same transaction. If set to an object, the properties of that object are displayed and store the authentication data.

Transaction Type: Add

An add transaction type is defined to allow the user to create a new object instance on the Agentry Client. An add transaction definition includes a target object collection property to which the new object instance will be added. This transaction type should contain all non-collection properties found in the object type it creates.

Add Transaction Attributes

Following are the attributes for an add transaction:

- **Type:** This attribute specifies the type of transaction. For add transactions this is set to “Add”. This attribute cannot be changed once the transaction has been defined.
- **Object:** The Object attribute specifies the type of object the add transaction instantiates on the Agentry Client. This may be any existing object defined within the same module.
- **Collection:** The collection attribute specifies the collection property in which the new object instance will be stored on the Agentry Client when the transaction is applied.
- **Name:** This is the identifier for the transaction definition. This value must be unique among all transaction definitions within the same module.
- **Display Name:** The display name is the value shown to users for the transaction on the Client. This is normally seen by the user in the Agentry Client’s Transmit Screen when an instance of the transaction is sent to the Agentry Server to be processed.
- **Key Property:** This attribute specifies the property within the transaction to be treated as the key property. This is almost always the transaction property that targets the object’s key property and is set as such by default.

Transaction Type: Edit

An edit transaction is defined to allow the user to edit the property values of an existing object instance on the Agentry Client. This type of transaction should, at a minimum, include the key property of the object type and all property values that should be changed in the object.

It is highly recommended that users never be allowed to edit the key property of an object, as this can make it difficult, if not impossible, to update the enterprise system with any other

changes for the object. Remember that the key property of an object is the value that uniquely identifies that object within both the mobile application and the enterprise system.

When an edit transaction is applied, the value of the properties are copied to the object properties they target. These new values will replace the previous values of the object properties. Object properties not modified by the transaction will not be changed. Once an object property is updated from an edit transaction, the previous value of that object property is lost and cannot be recovered.

When designing and developing an Edit transaction, the developer should consider whether or not the transaction definition should include merge functionality. Transaction merging is the behavior when an instance of an edit transaction is merged with an existing pending transaction targeting the same object instance on the Agentry Client. This functionality is controlled by the Edit transaction's merge attributes and is optional behavior.

Edit Transaction Attributes

Following are the attributes for an edit transaction:

- **Type:** This attribute specifies the type of transaction. For edit transactions this is set to Edit. This attribute cannot be changed once the transaction has been defined.
- **Object:** The Object attribute specifies the type of object the edit transaction targets on the Agentry Client. This may be set to any object type defined within the same module.
- **Name:** This is the identifier for the transaction definition. This value must be unique among all transaction definitions within the same module.
- **Display Name:** The display name is the value shown to users for the transaction on the Client. This is normally seen by the user in the Client's Transmit Screen when an instance of the transaction is sent to the Agentry Server to be processed.
- **Key Property:** This attribute specifies the property within the transaction to be treated as the key property. This is almost always the transaction property that targets the object's key property and is set as such by default.
- **Merge When:** This attribute specifies when the transaction should be merged. This can be set to either "Merge with adjacent transactions only" or "Merge with any transaction" to enable transaction merging on the Client. Adjacent transactions means the last transaction applied on the Client. Any transaction means the transaction will be merged with the first transaction found to meet the proper criteria for merging. This begins with the most recently applied transaction for the same object instance. The search continues back to the first applied transaction, or until a transaction is found that the edit transaction can be merged with.
- **Merge With:** This attribute specifies the type of transaction the edit transaction should be merged with. This can be set to "Same transaction type only" or "Similar transactions." The same transaction type is only another instance of the same edit transaction that targets the same object instance on the Client. A "Similar transaction type" also must target the same object instance on the client, but may be an instance of any add or edit transaction that meets the merge criteria.

- **Timestamp:** The timestamp can be set to either “New Timestamp” or “Original Timestamp.” This attribute specifies whether the timestamp from the original transaction is kept after the merge, or whether the timestamp from the new transaction instance is used.

Transaction Type: Delete

A delete transaction is defined to remove an object instance from the Agentry Client. When applied, this transaction will remove the object instance from the Client and may also remove any pending transactions for that object instance. A delete transaction should, at a minimum, contain the key property of the object type it targets.

When a delete transaction is applied, the object instance targeted by the transaction is removed from the Client. All data properties of the object instance, including any object collection properties, are removed.

When defining a delete transaction, the developer should ensure that the object should be allowed to be deleted. This is normally controlled by defining an enable rule for the action that will instantiate the delete transaction. The object and its data removed by the delete transaction cannot be recovered once the transaction has been applied.

Delete Transaction Attributes

Following are the attributes for a delete transaction:

- **Type:** This attribute specifies the type of transaction. For delete transactions this is set to “Delete”. This attribute cannot be changed once the transaction has been defined.
- **Object:** The Object attribute specifies the type of object the delete transaction targets and will remove from the Agentry Client.
- **Name:** This is the identifier for the transaction definition. This value must be unique among all transaction definitions within the same module.
- **Display Name:** The display name is the value shown to users for the transaction on the Client. This is normally seen by the user in the Client’s Transmit Screen when an instance of the transaction is sent to the Agentry Client to be processed.
- **Key Property:** This attribute specifies the property within the transaction to be treated as the key property. This is almost always the transaction property that targets the object’s key property and is set as such by default.
- **Discard Pending Transactions:** This attribute specifies whether or not pending transactions for an object instance removed by the delete transaction should also be removed. If this attribute is set, pending transactions targeting the deleted object instance will be removed. If false, these pending transactions will remain on the Agentry Client until the next transmit.

Transaction Type: Complex Table Change

A complex table change transaction is defined to allow the user to add or edit a complex table record stored on the Agentry Client. This transaction type is still defined to target an object type. It should, at a minimum, contain a property for the key field of the table and the

properties to target each field to be modified by the transaction. To allow for the addition of a new record, it should contain one property for each field in a table record.

When a complex table change transaction is applied, the transaction first looks for a record in the complex table whose key field value is equal to the value of the corresponding property in the transaction. If a match is found, the record is updated with the property values of the transaction. If no match is found, a new record is added to the complex table. The indexes of the complex table are then updated to match the new or modified record.

Complex Table Change Transaction Attributes

Following are the attributes for a complex table change transaction.

- **Type:** This attribute specifies the type of transaction. For complex table transactions this is set to “Complex Table Change.” This attribute cannot be changed once the transaction has been defined.
- **Object:** The Object attribute specifies the type of object the transaction targets. Though primarily intended to change a complex table record, this transaction type must still target an object.
- **Table:** The table attribute specifies the complex table the transaction targets and that will be changed when the transaction is applied.
- **Name:** This is the identifier for the transaction definition. This value must be unique among all transaction definitions within the same module.
- **Display Name:** The display name is the value shown to users for the transaction on the Client. This is normally seen by the user in the Client’s Transmit Screen when an instance of the transaction is sent to the Agentry Server to be processed.
- **Key Property:** This attribute specifies the property within the transaction to be treated as the key property. If no change is made to the targeted object, this attribute need not be set.

Transaction Type: Data Table Change

A data table change transaction is defined to allow the user to add or edit a data table record on the Agentry Client. This transaction is still defined to target an object type. It should, at a minimum, contain a property for the key and value fields of a data table record.

When a data table change transaction is applied, the transaction first looks for a record in the data table with the same key value as the corresponding key property in the transaction. If one is found, that record will be updated from the property for the value field. If there is no match on the key field, then a new record will be added to the data table using the values of the two properties for the key and value fields. Data table change transactions cannot delete a record from a data table.

Data Table Change Transaction Attributes

Following are the attributes for a data table change transaction.

- **Type:** This attribute specifies the type of transaction. For data table transactions this is set to “Data Table Change.” This attribute cannot be changed once the transaction has been defined.
- **Object:** The Object attribute specifies the type of object the transaction targets. Though primarily intended to change a data table record, this transaction type must still target an object.
- **Table:** The table attribute specifies the data table the transaction targets and that will be changed when the transaction is applied.
- **Name:** This is the identifier for the transaction definition. This value must be unique among all transaction definitions within the same module.
- **Display Name:** The display name is the value shown to users for the transaction on the Client. This is normally seen by the user in the Client’s Transmit Screen when an instance of the transaction is sent to the Agentry Server to be processed.
- **Key Property:** This attribute specifies the property within the transaction to be treated as the key property. If no change is made to the targeted object, this attribute might not be set.

Transaction Validation Rule

A transaction validation rule defines what rule definition will be used to validate the transaction’s data and how failed validation is handled on the Agentry Client. The rule referenced is called in a Boolean context and is expected to return true or false. False indicates failed validation, which may be treated as a warning or error. Messaging may be displayed to the user in relation to failed validation. An error requires the user to change the offending value(s) before proceeding. A warning displays an informational message giving the user the option to change the value(s), but does not require a change.

Not every transaction will have validation rules. Certain types of values do not need to be validated using a validation rule. Simple requirements such as the size of a string value or the minimum and maximum values of a numeric property can be enforced by the property itself. In other cases the information may not need to be validated. An example of this is some sort of note or description entry where the user is entering free form text.

Validation rules are used when more complex validation is required, such as when the valid value for a property is dependent on the value of a second property. Also, validation rules offer the flexibility to differentiate between a warning and an error. With a warning, the user is given the option of changing the value that violates the rule or leaving it as is. If treated as an error, the user must change the value before being allowed to proceed.

Validation rules are evaluated when the user clicks any navigation buttons in the wizard screen set displaying the transaction. If a validation rule references a property not yet displayed in the wizard, it will not fail validation. A false return by the rule is treated as a validation failure and the validation rule definition will then dictate the behavior of the Agentry Client.

Validation Rule Attributes

- **Rule:** This attribute specifies the rule definition within the same module to be used as a validation rule for the transaction. The rule is expected to return a Boolean value is evaluated in the context of the current transaction instance.
- **Type:** This attribute can be set to either “Warning” or “Error” and determines how a false return from the rule is treated by the validation rule. Warning means a failed validation does not require the user to change the value. The user will be displayed a message and given the option to change the value or keep it as set. An error type requires the user to change the value before proceeding.
- **Caption:** This is the text displayed in the title bar of the message for the validation rule.
- **Text:** This is the message displayed to the user when validation fails.
- **OK Label:** This is the text to label the OK button for the message screen on the Agentry Client.
- **Cancel Label:** This is the text to label the Cancel button for the message screen on the Agentry Client. This attribute is available only when the **Type** attribute is set to “Warning.”

Transaction Validation Rule Properties

Rule properties associate one or more object properties with a transaction validation rule. Rule properties are used to set the cursor focus on the Client when a validation warning or error rule is triggered. The focus is set to the first property on the current screen set screen that is included in the Rule Properties list. If no match is found or if all of the listed properties are contained in a screen other than the current screen shown on the Client, no focus is set.

Validation rules and their associated rule properties are evaluated when the user clicks any navigation buttons in the wizard screen set displaying the transaction. If the rule returns false, the rule runs through all properties on the rule list until it finds one that matches a transaction property that is displayed on the Client screen. At that point, the ‘next’ or ‘finish’ process is stopped, the user remains on the same screen, and the cursor focus is set to the matching property. Any property controls that are hidden, disabled, or set to read-only will be ignored.

Validation rule properties are available on the following transaction types: Add, Edit, Complex Table Change, and Data Table Change.

Setting rule properties is optional. If no rule properties are set and a rule returns false, only the error or warning message associated with the rule is displayed on the Client.

Transaction Server Data State Steps

A transaction server data state step references a step definition within the same module to be run by the Agentry Server to check the back end system for data collisions during transaction processing. Server data state steps are the first steps run by the Server when a transaction is being processed. When a data state step’s return is true, its defined data state is set for the transaction. This data state will then affect which server update steps for the transaction are

run by the Server. A data state step may also define a response to be sent to the Client to perform some additional action in relation to the object instance targeted by the transaction.

The step executed by a data state step should be defined to either return data or not, based on some condition. The data returned by a data state step is unimportant in most cases. The exception to this is when the Client Response attribute is set to “Update Client Key Property.” In this scenario the Agentry Server will expect the step to return a value identified as the key property for the target object.

Other than in this situation, the Server only looks to see if data is returned by the step. By default, when data is returned by a data state step, the Server treats this as a true response and will set the defined data state for the transaction. No data will be treated as false. This logic may be inverted, with data return treated as false and no data treated as true, if the logic of the step being executed is more efficient or more practical to be written in this manner.

If multiple server data state steps return true for a single transaction instance, the defined data state for the last step with a true return will be the one set for the transaction. The server data state steps may be defined to halt further data state step processing for the transaction if one of them returns true. The order in which server data state steps are processed is defined in the list of these definitions displayed in the transaction definition’s properties view of the Editor.

The Client Response attribute allows for the definition of a response to be sent to the Client in relation to the object targeted by the transaction. This response will be sent after the transaction has been successfully processed by the Server. The defined Client Response for a data state may be overridden by a subsequent data state step, or by the transaction’s server update steps.

Server Data State Step Attributes

The following is a list of the attributes for a server data state step definition:

- **Step:** This attribute references the step definition within the same module to run as a server data state step for the transaction.
- **Data State:** This text value is the name of the data state the data state step will set if its return is true. This value can then be referenced by the server update steps for the same transaction.
- **Step is True if:** This attribute is set to define what is treated as true for the data state step. When a data state step is true, its defined data state is set for the transaction. Its available options depend on the type of step selected in the **Step** attribute:
 - *SQL Step:* For a SQL step, this attribute can be set to “1 or more rows are returned” or “0 rows are returned”. The former will treat data being returned as true and no data returned as false. The latter will treat data returned as false and no data returned as true.
 - *Java Step:* For a Java step, the options are “doSteplet returns True” and “doSteplet returns False”. The first will treat a true return from the `doSteplet()` method of the Java step as true. The second will treat a false response from the `doSteplet()` method as true.

- *HTTP-XML Step*: For an HTTP-XML step, the available options for this attribute are “All response mappings succeed” and “A response mapping fails”. The former will set true for the data state when the HTTP-XML step is able to map all of the responses, per its definition. The latter will treat one or more failed mappings as true.
- **If True**: This attribute specifies whether the remaining server data state steps for the transaction should be processed if the data state step returns true.
- **If False**: This attribute specifies whether the remaining server data state steps for the transaction should be processed if the data state step returns false.
- **Response to Client**: This attribute specifies what response is sent to the Agentry Client after the Update Step has been processed. The response defined here will only be sent if the data state step is run and returns true. The responses that may be sent are “Delete Client Object”, “Replace Client Object”, “Update Client Key Property”, and “No Action Required”. If “Update Client Key Property” is set, the step being run by the server data state step is expected to return a value identified as the transaction’s target object’s key property. This value will replace the current value of this property on the Client for that object instance.

Transaction Server Update Step

A transaction server update step references a step definition within the same module that is run during transmit to update the back end system with the data captured by the transaction. This step has access to all of the properties of the transaction using the SDML or mechanisms available using the Agentry Java API. The value of these properties can be used by the steps to update the back end system. An update step can be defined to run or not run based on a data state being set for the transaction. An update step can also define a response to be sent to the Client to perform some additional action in relation to the object instance targeted by the transaction.

Using the data state functionality, update steps may be defined for a single transaction that process the data captured in the transaction normally, and other steps that run only when data states are set to provide data collision handling. Each server update step can contain its own list of selected data states, that is, the data states it is aware of. It can then be defined to run or not when one of its selected data states is set.

Server update steps can send a client response after they have been processed by the Agentry Server. This response will only be sent if the step that defines it is run. Only one response is sent for a transaction. There are different responses possible, and which one is ultimately sent to the Client is based on the type of response.

Server Update Step Attributes

Following is a list of the attributes for a server update step definition:

- **Step**: This attribute references the step definition within the same module to be run by the Agentry Server as a server update step for the transaction.
- **Run for which States**: This attribute defines when the step is run in relation to the transaction’s data states. This can be “All Data States”, “Data States except selected”,

“Only selected data states”, and “Do Not run Step”. This last option is normally only set for testing purposes, as the step will never be run if this option is selected. When set to one of the two data state options, a second tab is available in the Properties view of the Agentry Editor. This second tab lists all selected data states for the update step and allows for additional data states to be added.

- **Response to Client:** This attribute specifies what response is sent to the Agentry Client after the Update Step has been processed. The response defined here will only be sent if the update step is run. The responses that may be sent are “Delete Client Object”, “Replace Client Object”, “Update Client Key Property”, and “No Action Required”. If “Update Client Key Property” is set, the step being run by the server update step is expected to return a value identified as the transaction’s target object’s key property. This value will replace the current value of this property on the Client for that object instance.

Transaction Error Handling Steps

A transaction error handling step references a step definition that is run during transmit if an error occurs while the Server is processing the transaction. This includes errors returned by the data state or update steps. Error handling steps are run only when transaction failure handling is enabled, via a configuration option of the Agentry Server. An error handling step can respond to the Client to indicate the proper action to take in relation to the error that has occurred.

Error handling steps can perform multiple tasks to resolve such an issue. These include:

- Any post-error processing that may be necessary
- Setting the error fatality level
- Returning messaging to the Agentry Client for display to the user

One of the key components to transaction error handling steps is the error fatality. This term refers to the severity of the error and the proper way in which the transaction should be handled as a result of the error. This can include retrying the transaction, possibly after a change is made to it by the user, or removing the transaction from the Agentry Client and storing its data to the failed transactions queue on the Agentry Server.

Error handling steps may not need to be defined as a part of the transaction failure handling. The Agentry Server contains configuration options to set default behaviors, including the fatality level of an error. Error handling steps are normally defined to override these defaults where necessary.

Error Handling Step Attributes

- **Step:** This is the step definition within the module to be run as an error handling step for the transaction. The step referenced here should be defined to return data in the event of an error, or a specific type of error.
- **Error Type:** This attribute determines the behavior of the application when the error handling step returns true, indicating the error that occurred should be handled by the step. The options for this attribute are:

- *Fatal with Message* - The transmit will be aborted automatically and a message will be displayed to the user. The transaction will be removed from the Client and the data for it stored in the failed transactions queue on the Server.
- *Fatal without Message* - The transmit will be aborted automatically and no message will be displayed to the user specific to the transaction. The transaction will be removed from the Client and the data for it stored in the failed transactions queue on the Server.
- *No Change* - This selection will not change the error fatality for the transaction. Either another error handling step for the transaction will handle this, or the default fatality based on the error information returned by the back end system will remain. This is normally set for steps that either create messaging displayed to the user, or that perform other actions against the back end system to handle the error.
- *Retry with Change* - The user will be able to choose to abort the transmit and to change the data for the transaction. This requires transaction merging be enabled, as a new transaction will be instantiated by the user and it will then merge with the pending transaction as a result of an error. This will be an option for the user and, should the user choose not to retry, the transmit will continue. The transaction will be removed from the Client and saved to the failed transactions queue on the Server.
- *Retry without Change* - The user will be able to retry the transaction without editing the data it contains.
- **Step is true if:** This attribute controls whether data returned by the step is treated as a true or false return. When this attribute is true and the step returns data, this is treated as a true response.
- **If True:** This attribute defines whether or not the remaining error handling steps for the transaction should be run if the current error step returns true.
- **If False:** This attribute defines whether or not the remaining error handling steps for the transaction should be run if the current error step returns false.
- **Notification:** This Boolean attribute controls the external notification on the client device. If this attribute is true, a true result for the error handling step will result in the LED on the client device being activated and the transmit dialog flashing.
- **Sound:** This attribute defines whether or not the system default sound on the client device should be played when the error step returns true. It also controls the number of times to repeat the sound.
- **Interval:** If the **Sound** attribute is set to play the system sound two or more times, the interval attribute can be set to the number of seconds in between each time the sound is played.

Fetch

A fetch defines how the Agentry Server synchronizes data for a target object collection. This object collection must be a top-level collection within the module. A fetch is made up of steps that retrieve the data for the collection from the back end system. These steps are grouped into three categories within the Fetch definition: Client Exchange Steps, Server Exchange Steps,

and Removal Steps. A fetch may also include properties to store data captured from the user and validation rules for those property values.

A fetch may be a main or non-main fetch. A main fetch is processed during every transmit between the Agentry Client and Server. A given module may contain multiple main fetches. The order in which multiple main fetches, either within the same module or within multiple main fetches, are processed is undefined and should therefore not be a factor in the synchronization logic.

A non-main fetch will only be executed when an action step of type transmit explicitly defines such a fetch to be processed. Non-main fetches are normally defined to provide the search functionality to end users.

The basic structure of a fetch definition is intended to support the exchange data model of synchronization. This model is intended to allow for the synchronization of data in a more efficient manner, where only data changes on the back end system as compared to the current data on a given client are retrieved. Any data that has not been changed as compared to the client's data is not retrieved.

A fetch definition can be defined to retrieve new object instances to be added to a client application, replace existing objects on that client, a remove any objects the client should no longer store locally. The read steps of the object type targeted by the fetch are run after the fetch has been processed and may also retrieve objects for the client to either add them or replace existing instances.

Fetch Child Definitions

- **Property:** A fetch property defines data to be captured on the Agentry Client for use during fetch processing by the Agentry Server.
- **Validation Rule:** A fetch validation rule defines what rule definition will be used to validate the fetch's data and how failed validation is handled on the Agentry Client.
- **Client Exchange Step:** A fetch client exchange step defines how information about the target collection is processed by the Agentry Server.
- **Server Exchange Step:** A fetch server exchange step defines how information about the back end system's data is processed.
- **Removal Step:** A fetch removal step is defined to determine which objects should be removed from the collection targeted by the parent fetch.

Fetch Attributes

- **Collection:** This attribute references the object collection property within the same module and that is a direct child of the module main object for which the fetch will synchronize data. Steps executed by the fetch's child step usage definition will be processed by the Agentry Server in the context of this collection.
- **Name:** This attribute contains the name that identifies the fetch. This value must be unique among all fetch definitions within the same module.

- **Display Name:** This attribute contains the value that identifies the fetch on the client. This is displayed during synchronization in the Client's Transmit Screen when the fetch is processed by the Server.
- **Clear Collection:** This attribute specifies whether or not the object instances stored in the targeted collection should be removed from the Client prior to processing the fetch during synchronization. This attribute is normally only left set on when the fetch is either not using the exchange data model for synchronization, or when it is a non-main fetch performing search functionality and the previous search results should be removed from the client before performing a new search.
- **Main Fetch:** This attribute specifies whether the fetch is a main fetch. When checked, the fetch will be processed during every transmit between the Client and Server. When unchecked, the fetch will only be run when an action step of type Transmit explicitly lists the fetch to be processed and that action step is the one that initiates the transmit.

Fetch Validation Rule

A fetch validation rule defines what rule definition will be used to validate the fetch's data and how failed validation is handled on the Agentry Client. The rule referenced is called in a Boolean context and is expected to return true or false. False indicates failed validation, which may be treated as a warning or error. Messaging may be displayed to the user in relation to failed validation. An error requires the user to change the offending value(s) before proceeding. A warning displays an informational message giving the user the option to change the value(s), but does not require a change.

Not every fetch will have validation rules. Certain types of values do not need to be validated using a validation rule. Simple requirements such as the size of a string value or the minimum and maximum values of a numeric property can be enforced by the property itself. In other cases the data may simply not need to be validated.

Validation rules are used when more complex validation is required, such as when the valid value for a property is dependent on the value of a second property. Also, validation rules offer the flexibility to differentiate between a warning and an error. With a warning, the user is given the option of changing the value that violates the rule or leaving it as is. If treated as an error, the user must change the value before being allowed to proceed.

Validation rules are evaluated when the user clicks any navigation buttons in the wizard screen set displaying the fetch. If a validation rule references a property not yet displayed in the wizard, it will not fail validation. A false return by the rule is treated as a validation failure and the validation rule definition will then dictate the behavior of the Client.

Validation Rule Attributes

- **Rule:** This attribute specifies the rule definition within the same module to be used as a validation rule for the fetch. The rule is expected to return a Boolean value in the context of the current fetch instance.
- **Type:** This attribute can be set to either "Warning" or "Error" and determines how a false return from the rule is treated by the validation rule. Warning means a failed validation

does not require the user to change the value. The user will be displayed a message and given the option to change the value or keep it as set. An error type requires the user to change the value before proceeding.

- **Caption:** This is the text displayed in the title bar of the message for the validation rule.
- **Text:** This is the message displayed to the user when validation fails.
- **OK Label:** This is the text to label the OK button for the message screen on the client.
- **Cancel Label:** This is the text to label the Cancel button for the message screen on the client. This attribute is available only when the **Type** attribute is set to “Warning”.

Fetch Validation Rule Properties

Rule properties associate one or more object properties with a fetch validation rule. Rule properties are used to set the cursor focus on the Client when a fetch warning or error rule is triggered. The focus is set to the first property on the current screen set screen that is included in the Rule Properties list. If no match is found or if all of the listed properties are contained in a screen other than the current screen shown on the Client, no focus is set.

Validation rules and their associated rule properties are evaluated when the user clicks any navigation buttons in the wizard screen set displaying the transaction. If the rule returns false, the rule runs through all properties on the rule list until it finds one that matches a fetch property that is displayed on the Client screen. At that point, the ‘next’ or ‘finish’ process is stopped, the user remains on the same screen, and the cursor focus is set to the matching property. Any property controls that are hidden, disabled, or set to read-only will be ignored.

Validation rule properties are available on the following fetch types: Add, Edit, Complex Table Change, and Data Table Change.

Setting rule properties is optional. If no rule properties are set and a rule returns false, only the error or warning message associated with the rule is displayed on the Client.

Fetch Client Exchange Step

A fetch client exchange step defines how information about the target collection is processed by the Agentry Server. This definition references a step definition within the same module. This step has access to information about the target collection, as well as to any data captured in fetch properties. A client exchange step can be defined to execute once or iteratively, and can return data for an object collection. A fetch can contain multiple client exchange step definitions, which are processed by the Server in a defined order.

Though a client exchange step can return data to create and populate object instances, its intended purpose is to provide information about the current objects stored in the collection property targeted by the parent fetch (target collection) definition. A Client exchange step has access to the key property and last update value for each object instance in the target collection. This information is provided in support of the exchange data model. The intent is that the client exchange steps update this information to an exchange data object in the back end for later comparison to determine which data may need to be retrieved to update the Client.

Client Exchange Step Attributes

- **Step:** This attribute references the step definition within the same module to run as a client exchange step for the parent fetch.
- **Run:** This attribute specifies how to run the client exchange step during a single transmit. This may be set to one of the following values:
 - *Run one Time:* This setting will run the client exchange step a single time for the fetch processing. This setting assumes the step needs to be executed only once to return the data for all object instances to be added or replaced during synchronization, or the step being executed is not returning data but rather is being run in support of synchronization.
 - *Run Once per Object:* This setting will execute the client exchange step once for each object instance in the collection that is being synchronized. This includes both those object instances sent by the Client to the Server, as well as any that may have been added by previous fetch steps.
- **Read Into:** This attribute specifies the child or descendent object collection property of the target collection into which the data returned by the step should be read. This attribute has a default setting of “None”. This default means the data will not be read into a child collection but will instead be used to create object instances of the target collection. Other valid options for this attribute are any child collection properties of the target collection, or any descendent collections (e.g. collections within collections).

Fetch Server Exchange Step

A fetch server exchange step defines how information about the back end system’s data is processed. This definition references a step definition within the same module. This step has access to information about the target collection, as well as to any data captured in fetch properties. A server exchange step can be defined to execute once or iteratively, and can return data for an object collection.

The server exchange step definition is intended to perform one of two tasks within the exchange data model. First, it should compare information provided by the client exchange steps concerning which object instances the Client currently has and when they were retrieved to information in the back end system about when that same data was last modified or added. Second, it can then retrieve the data needed by the Client based on the differences found during this comparison. These tasks are normally accomplished by separate server exchange steps. Alternately or in addition to these definitions, the object read steps defined in the object type targeted by the fetch may retrieve data for the object instances.

Server Exchange Step Attributes

- **Step:** This attribute references the step definition within the same module to run as a server exchange step for the parent fetch.
- **Run:** This attribute specifies how to run the server exchange step during a single transmit. This may be set to one of the following values:

- *Run one Time:* This setting will run the server exchange step a single time for the fetch processing. This setting assumes the step needs to be executed only once to return the data for all object instances to be added or replaced during synchronization, or the step being executed is not returning data but rather is being run in support of synchronization.
- *Run Once per Object:* This setting will execute the server exchange step once for each object instance in the collection that is being synchronized. This includes both those object instances sent by the Client to the Server, as well as any that may have been added by previous fetch steps.
- **Read Into:** This attribute specifies the child or descendent object collection property of the target collection into which the data returned by the step should be read. This attribute has a default setting of “None”. This default means the data will not be read into a child collection but will instead be used to create object instances of the target collection. Other valid options for this attribute are any child collection properties of the target collection, or any descendent collections (e.g. collections within collections).

Fetch Removal Step

A fetch removal step is defined to determine which objects should be removed from the collection targeted by the parent fetch. A removal step references a step definition within the same module. This step has access to information about the target collection, as well as to any data captured in fetch properties. The step referenced by a removal step definition is expected to return the key property of any object(s) that should be deleted from the target collection on the Agentry Client.

Removal Step Attributes

- **Step:** This attribute references the step definition within the same module to run as a removal step for the parent fetch.
- **Run:** This attribute specifies how to run the removal step during a single transmit. This may be set to one of the following values:
 - *Run one Time:* This setting will run the removal step a single time for the fetch processing. This setting assumes the step need be executed only once to return the data for all object instances to be removed during synchronization, or the step being executed is not returning data but rather is being run in support of synchronization.
 - *Run Once per Object:* This setting will execute the removal step once for each object instance in the collection that is being synchronized. This includes both those object instances sent by the Client to the Server, as well as any that may have been added by previous fetch steps.
- **Read Into:** This attribute has no effect on a fetch removal step and will be deprecated in a future release.

Transaction and Fetch Properties

A transaction property defines a value to be captured by a transaction. Definable behaviors include the initial value for the property, the object property or table record field it targets, as

well as data-related behaviors. These include minimum and maximum values, a special value, and similar settings. These last behaviors will vary depending on the data type of the property.

A fetch property defines data to be captured on the Agentry Client for use during fetch processing by the Agentry Server. A fetch that contains properties is normally displayed in a screen set to allow the user to enter the desired values. The steps of the fetch then have access to these property values for use during synchronization. The fetch properties themselves define the data types of the values, and the initialization values when the fetch is instantiated.

Both transaction and fetch properties contain attributes related to initialization. These attributes are a part of all transaction and fetch property definitions regardless of the property data type. These attributes are in addition to the data type specific attributes.

For both a fetch and a transaction property, the purpose is to capture data on the Client. How this data is used depends on the property's parent. A transaction property's value will be copied to the object property it targets when the transaction is applied. This value will then also be available to the steps used by the transaction during synchronization and, depending on the defined processing, will likely be updated to the back end system.

A fetch property will be stored with the fetch and sent to the Agentry Server during synchronization. This will make the value available to all steps run by the fetch. However, the fetch property value will not affect the object property, as fetch properties do not modify object instances on the Client.

Transaction and Fetch Property Attributes

The following list of attributes are specific to properties defined for a transaction or fetch. These attributes are common to all properties regardless of data type:

- **Object Property:** This attribute specifies the object property targeted by the transaction or fetch property. This value may be used for initialization. For a transaction, this is also the object property the transaction property will set when the transaction is applied.
- **Initial Value:** This attribute specifies the data source to initialize the property. This may be the object property targeted by the transaction or fetch property, the property of a different object not targeted by the fetch or transaction, a constant value, or via a rule. When a rule is used, the rule may be evaluated before or after data entry.
- **Constant:** This attribute is enabled only when **Initial Value** is set to "Constant". The **Constant** attribute then contains the constant value to which the property will be initialized whenever the parent transaction or fetch is instantiated on the Client. This may be left blank for many property data types to initialize the property to null.
- **Rule:** This attribute is enabled only when the **Initial Value** attribute is set to either "Rule - before data entry" or "Rule - after data entry". It contains a reference to the rule definition to be evaluated to initialize the property.
- **Other Property:** This attribute is enabled only when the **Initial Value** attribute is set to "From a different object property". **Other Property** then contains the target path to the object property whose value will be used to initialize the property.

Property Data Types

The property data type definition can be a child to an object, transaction, or fetch definition. A property is defined to be a certain data type when it is created. This data type then specifies the type of data and its behavior within the property. The data types range from primitive types common to most or all development platforms, to more robust types that in other languages would be created by developers as classes, structures, or objects depending on the tool or language in use.

Following is a brief description of each property data type available in Agentry:

- **Boolean:** The Boolean property data type stores a true or false value.
- **Collection:** The collection property data type is defined to store multiple object instances of the same type as a property of a parent object, transaction, or fetch.
- **Complex Table Selection:** The complex table selection property type is used to store a selection made by the user from a complex table.
- **Data Table Selection:** The data table selection property type is used to store a selection made from a data table.
- **Date:** The date property type is used to store a calendar date value.
- **Date and Time:** The date and time property type stores a value consisting of a calendar date and time of day.
- **Decimal Number:** The decimal number property data type stores numeric value with a fractional component.
- **Duration:** The duration property data type is used to store a duration of time.
- **External Data:** An external data property stores a reference to a file stored on the client device's file system and that is external to the production data of the application.
- **Identifier:** The identifier property data type stores a non-negative integer value that is a unique identifier for an object.
- **Image:** The image property stores a still picture or other image captured on the client device from either the device's camera or selected from the file system.
- **Integral Number:** An integral number property stores whole numbers.
- **Location:** A location property stores a location value returned by a GPS unit that includes the latitude, longitude, dilution, and number of satellites.
- **Object:** The object property data type stores an object instance as a property of a parent definition.
- **Signature:** The signature property type stores a signature entered by a user on the Agentry Client.
- **String:** The string property data type stores any character values as a single string.
- **Time:** The time property data type stores a time of day value.

Boolean Property Type

The Boolean property data type stores a true or false value. When a Boolean property value is set, a null value is treated as false and any other value is treated as true.

The attributes for a Boolean property include the true and false value. These values define what will be displayed when the property contains a true or false value.

Boolean Property Attributes

Note: This property type does not have Special Value attributes.

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.
- **True Value:** This attribute contains the value to display when the Boolean property is set to true.
- **False Value:** This attribute contains the value to display when the Boolean property is set to false.

Collection Property Type

The collection property data type is defined to store multiple object instances of the same type as a property of a parent object, transaction, or fetch. The object type used in a collection property must have a defined key property to uniquely identify each object instance within the collection property. The default initialization for a collection property is an empty collection.

Each object instance within a collection is considered a child instance to the parent definition of the collection property. In objects, collection properties are commonly used to store object instances within a module to provide a data structure within the module representing the relationship between the different business entities for the module. Collection properties defined in the module main object are commonly referred to as “top-level collections”. Collection properties defined within an object other than the main object are referred to as nested collections.

The collection property type may also store other data types. However, in practice there is limited use for this type of definition. A collection defined to store another collection is not valid.

Collection Property Attributes

Note: This property type does not have Special Value attributes.

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.

- **Property Type:** This attribute specifies the type of property the collection will contain. The default and most common setting for this attribute is Object. There are limited use cases for collections storing instances of any other type of data.
- **Object:** This attribute is available when **Property Type** is set to “Object”. It lists all object definitions within the module and the selection made specifies the type object instances the collection property will contain. The object type to be stored in the collection property must have been defined previously and must have its **Key Property** attribute set prior to selecting it in the **Object** attribute field of the collection property definition.

Complex Table Selection Property Type

The complex table selection property type is used to store a selection made by the user from a complex table. The value stored in a complex table selection property is the key field of the selected record within the complex table. The data type of this value will be a string, integral number, or decimal number, based on the data type of the key field.

The complex table selection contains a single attribute specific to the data type named complex table. The setting of this attribute specifies the complex table definition that is the source for the property.

The value contained within a complex table selection property requires a brief explanation of complex tables. Complex tables are made up of records. The records are made up of multiple fields. Within the complex table definition, indexes are defined on the fields to allow users to search the table. Each complex table is required to contain a unique index, which is defined for the field that contains the unique value for each record. The complex table selection property will contain the value of this field for the record selected by the user.

Complex Table Selection Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.
- **Complex Table:** This attribute specifies the complex table definition within the application that the property will use. Only selections from the complex table specified here can be stored in the property. The complex table definition must exist and contain at least one field and the primary index before it may be selected for the **Complex Table** attribute of the property definition.

Data Table Selection Property Type

The data table selection property type is used to store a selection made from a data table. The value stored in a data table selection property is the code field of the selected data table record. This value will always be a string data type.

The data table selection property type includes display options for its value within the definition. Whenever this property type is displayed, the entire data table record may be displayed for its code. Also, only the code field or the value field may be displayed. Which is shown on the client is defined within the data table selection property definition.

It is important to note that it is not a requirement that a value selected from a data table be stored in a data table selection property. It is only one of the options available, and other property data types may be used for this purpose.

Data Table Selection Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.
- **Data Table:** This attribute specifies the data table from which the selection will be made for the value stored in the property. Only selections for the data table definition chosen here can be stored in the property. The data table definition selected here must exist within the application prior to defining this property type.
- **Display Type:** This attribute specifies how the selected data table record stored in the property will be displayed. The options are to display the code field, value field, or code and value field of the selected data table record; or to specify format text.
- **Format Text:** This attribute is available only when Display Type is set to “Format Text.” It specifies the format string to display the selected data table record stored in the property. This attribute may contain any printable characters plus the format strings %code and %value.

Date Property Type

The date property type is used to store a calendar date value. This value is stored internally as the number of days before or after the Agentry epoch date of January 1st, 2001. Negative values reflect dates prior to epoch. A date property is displayed on the Agentry Client in the format MM/DD/YYYY by default.

The date property may also be unset or invalid. In this case the year portion of the date property is set to zero (0000). This condition may be checked to determine if the date property has been set.

Date Property Type Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.

- **Blank:** This attribute specifies whether or not a blank value is displayed for a date property when it has not been set (year is zero).

Date And Time Property Type

The date and time property type stores a value consisting of a calendar date and time of day. This value is stored internally as the number of seconds before or after the Agentry epoch date of January 1st, 2001 12:00:00 am. Negative values reflect dates prior to epoch. A date and time property is displayed on the Agentry Client in the format MM/DD/YYYY HH:MM:SS am/pm by default.

A date and time property may contain an unset or invalid value. This is indicated by the year portion of the value, which is set to the year zero (0000). This condition may be checked to determine if the date and time property is invalid.

Date And Time Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.
- **Blank:** This attribute specifies whether or not to display a blank field when the date and time property does not contain a valid date and time (year is 0000).
- **Display Order:** This attribute specifies the order in which to display the date and time components of the property value. This may be set to either “Date - Time” or “Time - Date”.
- **Time Zone Adjust:** This attribute specifies whether or not to adjust the date and time value of the property during synchronization based on differences in time zones. For object properties the value retrieved from the back end system may be adjusted from the back end systems local or standard time, or from universal time, to the client device’s time zone. For transaction properties the date and time value can be adjusted from the client device’s time zone to the local or standard time of the system connection or to universal time. The default for this attribute is “Do not adjust”, which will not modify the date and time value during synchronization.

Decimal Number Property Type

The decimal number property data type is used to store a numeric value with a fractional component. The definable behaviors of a decimal number property include standard or NIST rounding, precision, and significant digit math options.

Values stored in a decimal property can contain values with a precision of up to 20 places past the decimal point. The values may be positive or negative.

Decimal Number Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.
- **Minimum Value:** This is the smallest value that may be contained within the property. This attribute and the **Precision** attribute are interdependent. You cannot specify a minimum value with more decimal places than is specified in the **Precision** attribute. To force the value to be positive, set the this attribute to 0. The minimum value specified here can be no greater than the defined **Maximum Value**.
- **Maximum Value:** This is the maximum value that the decimal property can contain. This is dependant on the **Precision** attribute. You cannot specify a maximum value with more decimal places than is specified in the **Precision** attribute. The value defined here can be no less than the defined **Minimum Value**.
- **Precision:** This attribute specifies the maximum number of places past the decimal point. A negative precision indicates places before the decimal, with any values past this point padded with zeroes. A precision of 0 specified whole numbers only, though consider using an integral number property for this purpose.
- **Blank** - This is a Boolean attribute that specifies whether to display a blank for the property when it has a value of 0.
- **Math:** This is a Boolean attribute that specifies whether or not to use significant figure math in any calculations that use the property value.
- **Rounding:** This attribute specifies the rounding method to use when this value is rounded. This may occur within rule definitions (ROUND function term) or when calculations involving this property are performed. The resulting value for the property will be rounded to the defined precision, as well as based on the significant digits operations. The methods for rounding are **Nearest** or **NIST**. Nearest is the typical rounding method in which the digit immediately after the digit to be rounded determines value of that rounded digit. Values below 5 leave the digit unchanged. Values 5 or above increment the rounded digit by 1. The NIST rounding method rounds values according to the rules set forth by the National Institute of Standards and Technology, specifically as they relate to calibrations measurements.

Duration Property Type

The duration property data type stores a duration of time. The value of a duration property is stored in seconds and may be positive or negative. It is possible to convert the value to other time units, including hours, minutes, or milliseconds when referenced in a step definition. This behavior is controlled by the definition of the duration property.

This data type does not store fractional seconds. During downstream synchronization, if the back end units for this property include precision smaller than whole seconds, the fractional

second portion of the value will be truncated when assigned to the property. The logic of the synchronization step should round the value prior to returning it to the Server if this is not the desired behavior. If it is necessary to keep the fractional portion of the duration value during synchronization, a decimal number property should be used.

Duration Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.
- **Minimum Value:** This attribute specifies the minimum duration value the property will accept. This is set in hours, minutes and seconds and is converted to total number of seconds within the property. This value can be no greater than the defined **Maximum Value**.
- **Maximum Value:** This attribute specifies the maximum duration value the property will accept. This is set in hours, minutes and seconds and is converted to total number of seconds within the property. This value can be no less than the defined **Minimum Value**.
- **Display Format:** This attribute specifies the format in which to display the duration value. The options are Fractional Hours, Hours:Minutes:Seconds (“H:M:S”), Minutes:Seconds (“M:S”), or Hours:Minutes (“H:M”).
- **Back End Units:** This attribute specifies the units in which the duration value is stored in the back end system. For object properties the value returned from the back end system will be converted from the unit selected here to seconds. For transaction and fetch properties the value will be converted from seconds to the units specified here.

External Data Property Type

An external data property is used to reference a file stored on the client device’s file system. This file is external to the application’s production data. The file data itself is not stored with the production data. This property type is normally used in conjunction with the file transfer functionality. The default display value of an external data property is the full path and file name of the referenced file.

For object properties the attributes of this property type related to the location for the file specify where the file will be stored on the client device when retrieved from the back end system. For transaction properties these same attributes specify the default location from which the user should make a selection. The file dialog opened in this case does allow the user to navigate the file system to select the desired file. Two separate paths can be defined for the external data property, one for client devices running the Windows PC group of operating systems, and a second for client devices running supported versions of the Mobile Windows OS’s.

The attributes of this data type also allow for designating whether the file should be read-only on the client device, the file extension for the file, and whether or not to delete the file when the parent object to the property is deleted.

The recommended use for this property type is define an object that represents the document and includes this property, as well as other information about the file. The external data property itself will reference the location of the file and can return the file's full path and name, just the file name, just the file path, just the file extension, as well as metadata about the file such as its last modified date and time and whether or not it has been modified since it was downloaded to the client device. Many of these values are exposed via rules and/or format strings. A separate property of a data type other than external data must exist and be referenced by the external data property that contains the name the file will be given when saved on the client device. This value must be set during synchronization prior to transferring the file itself.

External Data Property Attributes

Note: This property type does not have Special Value attributes.

General Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.

Client File Attributes

- **File Name:** This attribute references a property within the same parent definition as the external data property. The referenced property's value will be used as the file name for the file referenced by the external data property when it is downloaded from the back end during object synchronization. In a transaction, the File Name property will store the name of the file as saved on the client device.
- **File Extension:** This attribute is optional and can contain the file extension for the file referenced by the external data property. For objects this extension will be appended to the file when it is downloaded from the back end and saved on the client device. For transactions this extension will be used to filter the options displayed to the user in the file dialog when selecting the file. Only files with the same extension will be displayed. If this attribute is not set, files saved to the Agentry Client during object synchronization will keep the same file extension as provided by the back end. For transactions, all files will be listed in the File Dialog to the user, regardless of file extension.
- **When Object is Deleted:** This attribute specifies what to do with the file referenced by the external data property when the parent object of the property is deleted from the Agentry Client. The options are to always delete the file, never delete the file, or only delete the file when retrieved from the back end by the mobile client application. This last option will exclude files attached locally on the Agentry Client via a transaction. This option is

unaffected by the Read Only attribute, meaning if this setting results in the file being deleted, it will be removed regardless of whether or not it is read-only.

- **Read Only:** This attribute specifies whether or not the file's read-only switch will be true. When set, this will prevent the user from modifying the file but will not prevent the Agentry Client from deleting or otherwise accessing the file.
- **Use Most Recent Location:** This attribute specifies whether, when selecting a file on the Agentry Client to be referenced by the external data property, the file dialog displayed should be opened to the most recently selected folder, or to the default folder regardless of the previous selection made.

Filter

- **File Filter:** This attribute specifies the file type that may be selected or referenced by the external data property.
- **File Filter Description:** This attribute allows for the specification of a file description to be associated with the file extension listed in the File Filter attribute.
- **Restricted Files:** This attribute allows for the specification of file names or file extensions that may not be selected. Multiple files or file types can be listed here separated by semi-colons.

Windows 9.x/NT/2000/XP

- **Base Path:** This attribute specifies the base path to which the file will be saved (objects) or the default location the user will be displayed in the file dialog to select a file (transactions). This attribute is for Windows PC operating system builds for PC's, laptops, and tablets. This may or may not be the entire path for the application, dependent on the Relative Path attribute. Options for this attribute include:
 - **Absolute Path:** This selection will result in the value of the Relative Path attribute being used and is assumed to contain the full path, including drive letter, for the files location.
 - **Application Data:** This selection will set the file's location to be the path configured in Windows to be the location for application data.
 - **My Documents:** This selection will set the file's location to be the path configured in Windows to be the user's My Documents folder.
 - **My Pictures:** This selection will set the file's location to be the path configured in Windows to be the user's My Pictures folder.
 - **Program Files:** This selection will set the file's location to be the path configured in Windows to be the Program Files folder.
 - **Windows Temporary Directory:** This selection will set the file's location to be the path configured in Windows to be the Windows TEMP folder.
- **Relative Path:** The value of this attribute will be appended to the path resulting from the Base Path attribute setting. If Base Path is set to Absolute Path, the value of Relative Path will be used as the full path for the file's location.

Windows CE (Mobile Windows versions)

- **Use Path:** This attribute, when checked, will use the same path as defined in the Windows 9.x/NT/2000/XP set of attributes. This will disable the Base Path and Relative Path attributes for mobile devices.
- **Base Path:** This attribute specifies the base path to which the file will be saved (objects) or the default location the user will be displayed in the file dialog to select a file (transactions). This attribute is for Mobile Windows operating system builds. This may or may not be the entire path for the application, dependent on the Relative Path attribute. Options for this attribute include:
 - **Absolute Path:** This selection will result in the value of the Relative Path attribute being used and is assumed to contain the full path, including drive letter, for the files location.
 - **My Documents:** This selection will set the file's location to be the path configured in Windows to be the user's My Documents folder.
 - **Program Files:** This selection will set the file's location to be the path configured in Windows to be the Program Files folder.
 - **Windows Temporary Directory:** This selection will set the file's location to be the path configured in Windows to be the Windows TEMP folder.
- **Relative Path:** The value of this attribute will be appended to the path resulting from the Base Path attribute setting. If Base Path is set to Absolute Path, the value of Relative Path will be used as the full path for the file's location.

Identifier Property Type

The identifier property data type is used to store a non-negative integer value that is a unique identifier for an object. The intent of this data type is to be used as a key property for an object. This is not a requirement and a property of a different data type may be used as an object key property.

Identifier Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.
- **Blank:** This attribute specifies whether to display a blank value or 0 when the identifier property has not been set.

Image

The image property stores a still picture or other image captured on the client device from either the device's camera or selected from the file system. This property type is provided as a part of the overall image capture functionality that may be implemented in the mobile application. This property should only be displayed in detail screen fields with an edit type of image capture.

This property type will simply store an image captured from the client device's camera or selected from the device's file system. Its contents can be displayed to the user in detail screen fields of type image capture. It has no attributes beyond the standard property attributes. For an object these are the name and display name. If the parent is a transaction, which it should be in most cases, the standard transaction property attributes are set, including name, display name, the initial value attributes, and optionally special value attributes.

To synchronize data for an Image property, the file document management step type can be used to store the image as a .jpg file on the file system of the Agentry Server. Also, SDML data tags can be used to access the image data within other step types.

Integral Number Property Type

The integral number data type stores a whole number. An integral number property can define the minimum and maximum values it can contain. The hard minimum and maximum limits for this data type are equivalent to a 32-bit value, allowing for a positive/negative indicator bit.

Integral Number Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.
- **Minimum Value:** This attribute specifies the minimum value accepted by the property. This attribute can be no greater than the defined Maximum Value.
- **Maximum Value:** This attribute specifies the maximum value accepted by the property. This attribute can be no less than the defined Minimum Value.
- **Blank:** This attribute specifies whether to display a blank value or 0 when the value of the property is zero.

Location Property Type

A location property stores a location value returned by a GPS unit that includes the latitude, longitude, dilution, and number of satellites. The location property value can be invalid if the parameters of the property definition are not met. The transaction property location type includes attributes to define these parameters for the location value. The location property may also be set via rule functions that take the latitude and longitude values, converting them to a location value.

When defining this property type there are certain attributes specific to it for transactions and fetches versus objects. The object location property will contain attributes to initialize the value with a latitude, longitude, position dilution, and satellite count.

For a transaction or fetch location property, these same attributes can be set. In addition to these, there are also attributes to specify what is considered the minimum requirements for a valid location value for that property. These attributes set the minimum number of satellites,

and the maximum age and position dilution for a location value returned from the GPS unit. If these minimums are not met, the behavior is definable within the property. The value can still be accepted, or it can be rejected.

For transaction and fetch location properties, there also exist the common initialization attributes. These attributes will override the defined latitude, longitude, position dilution, and number of satellites values for the property.

Location Property Attributes

- **Name:** Contains the internal unique name for the property definition. This value must be unique among all properties within the same parent definition.
- **Maximum Reading Age:** This attribute specifies the maximum reading age in seconds for the value returned by the GPS unit. This reading age represents the last time the unit took a reading. The Maximum Reading Age will dictate the oldest allowable reading for the location property. A location with a reading age older than the one specified in this attribute will be considered an invalid location.
- **Minimum Number of Satellites:** This attribute specifies the minimum number of satellites used to calculate the location. There is a minimum of 3 satellites required for any GPS location. A higher minimum may be specified. Note that this differs from the number of satellites the unit can see. This value specifies the number actually used to calculate the location. If this number is less than the minimum number specified the location will be considered invalid.
- **Maximum Position Dilution:** This attribute specifies the maximum acceptable position dilution for a location returned to the location property. This is an integral number with a range of values from 1 through 50, inclusive. If the position dilution returned with the location value exceeds this maximum the location will be considered invalid.
- **Accept Invalid Data:** This attribute specifies whether or not a location value that does not meet the criteria set for a valid location value to be accepted. If this attribute is set, invalid locations will be accepted. The property will return an invalid location value, which may be checked using the rule function term @IS_VALID_LOCATION.

Transaction and Fetch Attributes - The standard fetch and transaction attributes for initializing the property and targeting object properties are available for the Location property type. The attribute Initial Value includes the normal available settings plus the options listed below, which are specific to the Location type.

- **Current Location After Data Entry:** This option specifies the property should be updated to the device's location after the transaction has been finished and just before it is applied. This value is obtained from the device's GPS unit.
- **Current Location Before Data Entry:** This option specifies the property should be updated to the device's location before the transaction is displayed to the user. This value is obtained from the device's GPS unit.

Object Property Type

The object property data type is used to define an object as a property. The object property type stores a single object instance of a defined type as a property of a parent object, transaction, or fetch.

Object Property Attributes

Note: This property type does not have Special Value attributes.

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.
- **Object:** This attribute specifies the type of object the property is to contain. The object selected here must already exist within the same module before the property is defined.

Signature Property Type

The signature property type stores a signature entered by a user on the Agentry Client. This is an actual, written signature that can be entered on the device using a stylus or some other electronic pen. This signature is stored internally as a bitmap image. Normally only transaction definitions contain signature properties.

Signature properties may not be initialized to the value of another property. Also, it is outside the normal usage to target an object property with a transaction property of type signature. The primary intent of the signature property is to capture a signature in bitmap format on the client device and to then transfer that bitmap image to the back end system as a part of the transaction's synchronization processing.

This property type includes definable behaviors covering the control that will display the property, and the minimum height and width of the bitmap image captured to treat as a valid signature.

This property type has several associated SDML data tags for accessing its bitmap data. The information on these should be reviewed when working with this property type.

Signature Property Attributes

Note: This property type does not have Special Value attributes.

General Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.

- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.
- **Required:** This attribute specifies whether or not the signature is required. For a transaction when this attribute is true, the user will not be able to advance the wizard until the signature has been captured. This includes meeting the *Minimum Signature Size Requirements* attribute settings.
- **Time and Date:** This attribute specifies whether to embed the client device's current date and time in the image.
- **Update Rule:** This attribute references a rule definition, the return value from which is expected to be a string. This value will be embedded in the bitmap image with the signature.
- **Signed:** This attribute contains the text value displayed in the detail screen field targeting the property when the signature has been captured.
- **Get Signature:** This attribute contains the text value displayed in the detail screen field targeting the property before the signature has been captured.

Maximum Window Size

- **Height:** This attribute specifies the maximum height of the window, in pixels, where the signature is entered.
- **Width:** This attribute specifies the maximum width of the window, in pixels, where the signature is entered.

Minimum Required Signature Size

- **Height:** This attribute specifies the minimum height, in pixels, for the signature value. If the signature does not meet this minimum, the signature will not be accepted. If the signature is required, the user will not be able to advance the wizard until the signature has been entered with this minimum height.
- **Width:** This attribute specifies the minimum width, in pixels, for the signature value. If the signature does not meet this minimum, the signature will not be accepted. If the signature is required, the user will not be able to advance the wizard until the signature has been entered with this minimum width.

String Property Type

The string property data type stores any character values as a single string. Definable behaviors of a string property include the ability to word wrap its contents upon display, to trim leading or trailing spaces within the string, and to treat the value as a password.

String Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.

- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Agentry Client. This will also be the default label for any screen control that displays this property.
- **Case:** This attribute specifies the case of the characters within the string property. This also can affect how multiple parent definitions, such as objects, are sorted based on the value of the property. The options for this attribute are: Lowercase Only, Uppercase Only, Mixed (case-insensitive), and Mixed (case-sensitive). The lower and uppercase settings will force any characters within the property to either lower or uppercase, respectively. The two Mixed case options will preserve the case of the characters as entered. The case-sensitive and case-insensitive settings specify how the value of the property is compared, either with respect to case or ignore it.
- **Format:** This attribute specifies how the value contained within the string property should be treated by the device or, more specifically, the operating system of the device. The options available for this attribute include email address, telephone number, and URL. Specifying this one of these options will result in the value being passed to the operating system with instructions to “open” the value in the corresponding application for the selected format; e.g. specifying the option URL will open the device’s web browser and navigate to the value in the string property.
- **Minimum Length:** This attribute specifies the minimum number of characters the property will accept. For transaction string properties, the user will not be able to advance the wizard unless this minimum number of characters is entered or set for the property.
- **Maximum Length:** This attribute specifies the maximum number of characters the property can contain. Editable fields displaying this property will not allow the entry of more than this number of characters. Object properties will truncate any value to this maximum number of characters for the string property.
- **Carriage Return:** This attribute will affect properties that are displayed in fields with multiple lines. If set to true, when the user hits the Enter key, or if a carriage return value exists in the string, a new line will be started within the multi-line field.
- **Word Wrap:** This attribute is another that affects properties displayed in multi-line fields. When set to true, if the text contained within the property is longer than the width of the field in which it is displayed, it will automatically wrap to the next line of the field, rather than scrolling past the far right edge.
- **Password:** This attribute controls whether the value entered for the property should be displayed or hidden. When set to true, the value for the property will not be displayed, but rather each character will be replaced by an asterisk (*). This is also true when users enter a value for this property.
- **Trim:** This attribute specifies whether white space characters at the beginning or end of the string should be preserved. When set to true, any leading or trailing white space will be trimmed from the value. Any white space within the string will not be trimmed.

Time Property Type

The time property data type stores a time of day value. This value is stored internally as the number of seconds after midnight, with midnight itself represented as 0. The default display format of a time property is HH:MM:SS am/pm.

Time Property Attributes

- **Name:** This is the unique identifier for the property definition. This value must be unique among all properties within the same parent definition.
- **Display Name:** This attribute sets the default display name to label or otherwise identify the property value on the Client. This will also be the default label for any screen control that displays this property.

Push

A push defines when it is necessary to push an object from the back end system to the Agentry Client and how that object's data is retrieved. A push provides real-time data synchronization for server-to-client data transfer, targeting a top-level object collection property within the same module. The push determines if changes have been made to the back end system and also retrieves the needed data to send those changes to the client. Part of the push definition is the optional behavior to notify users when data has been pushed to their clients.

There are five child definitions to the push, each a step usage definition. These steps provide the behaviors of polling the back end system for data changes that include new or modified business objects, polling and retrieving data for objects to be removed from clients, steps to retrieve data when changes have been found during a poll, updating the back end system after the new objects have been processed, and error handling steps.

The push itself defines how often to poll the back end system for changes, whether or not to display notifications on the Client of data after it has been pushed down and the nature of that notification, and an optional action that may be executed for each object pushed down to a Client.

The Push definition is the primary definition for implementing push behavior, but it is not the only definition type involved. The application-level definition transmit configuration also plays a part in this behavior. Specifically, a transmit configuration must be defined to maintain a constant connection between the Agentry Client and Agentry Server.

Additionally, if the system connection to be used for push processing is a SQL Database connection type, the configuration file `SqlBE.ini` for the Agentry Server is likely to need modification. Two sections within this file, `EnablePushUser` and `DisablePushUser` are processed by the Server as a part of the overall push processing for a database system. For other system connection types this file is not involved. Information on this configuration file can be found in the *Agentry Implementation Guide* for both Windows and Linux.

Push Child Definitions

- **Retrieval Step:** A push retrieval step references a step definition run to determine if object data has changed in the back end system and how that data is retrieved.
- **Removal Step:** A push removal step references a step definition run to determine what objects should be removed from the collection on the Agentry Client.
- **Read Step:** A push read step references a step definition to be run to continue the data retrieval for the object collection targeted by the push.
- **Response Step:** A push response step references a step to be run after the Agentry Server receives notification from the Agentry Client that an object has been successfully pushed down.
- **Error Step:** A push error step references a step definition to be executed when one of the other step usage definitions within the push return an error.

Push Attributes

General Attributes

- **Collection:** This attribute specifies the target object collection for which the push will synchronize data. This must be a top-level object collection property within the same module as the push definition.
- **Name:** This is the internal definition name for the push. This must be unique among all push definitions within the same module.
- **Display Name:** This is the default display value for the push definition when reference is made to it on the Agentry Client.
- **Poll Interval:** This attribute contains a duration value in hours, minutes and seconds, specifying how often to poll the back end system for modifications. At each poll interval the push retrieval steps and push removal steps will be processed by the Agentry Server. The value of the **Poll Interval** must be less than the transmit configuration attribute **Inactive Time** attribute of the transmit configuration defined to support the push behavior.
- **Read Steps:** This attribute specifies whether to use the read steps defined in the object type for the push's target collection property. When this attribute is checked, the read steps in the object type will be processed to synchronize data rather than the push read steps.
- **Queue Messages:** This attribute allows for push messages from the Agentry Server to the Agentry Client to be queued if they are not successfully sent after the first attempt or if the Agentry Client indicates it is still processing the previously received message.

Notification Attributes

- **Dialog Pops Up:** This attribute specifies whether or not a notification dialog is displayed on the Agentry Client after an object has been pushed down to the Client and when that dialog should be displayed. The options for this attribute are:
 - *After all data received:* This setting will display the notification after all objects have been successfully pushed to the Client.

- *Immediately*: This setting will display the notification after each object has been successfully pushed to the Client. If multiple objects are pushed based on a single poll, the notification dialog will be displayed once for each object.
- *No Dialog (sound only)*: This setting will not display any notification dialog to the user, with the client's default system sound being the only one played.
- *When user clicks icon*: This setting will display an icon on the Agentry Client after objects have been pushed down. The notification dialog will then not be displayed until the user clicks this icon.
- **Data Received Text**: This attribute specifies whether or not to display notification text for new or replaced object instances and, if enabled, the contents of the notification message. This includes both the message to display to the user and the text in the notification dialogs title bar.
- **Data Removed Text**: This attribute specifies whether or not to display notification text when object instances are removed and, if enabled, the contents of the notification message. This includes both the message to display to the user and the text in the notification dialogs title bar.
- **Rec'd & Rem'd Text**: This attribute specifies whether or not to display notification text when both new object instances received from the push and other object instances are removed. If enabled, the contents of the notification message are also a part of this attribute setting. This includes both the message to display to the user and the text in the notification dialogs title bar.
- **Notification**: This attribute enables or disables the external notification behavior. When enabled, the client device's hardware LED light will be activated by the Agentry Client. This attribute has no effect on client devices without such hardware.
- **Play Sound**: This attribute enables or disables the system's default sound for notification when objects are synchronized by the push. When this attribute is enabled, the option for how many times to play the sound is set. This will also enable the Interval Between Sounds attribute.
- **Interval Between Sounds**: This attribute is enabled when the Play Sound attribute is enabled and it specifies the sound be played multiple times. The Interval Between Sounds attribute then specifies the duration of time between each instance of the sound.

Action Attributes

- **Action After Object Received**: This attribute references an action to execute when an object is pushed down to the Agentry Client. When an action is selected here, that action will be executed for each object instance pushed to the Client, targeting that object. This action is not executed when objects are removed based on the push synchronization. If the user is currently executing an action when an object is pushed to the client (e.g., the user is viewing a transaction wizard), the action defined for the push is queued and will be executed when the current action is completed.
- **Action When Push Completes**: This attribute references an action to execute when the push has finished pushing down all object instances to the Agentry Client. This action targets the object collection targeted by the push.

- **Cancel Action:** This attribute specifies whether or not any action currently being executed on the Agentry Client is cancelled when objects are pushed down. If this option is disabled and an action is being executed while an object is being pushed down to the Agentry Client, the object will not be received by the Agentry Client.

Push Retrieval Step

A push retrieval step references a step definition run to determine if data has changed in the back end system and how that data is retrieved. A retrieval step can be executed either once, or iteratively based on the number of users logged in to receive push data. A push retrieval step is run as a part of the back end polling performed by the push. The step referenced by a push retrieval step is expected to return the key property of any object instances to be pushed to the Agentry Client. It may also return additional property values for the object type. If defined to one run once per poll period it is also expected to return the Client user ID to which the object will be sent.

A push retrieval step can also return data for the object type targeted by the fetch, as well as for its child objects. However, it is recommended that child, or nested collections be synchronized by the push read steps or the object read steps (depending on how the push is defined). Since retrieval steps are run every poll period, the step definitions they use should be defined to perform the least amount of processing necessary to determine if new object data needs to be retrieved. Non-collection property values for the target object type can also be retrieved by the retrieval step in this model, but no additional data should be retrieved here for the sake of efficiency of the push's polling activity.

The retrieval step may be executed, or run, in one of two ways for a given poll. First it may be defined to run once per user currently logged in to receive push data. For this type of execution, the data returned by the retrieval step will be organized internally by the Server for each user. The step being run should then include logic that includes retrieving data specific to each user, matching the criteria for the implementation related to how objects are synchronized. The step itself will then be executed multiple times, once for each user, during a single poll of the back end. Note that this can be a significant number of executions in a production environment, where it is common for hundreds of users to be connected to the Server for push processing. When run in this manner, the user ID value for each user is accessible via the <<userID>> SDML data tag.

The second option for running push retrieval steps is to run them once per poll period. This behavior will run the step a single time for a given poll regardless of the number of users currently connected to the Server for push data. In this scenario, the data returned by the step must include the user ID specifying which client user will receive a given object. This value should be identified to the Server as UserID. Once the object instance has been created by the Server and the synchronization of the push overall is completed, the object instance will be pushed to that user. When run in this manner individual user ID's are not available to the step.

When a push retrieval step returns the key property of the push's target object type, the push read steps or the object read steps (depending on the definition of the push) will be run to

continue the synchronization of the target object collection. If no key property is returned by any push retrieval step, it is assumed no new data needs to be pushed to the Client. No read steps will be run in this situation.

Retrieval Step Attributes

- **Step:** This attribute references the step definition within the same module to be run as a push retrieval step. These steps may return values for any property within the targeted object type of the push, but must return the key property of that object to indicate that one or more objects should be synchronized by the push and sent to the Client. For steps run once, the data returned by the step must also include the Client user ID to which the object will be sent.
- **Run:** This attribute specifies how often to run the referenced step during a single poll interval. This may be set to either Run One Time, or Run Once Per User. The former will execute the step once for a given poll period and the user ID value is not available. The latter will execute the step once per user currently connected to the Server for push data during a given poll period and the user ID value is available.
- **Read Into:** This attribute specifies for which object within the data structure of the targeted object collection the step will return data. While retrieval steps can return data for nested collections, it is recommended that this be handled by the push read steps, as these will only be run when the retrieval steps indicate new data is needed.

Push Removal Step

A push removal step references a step definition run to determine what objects should be removed from the target collection on the Agentry Client. A removal step can be executed either once, or iteratively based on the number of users logged in to receive push data. A removal step is run as a part of the back end polling performed by the push. The step referenced by a push removal step definition is expected to return the key property of any object instance to be deleted from the Client. If defined to run once per poll period it is also expected to return the Client user ID from which the object will be removed.

A removal step may be run once per poll of the back end system, or once per user per poll period, depending on how it is defined. When a removal step is run once per user, data returned by the step will be organized according to each user. Note that this scenario can result in a large number of executions of the step per poll, as the number of users logged in into the Server is commonly in the hundreds or more in a production environment. When run in this manner, the user ID value for each user is accessible via the <<userID>> SDML data tag.

When a removal step is defined to run one time, it will be executed once per poll period, regardless of the number of users connected to the Server. In this situation, the data returned by the removal step should also include the user ID as entered on the Client, indicating which client will receive the key property for the object to be removed. It is recommended that this is how most, if not all removal steps are defined within a push as it is a more efficient model of data synchronization. When run in this manner individual user ID's are not available to the step.

Removal Step Attributes

- **Step:** This attribute specifies the step definition within the same module as the push to be run as a push removal step.
- **Run:** This attribute specifies how often to run the step in a given poll period. The options for this attribute are Run One Time or Run Once Per User. The former will execute the step once during the poll period and individual user ID's are not available. The latter will execute the step once per user currently connected for push data and individual user ID's are available to the step.
- *Read Into:* Not currently supported - leave set to default.

Push Read Step

A push read step references a step definition to be run to continue to read data for the target object collection. This is a continuation of the synchronization process begun by the retrieval steps for the push. A push read step can be executed once or iteratively based on the number of objects and number of users logged in to receive push data. A push read step is only run if the retrieval steps indicate their are objects to be retrieved.

The execution of the read step can be based on the number of objects created by the retrieval steps, the number of objects in a nested collection created by previous retrieval or read steps, or based on the number of users currently connected to receive push data.

When run once per user, the step will have access to individual user ID's via the SDML data tag `<<user.agentryID>>`. Also, any values stored in the `<<user.info>>` data tag are still available. When run in this manner it is important to note that the step will be executed once for each user during each poll period when the retrieval steps indicate there is data to be synchronized. For production systems it is not uncommon for the number of users connected for push data to be in the hundreds or more.

When run once per object, the read step will be executed for each object instance for the collection targeted by the push created by any other push steps prior to the current step's execution. When run in this manner the step will have access to the key property of the object for which it is currently executing.

When run once per collection object, the read step should target (Read Into) a nested collection of the collection targeted by the push. The step will then be executed once for each object instance within in this nested collection created by the push steps prior to the current step's execution. The step is expected to return property values for the object type stored in the nested collection. It should also return the key property of any object between the nested object and the top-level object type in the data hierarchy of the module. This configuration is primarily intended for file transfer functionality and it is recommended it not be used for other purposes unless no alternative is available.

When the read step is defined to run once per poll period, it is expected to return data for the object type in the collection it targets, which will either be the same as the push's target collection, or a nested collection of that target. It will not have access to individual user ID's or

to any object key properties. It must also return the Client user ID indicating to which user the object should be sent.

Read Step Attributes

- **Step:** This attribute specifies the step definition within the same module as the push to be run as a push read step.
- **Run:** This attribute specifies how often to run the step in a given poll period. The options for this attribute are Run One Time or Run Once Per User. The former will execute the step once during the poll period and individual user information is not available. The latter will execute the step once per user currently connected for push data and individual user information, including user ID's are available to the step.
- **Read Into:** This attribute specifies which objects to create with the data returned by the step. This may be the same collection as is targeted by the fetch, or one of its nested collections. To read data into the push's target collection, this attribute is left set to its default value of "None." For nested collections, the desired collection is selected in the Add Wizard or in the properties view.

Push Response Step

A push response step references a step to be run when the Agentry Server receives notification from the Agentry Client that an object has been successfully pushed down. This step is run to update the necessary back end objects that the object has been processed by the push. A push response step is always executed once per object pushed to the Client.

Response Step Attributes

- **Step:** This attribute specifies the step definition within the same module as the push to be run as a push response step.
- **Run:** This attribute cannot be changed for this step usage definition. This step type is always run once per object.
- *Read Into:* Not currently supported - leave set to default.

Push Error Handling Step

A push error step references a step definition to be executed when one of the other step usage definitions within the push return an error. An error step is always executed once per object.

The intended purpose of a push error step is to perform any cleanup or similar actions in the event an error occurs in processing on of the push's steps. This may include items such as marking the object data in the back end as not pushed, or other similar processing.

Error Handling Step Attributes

- **Step:** This attribute specifies the step definition within the same module as the push to be run as a push error step.

- **Run:** This attribute cannot be changed for this step usage definition. This step type is always run once per object.
- *Read Into:* Not currently supported - leave set to default.

Service Event

A service event defines how the Agentry Server synchronizes data between two back end systems. A service event will normally perform such a synchronization when a change or “event” occurs in a source back end system that must be reflected in a destination back end system. Depending on its type, a Service Event can either actively poll a back end system, or listen to the source back end for messages notifying it of a change. A service event targets an object collection to facilitate this data transfer, with the object instances in that collection storing the data retrieved from the source back end. The synchronization processing of a service event does not involve or affect any Agentry Clients.

The service event creates object instances based on data retrieved from the source back end system. It then updates this object data to the destination back end system. The components of the service event that retrieve data from the source back end system differ for each service event type. The child definitions to update the destination back end system are the same set of step usage definitions for all service event types.

There are four types of service events that may be defined:

- **Poll With Step:** A Poll With Step service event type references a step definition that is run by the Agentry Server periodically to actively poll the source back end system for data changes.
- **Java Callback:** A Java Callback service event type includes a Java code component that is an extension of the ServiceEvent Agentry Java API class that allows a the source back end system to call into the Agentry Server as a notification of a modification to that back end system’s data.
- **HTTP-XML Message Received:** An HTTP-XML message received service event type includes XML message mappings that will map messages sent from the source back end system to the Agentry Server to indicate data has changed in that back end system.
- **File System Monitor:** A File System Monitor service event type is defined to monitor a specified directory on the Agentry Server’s host file system for changes and includes document mappings to map data in that directory’s files to the properties of the service event’s target object type.

Service Event Child Definitions

While each service event type has different components to capture data changes in the source back end system, all service event types contain the same child definitions to update the destination back end system:

- **Read Step:** A service event read step references a step definition run to retrieve any additional data for the target object collection from the source back end system.

- **Data State Step:** A service event data state step references a step definition run to check for data collisions in the destination back end system before the service event makes any changes to it.
- **Update Step:** A service event update step references a step definition run to update the destination back end system with data stored in the service event's target object collection.
- **Error Handling Step:** A service event error handling step references a step definition run only when one of the other service event child step usage definitions returns an error.

Service Event Attributes

The attributes for a service event vary depending on the service event type. See the service event type-specific information for details on these attributes.

Service Event Type: Poll With Step

A Poll With Step service event type references a step definition that is run by the Agentry Server periodically to actively poll the source back end system for data changes. This step definition is commonly a SQL step, though this is not a requirement. The step polling the source back end must return the key property of the object for which the service event has been defined to indicate there is data to be updated to the destination back end system. When the step returns this value, the service event's read, data state, and update steps are processed.

This service event type is typically defined when the source back end is a SQL Database system connection, although any back end may be actively polled provided the correct step type definition is used. The only requirement of the step referenced by the service event to poll the source back end is that it return the key property for each object instance of the target object type to be synchronized.

This service event type includes the poll interval, which is the duration of time between polls of the source back end by the service event. The number of objects retrieved from the source back end can be limited to a maximum number of instances per poll interval.

Poll With Step Service Event Attributes

- **Name:** The internal unique name for the service event definition, must be unique among all service events defined within the same module.
- **Step:** This attribute references the step definition to be run by the service event to poll the source back end system. The step referenced here should be written to return the key property of the target object type of the service event whenever data has changed in the source back end.
- **Object Type:** This attribute specifies the target object type of the service event. The properties of this object will store the data retrieved from the source back end to be updated to the destination back end. This may include child object collections where necessary.
- **Poll Interval:** This attribute specifies the duration of time between polls of the source back end by the service event. This attribute is set in hours, minutes and seconds. The step

definition referenced in the Step attribute will be run periodically based on the value in the Poll Interval attribute.

- **Object Limit:** This attribute specifies the maximum number of object instances to create using the data returned by the service event's defined step. If this step returns the key property values for objects than specified here, the order in which the step returns them will determine which object instances will be created and which will not.
- **Read Steps:** This attribute specifies whether to retrieve data from the source back end system using the read steps defined in the service event, or those defined in the target object type of the service event. When this attribute is true, the object's read steps will be used and the service event's read steps will not be processed.

Service Event Type: Java Callback

A Java Callback service event type includes a Java code component that is an extension of the ServiceEvent Agentry Java API class. Included in this class must be a method into which the source back end system can the Agentry Server as a notification of a modification to that back end system's data. this class is instantiated when the service event is loaded by the Server during startup. The information passed to this method must include the key property of the service event's defined object type. When a message is received by this class that includes the service event's target object's key property the read steps, data state steps, and update steps will be processed by the Agentry Server to update the destination back end system.

The Java code component of this service event type is initially created as a skeleton class. The developer must then implement the methods for this class to process the message received from the source back end system. It may include additional methods for processing the message once received, but only one method within the class can be called by the source back end. Data captured from the source back end must then be passed from this class to the Agentry Server using the standards within the Agentry Java API.

Java Callback Service Event Attributes:

- **Name:** The internal unique name for the service event definition, must be unique among all service events defined within the same module.
- **Connection:** This attribute references a Java Virtual Machine system connection within the application. This system connection is the one to with which the service event will communicate. This connection will be the one over which the source back end system sends the message to the service event when a data change occurs. This system connection must exist prior to defining the service event and it must be of type Java Virtual Machine.
- **Object Type:** This attribute specifies the target object type of the service event. The properties of this object will store the data retrieved from the source back end to be updated to the destination back end. This may include child object collections where necessary.
- **Read Steps:** This attribute specifies whether to retrieve data from the source back end system using the read steps defined in the service event, or those defined in the target object type of the service event. When this attribute is true, the object's read steps will be used and the service event's read steps will not be processed.

Service Event Type: HTTP-XML Message Received

An HTTP-XML message received service event type includes XML message mappings that will map messages sent from the source back end system to the Agentry Server to indicate data has changed in that back end system. This call is made by the back end system via a CGI message containing XML data or an XML document. The service event will filter messages using an XPath value. The service event will then handle this message by processing the read, data state, and update step definitions to update the destination back end system.

This type of service event includes a child definition called message mapping. A given HTTP-XML Message Received service event may have one or more message mapping. Each mapping definition is intended to map data from the XML message or document to the properties of the target object of the service event, or to one of a selection of other data items within the application.

As a part of the HTTP-XML service event type, an HTTP response is defined. This response is sent by the Agentry Server back to the HTTP server that initially sent the XML message; that is, the response is sent back to the source back end system. Included in this response is one of the standard HTTP response status codes, as well as other possible information. This information can include fixed string data, or HTTP markup. This response after all processing for the service event has completed, including all read, data state, and update step execution.

HTTP-XML Message Received Service Event - General Attributes

- **Name:** The internal unique name for the service event definition, must be unique among all service events defined within the same module.
- **Connection:** This attribute references an HTTP-XML system connection type within the same application. This connection is the one over which the source back end system will send the message to the Agentry Server containing the XML data or document to be processed by the service event. This system connection must exist prior to defining the HTTP-XML Message Received service event and must be of type HTTP-XML.
- **Message Filter:** This attribute contains the XPath statement to filter messages received by the service event. If the service event can select one or more nodes within the message document using this XPath, the service event will process the message. If it cannot make such a selection, it will ignore the message.
- **Object Type:** This attribute specifies the target object type of the service event. The properties of this object will store the data retrieved from the source back end to be updated to the destination back end. This may include child object collections where necessary.
- **Read Steps:** This attribute specifies whether to retrieve data from the source back end system using the read steps defined in the service event, or those defined in the target object type of the service event. When this attribute is true, the object's read steps will be used and the service event's read steps will not be processed.
- **Run Push:** This attribute specifies a push definition to run when this service event type is called from the back end system. This is provided as an alternative to the default polling

behavior of a push definition, allowing them to be run on demand by the back end system instead.

HTTP-XML Message Received Service Event - HTTP Response Attributes

HTTP Response

- **HTTP Response Code:** This attribute specifies the HTTP response status code sent by the Agentry Server to the source back end system that initially called the service event.
- **Response Data Type:** This attribute specifies the data type of any data within the HTTP response status sent by the Agentry Server. This can be set to Fixed String, Small Markup or Large Markup. A fixed string is a single string value defined in the **Response** attribute. Small Markup is a short piece of HTML text, also set in the **Response** attribute. Large Markup is a larger chunk of HTML text, likely spanning multiple lines. This text is stored in the file referenced by the **Markup File** attribute.
- **Markup File:** This attribute is available only when the Response Data Type is set to Large Markup. Markup File references the text file containing the HTML text to be sent as a part of the HTTP Response by the service event.
- **Response:** This attribute is available when Response Data Type is set to either Fixed String or Small Markup. For fixed string, the Response attribute can contain any text value. For Small Markup the Response attribute should contain HTML text. The contents of the Response attribute are sent as a part of the HTTP Response by the service event.

Error Response

- **Error Response Code:** This attribute specifies the HTTP response status code sent by the Agentry Server to the source back end system that initially called the service event.
- **Error Data Type:** This attribute specifies the data type of any data within the HTTP response status sent by the Agentry Server. This can be set to Fixed String, Small Markup or Large Markup. A fixed string is a single string value defined in the **Response** attribute. Small Markup is a short piece of HTML text, also set in the **Response** attribute. Large Markup is a larger chunk of HTML text, likely spanning multiple lines. This text is stored in the file referenced by the **Markup File** attribute.
- **Markup File:** This attribute is available only when the **Response Data Type** is set to Large Markup. **Markup File** references the text file containing the HTML text to be sent as a part of the HTTP Response by the service event.
- **Response:** This attribute is available when **Response Data Type** is set to either Fixed String or Small Markup. For fixed string, the **Response** attribute can contain any text value. For Small Markup the **Response** attribute should contain HTML text. The contents of the **Response** attribute are included as a part of the HTTP Response sent by the service event.

HTTP-XML Service Event Message Mapping

The HTTP-XML service event type includes the child definition type Message Mapping. Service events of this type can contain one or more of these child definitions. The purpose of a

message mapping is to map data in the XML document that is a part of the message received by the service event to the properties or other data values within the application.

A part of the message mapping definition is the XPath to the location of the data in the XML structure. When a data value is found it is then mapped, according to the message mapping, to either the property values of the object type targeted by the service event, or to one of a list of other options for data sources within the application.

HTTP XML Service Event Message Mapping Attributes

- **Mapping Type:** This attribute specifies the mapping type. This may be either XPath Expression or XML Transformation.
- **Base XPath:** This attribute is optional and should be used when returning multiple instances of the same data element in the XML content. When a **Base XPath** is defined for a response mapping, the same value will be set by default in the add wizard for subsequent message mappings within the same parent service event.
- **XPath:** This attribute is only available when the **Mapping Type** is set to XPath Expression. This attribute contains the XPath value to extract the desired value from structured XML data contained in the message received by the service event.
- **XSL:** This attribute is only available when the **Mapping Type** is set to XML Transformation. It contains the XSL expression to transform the XML data contained in the message received by the service event.
- **Maps To:** This attribute specifies where the value extracted by the message mapping is stored in the application. This may be one of the following values for a service event:
 - *Last Update:* This selection specifies the extracted value is a date and time indicating when the object's source in the back end system was last modified. This value is mapped to the last update value within the object instance created by the service event.
 - *Local String (<<local>>):* This selection will create a local data tag available to subsequent message mappings in the same parent service event. The value of this data tag will be the value extracted by the response mapping. When selected, the attribute String Name will be available to name the local data tag created. This is the equivalent to calling the SDML function tag <<local . . .>>.
 - *Local XML (<<localXML>>):* This selection will create a local XML data tag available to subsequent message mappings in the same service event. The value of this data tag will be the value extracted by the response mapping.
 - *Parent Object Key Property:* This selection will set the value extracted by the message mapping to the key property of the parent object to the object created by the message received by the service event. This will not change the parent object's key property, but rather is used by the Agentry Server to identify which object is the parent object.
 - *Property Path:* This select will set the value extracted by the message mapping to the property selected in the Property Path field. This will change the value of the property to the value extracted from the message received by the service event.

Service Event Type: File System Monitor

A File System Monitor service event type is defined to monitor a specified directory on the Agentry Server's host file system for changes. When a file system monitor service event is defined and published to the Server, the Server will begin monitoring the directory the service event has been defined to watch, which is then the source back end system for the service event. When a change occurs to the contents of this directory, the Server will attempt to open each file in the directory for reading. Once a file is opened successfully, the service event's defined command will be executed, followed by its read, data state, and update step definitions to update the destination back end system.

If the Agentry Server is unable to open a file for reading, it will wait a short period and attempt the operation again. If it is unable to open the file after this second attempt, it will skip the file and process the next one found in the directory. When a file is successfully opened, the defined command for the service event is executed for that file. The service event's child step usage definitions are then processed. Once this is complete, the next file in the directory is processed according to this same procedure. Note that the Agentry Server will delete all files from the directory that it successfully processes. To prevent this behavior, the command should include copying the file to a different location, or rename the file to one that does not match the **File Filter** attribute of the service event definition.

The command defined for this service event type is primarily intended to prepare files for transfer or reading by the service event. The command can be any executable file type and is a Windows batch file (.bat) by default. The file type may be changed by editing the file extension of the file name in the **File** attribute of the Command tab.

A file system monitor service event includes the document mapping child definition type. The service event may contain one or more of these child definitions, each of which will map a file or file-related data to the properties of the object type targeted by the service event; or alternately one of the other data components of the application.

File System Monitor Service Event Attributes

- **Name:** The internal unique name for the service event definition, must be unique among all service events defined within the same module.
- **Connection:** This attribute references a File system connection type to the Agentry Server's host file system. The directory monitored by the service event must exist on this file system. The Agentry Server must have read-write access to this directory. The system connection must be of type File System and must exist prior to defining the service event.
- **Directory:** This attribute specifies the directory to be monitored by the service event. This path can be either a full path beginning with the file system root, or it may be a relative path to the installation location of the Agentry Server. The Server must have read-write privileges to this directory. Changes to this directory monitored by the service event include the addition of new files or modifications of existing files determined by changes to the modification date in the file's metadata.

- **File Filter:** This attribute can contain any file name matching characters to specify which files or file types the service event should monitor. This value can include wild cards in the form of asterisks. Any files at the location specified by the Directory attribute that match the file name pattern specified in File Filter will be monitored. Any others will be ignored by the service event. If File Filter is left blank, all files within the directory will be monitored.
- **Object Type:** This attribute specifies the target object type of the service event. The properties of this object will store the data retrieved from the source back end to be updated to the destination back end. This may include child object collections where necessary.
- **Read Steps:** This attribute specifies whether to retrieve data from the source back end system using the read steps defined in the service event, or those defined in the target object type of the service event. When this attribute is true, the object's read steps will be used and the service event's read steps will not be processed.

File System Monitor Service Event Document Mapping

The File System Monitor service event type includes the child definition type Document Mapping. A document mapping is defined to map a file or other data generated by the service event's command to a property within the application. Data that may be mapped includes a file created by the command, output written by the command to standard error or standard output, or the exit code passed to the operating system by the command.

Multiple document mappings may be defined for the same parent service event to capture each of these values. This can allow the application to determine if an error occurs when the service event's command is executed, as well as specific information about the error.

File System Monitor Service Event Document Mapping Attributes

- **Property:** This attribute specifies the property to which the data extracted by the document mapping is assigned. The data type of the property selected should reflect the setting for the Output Type attribute.
- **Output Type:** This attribute specifies which output from the service event's command contains the data to be mapped to the item referenced in the **Property** attribute. This may be set to one of the following options:
 - *Command Exit Code:* This selection specifies the value returned by the command to the operating system. This exit code will be the value stored in the item selected in the Property attribute. The property selected should be of type integral number to store the command exit code in most cases.
 - *File Created by Command:* This selection specifies that the file created by the service event's command should be assigned to the item referenced in the Property attribute. This data of the selected property should be External Data when this Output Type is defined.
 - *STDERR:* This selection specifies that any output from the service event's command written to STDERR, or standard error, is assigned to the item referenced in the Property

attribute. The data type of the selected property should be String when this Output Type is defined.

- **STDOUT:** This selection specifies that any output from the service event's command written to STDOUT, or standard output, is assigned to the item referenced in the Property attribute. The data type of the selected property should be String when this Output Type is defined.
- **File Name:** This attribute is available only when the Output Type is set to File Created by Command. The File Name attribute specifies the name of the file created by the service event's command that is to be referenced by the item selected in the Property attribute. This attribute can include the SDML <<script>>, which expands to the name of the file in which the service event's command is stored. It is common to use this value as a part of the name for the file generated by the command.
- **Delete File:** This attribute specifies whether or not to delete the file created by the service event's command. When set, the file will be removed after the service event has finished processing it. Otherwise the file will remain after the service event has completed processing.

Step

A step defines a single piece of processing to be performed by the Agentry Server with a specific back end system. There are different types of steps, defined based on the system connection for which the step is defined. A step defines what action to take and against which back end system. It will not be executed by the Agentry Server unless it is referenced by another definition that defines Server processing.

The step definition must be used by a step usage definition to give it context and purpose. The step itself defines the back end system and the task to perform. The context of the step will dictate what values the step will have access to within the application data.

Regardless of the type of step, all may update data to a back end system, including adding, editing, or deleting that data; and all may retrieve data from the back end system, returning it to the Agentry Server for use in the application.

There are five types of steps that may be defined, with one each for the Java Virtual Machine, SQL Database, and HTTP-XML system connection types. The File system connection supports two step types. When a step definition is created in the Agentry Editor the first information entered is the system connection for which the step is defined. Based on this selection the type of step can then be entered. Following are the types of step definitions that can be defined. The appropriate system connection must exist prior to defining the step.

- **SQL Query:** A SQL Query step is defined for a SQL Database system connection and contains the SQL logic to be processed by the Agentry Server for a database back end system.
- **Java Steplet:** A Java Step, or Steplet, is defined for a Java Virtual Machine system connection and contains the Java logic to be processed by the Agentry Server for a Java interface.

- **XML via HTTP:** An XML via HTTP step is defined for an HTTP-XML system connection and defines a URL called by the Agentry Server and also defines how the XML data returned from this call is mapped to the data members of the mobile application.
- **File Command Line:** A File Command Line step is defined for a File System Connection and defines a command to be executed by the Agentry Server on the host system of the Server.
- **File Document Management:** A File Document Management step is defined for a File System Connection and defines a command to be executed by the Agentry Server on the host system of the Server in support of transferring files between the Server and the Agentry Client.

Step Type: SQL Query

A SQL Query step is defined for a SQL Database system connection and contains the SQL logic to be processed by the Agentry Server for a database back end system. The logic for a SQL Query step is contained in a text file with a `.sql` extension. The contents of this file are processed by the Agentry Server prior to submission to the database for execution. This preprocessing includes expanding any SDML tags. The results from this expansion must be a valid SQL statement for the target database type.

The contents of the query file for this step type may be accessed directly on the Agentry Development Server for the application project. The file path listed for this file is relative to the installation location of this Server. If this file is modified it is not necessary to publish the application project for the change to be exhibited. In a production environment this file is not directly accessible in this manner and must be modified through the Agentry Editor. Changes made in a production environment must be published.

SQL Query Step Attributes

- **Name:** Contains the unique internal name of the step definition. This value must be unique among all step definitions within the same module.
- **Connection:** This attribute references the system connection for which the step is defined. This attribute is set when the step is initially created and cannot be modified. For a SQL Query step this must be a SQL Database system connection type.
- **File:** This attribute contains the path and file name of the `.sql` file containing the step's SQL statement. This path is relative to the path

```
ServerDirectory\Application\Development\Scripts
```

where `ServerDirectory` is the installation location of the Agentry Development Server.

Step Type: Java Steplet

A Java Step, or Steplet, is defined for a Java Virtual Machine system connection and contains the Java logic to be processed by the Agentry Server for a Java interface. The logic for a Java Steplet is contained in a Java source file with a `.java` extension. This file is added to an

existing Java project in the Eclipse Java perspective. This class created is an extension of the Agentry Java API class `Steplet`. The contents of this file are processed by the Java Virtual Machine running on the host system for the Agentry Server.

With the release of the Agentry Mobile Platform version 5.1 the process for creating of a Java step definition has changed. The new procedure reflects support for the Java perspective provided within the Eclipse and allows the developer to add Java logic for a step definition to existing projects within the Java perspective. The creation of the Java logic portion of a step definition is now performed through the Java perspective's wizard for creating classes, and allows for the selection of the package to which the step is to be added. Java steps created in previous versions of the mobile platform are still supported and will still reside on the Server's file system. New Java steps defined for the application should be created using the Java wizards provided by the Java Perspective. The file for these steps will then be saved to the file system according to the configuration of the Java project to which the `Steplet` is added.

Java Steplet Attributes

- **Name:** Contains the unique internal name of the step definition. This value must be unique among all step definitions within the same module.
- **Connection:** This attribute references the system connection for which the step is defined. This attribute is set when the step is initially created and cannot be modified. For a Java `Steplet` this must be a Java Virtual Machine system connection type.
- **Source Type:** This attribute specifies the source of the Java logic for the step and may be set to one of the following options:
 - *Existing Class:* This selection will allow for the selection of an existing class within a project in the Java perspective. The class selected will be the one called when the Java step is processed by the Server at run time. This class must be an extension of the Agentry Java API class `Steplet`.
 - *New Class:* This selection will create a new Java class that is an extension of the Agentry Java API class `Steplet`. The Java class wizard within the Java perspective in Eclipse will be displayed to allow for the creation of this class, including specifying the project and Java package to which it should be added.
 - *Source (deprecated):* This selection, as indicated, is deprecated and is provided to support the now deprecated method of managing Java classes for an Agentry mobile application. This selection will use and store a `.java` file on the Agentry Server's file system. Note that this selection will prevent the ability to organize the source file for the step in a Java project within the Eclipse Java perspective.
- **File:** Note that this attribute is deprecated as of version 5.1 of the Agentry Mobile Platform. While still supported for existing Java steps, it should not be used in new step definitions. It is only valid when the Source Type attribute is set to the option "Source." This attribute contains the path and file name of the `.java` file containing the step's SQL statement. This path is relative to the path

```
ServerDirectory\Application\Development\Scripts
```

where `ServerDirectory` is the installation location of the Agentry Development Server.

Step Type: XML via HTTP

An XML via HTTP step is defined for an HTTP-XML system connection and defines a URL called by the Agentry Server and also defines how the XML data returned from this call is mapped to the data members of the mobile application. This step includes two child definitions that encapsulate the arguments passed by the Server to the defined URL, and the mapping of return values to the data members of the application. An XML via HTTP step is defined for a specific definition type within the same module.

The first information entered for an XML via HTTP step is the definition for which it is defined, which may be an object, transaction, or fetch. This information is needed by the step definition for use in its child definitions, which must have access to the property values of the selected data definition as a part of their behaviors.

Within this step type is the HTTP request. This portion of the step defines the URL called by the Agentry Server and the HTTP request method. This may be one of GET, HEAD, POST, or PUT.

The child definitions to an XML via HTTP step include its request arguments and response mappings. Request arguments provide access to the property values and other data values in scope for the step to be passed to the URL defined by the step. Included in this definition is the type of argument the data represents.

Response mappings extract data from the structured XML data or document returned from the request. They may use of XPath's to locate and retrieve these values from the XML and define to which property or other data member of the application the XML contents will be mapped. Response mappings may be used to extract a specific XML element's contents, or a parent element may be specified with a second, child element within that parent of which there may be multiple instances.

XML via HTTP Child Definitions

- **Request Argument:**
- **Response Mapping:**

XML via HTTP Step Attributes

General Attributes

- **Used For:** This attribute specifies the data definition within the module for which the step will synchronize data. This may be any fetch, transaction, or object definition within the application. This attribute set by first selecting the type of definition and then selecting the specific definition within the project.
- **Name:** Contains the unique internal name for the step definition. This must be unique among all steps within the same module.

- **Connection:** This attribute references the system connection for which the step is defined. This attribute is set when the step is initially created and cannot be modified. For an XML via HTTP step this must be an HTTP-XML system connection type.

HTTP Request Attributes - These attributes are accessible after the step has been defined. They are organized as a child definition to the step itself and can be navigated to in the Editor within the Application Explorer view. The HTTP Request for an XML via HTTP step is one of the rare instances within the Agentry project structure where there may be only one instance of a child definition within a given parent.

- **Name:** Contains the unique internal name for the HTTP request within the step definition. This is set by default to the name of the parent step definition. It may be modified. A given XML via HTTP step will have only one HTTP request.
- **URL:** This attribute contains the URL to which the Agentry Server will make a request. This value will be appended to the value configured as the base URL for the HTTP-XML system connection. This base URL is configured within the HTTP-XML system connection configuration options for the Agentry Server. Proper use of both this base URL option and the URL entered in the requests of the step definitions can support portability for the application, with the base URL being the implementation-specific portion and the step's URL being the portion not likely to change for the same back end system from one implementation to the next.
- **Method:** This attribute specifies the HTTP request method for the request. This may be set to one of GET, HEAD, POST, or PUT.

XML via HTTP Step Request Argument

An HTTP Request Argument is a child definition to the XML via HTTP step definition, defining the data values passed as arguments to the parent step's defined URL. Included in the request argument definition is the type of request and the property value or other data value in scope for the step to be passed as the argument. Request arguments also have a data type, which specifies the source for the argument's data.

A request argument is defined for the parent step only when it is necessary to pass arguments to the step's defined URL request. The request allows for access to the property values of the definition for which the step was defined, as well as values at the user or application level via the SDML. A fixed string value may also be defined to be passed as the argument.

The values accessed via the SDML can be contained in either a small or large markup value. Both allow for the use of HTML markup text. The difference between these two items is the manner in which the markup is stored. For a small markup argument, a single field that can contain one line of markup text is available within the request argument.

Each of these data sources is a different data type within the request argument. A data type is selected first within the definition, followed by the specific value of that type.

The argument itself also has a type. This may be one of CGI Argument, Cookie, HTTP Header, or XML Body. The selection of the argument type specifies how the data for the request argument is passed to the URL defined in the parent step definition.

A given XML via HTTP step can contain multiple arguments. All arguments are listed within the Properties View of the parent step definition. Within this list, the position of each request argument definition specifies the order in which the arguments will be passed to the URL request. This order can be changed by moving the arguments up or down within the list.

Request Argument Attributes

- **Argument Type:** This attribute specifies how the data within the argument will be passed to the URL defined in the parent step definition. This may be one of CGI Argument, Cookie, HTTP Header, or XML Body.
- **Name:** Contains the unique internal name of the request argument. This value must be unique among all request arguments within the same step definition. The field label in the Editor for this attribute will change based on the selection of the Argument Type attribute.
- **Data Type:** This attribute specifies the type of the data for the argument. This selection determines the source for the argument data within the mobile application.
 - *Fixed String:* This selection specifies the argument will be a plain text value. When this is selected the String attribute will be enabled, allowing for the entry of text value to be passed for the argument.
 - *Large Markup:* This selection specifies the argument will be HTML markup. The markup text will be stored in a text file. This file is accessible on the Agentry Development Server for the application project and may be edited directly from this location, or from within the Agentry Editor. The relative path and name for this file is listed in the Markup File attribute, which is enabled when the “Large Markup” type is selected in the **Data Type** attribute.
 - *Property Path:* This selection specifies that the argument value is contained in a property of the definition for which the parent step has been defined. When this selection is made the Property Path attribute is enabled, where the property can be selected.
 - *Small Markup:* This selection specifies the argument will be HTML markup. The markup text will be entered in the Markup Text attribute field, which will be enabled for this selection. This argument Data Type allows for a single line of HTML markup to be entered for the argument.
 - *User ID:* This selection specifies that user’s login ID for the Agentry Client is passed as the argument value. No other attributes are enabled in relation to this selection.
- **String:** This attribute is enabled when the Data Type attribute is set to Fixed String. The String attribute contains the plain text value passed as the argument value to the parent step’s URL request.
- **Mask in Log:** This attribute can be set to hide the value of the argument in logs generated by the Agentry Server. In place of the value, a series of asterisks is recorded. Typically this is used for passwords and other sensitive values.
- **Markup File:** This attribute is enabled when the Data Type attribute is set to Large Markup. The Markup File attribute lists the relative path and file name for the text file containing the HTML markup text. This path is relative to the Agentry Development Server’s installation location. The default value is the location:

```
ServerDirectory\Application\Development\Scripts
```

where `ServerDirectory` is the installation location for the development server. This path may be changed, relative to this location, though this is rarely necessary for this definition type.

- **Property Path:** This attribute is enabled when the Data Type attribute is set to Property Path. The Property Path attribute references the property definition within the definition for which the parent step was defined. The value of this property will be the value passed as the argument to the parent step's URL request.
- **Markup Text:** This attribute is enabled when the Data Type attribute is set to Small Markup. The Markup Text attribute can contain a single line of HTML markup text that will be passed as the argument to the parent step's URL request.

XML via HTTP Step Response Mapping

An XML Response mapping is a child definition to the XML via HTTP step definition, defined to extract data returned by the parent step and map it to the property values or other data members of the application. Data returned may be extracted by the mapping when that takes the form of a Cookie, HTTP Header, or XML. The value extracted may be assigned to the properties of the definition for which the step was defined, or to one of several other data values within the application.

The response mapping defines both the source type, or "Mapping Type," of the data returned by the step definition, and the data component of the mobile application where the value is stored. If the mapping type is an XPath Expression or XSL Transformation, the return data must be structured XML. Included in the response mapping then is the XPath or XSL to extract the data from the XML document received by the step after its request was submitted.

Once the data is extracted by the response mapping definition it is assigned to the data component of the mobile application as specified within the mapping definition. This can include a property within the definition for which the parent step was defined, as well as messaging, user ID, the creation of local and local XML data tags, the parent object's key property, or the value may be used for validation.

A given XML via HTTP step can contain multiple response mappings. Each will extract data from the same data set returned by the step to the Agentry Server. The parent step definition's property view contains a list of all response mappings. The order in which the mappings are processed is the position in which the mappings are listed in this view. This order can be modified by moving the mappings up or down within the list.

Response Mapping Attributes

- **Mapping Type:** This attribute specifies the type of data from which the value will be extracted for the response mapping. This may be one of Cookie, HTTP Header, XPath Expression, or XSL transformation. If XPath or XSL is selected, the return set from the step is assumed to be an XML document. This requires the definition of the Base XPath

and XPath, or XSL attributes to specify which components of the XML document are to be extracted.

- **Name:** Contains the unique internal name for the response mapping. This value must be unique among all response mappings within the same parent step. The label for this attribute field in the Editor will change based on the selected Mapping Type.
- **Base XPath:** This attribute is enabled when the Mapping Type attribute is set to either XPath Expression or XSL Transformation. The Base XPath is set to locate an element within the XML Document that contains one or more child elements of the same type. The XPath attribute then specifies the specific child element type within the element specified by the Base XPath. The response mapping will iterate over all instances of the child element, extracting the value of each and assigning to the value specified in the mapping attributes. This is most commonly used when synchronizing data object instances within a collection property.
- **XPath:** This attribute contains the XPath expression for the specific element within the XML Document whose contents are to be extracted by the response mapping. This expression is used in combination with the Base XPath (if specified) to provide iterative processing of multiple instances of the same element within the same parent element within the document.
- **XSL:**
- **Maps To:**
 - *Error Message:* This selection will map the data to error text display by the mobile application.
 - *Last Update:* This selection specifies the extracted value is a date and time indicating when the data table's source in the back end system was last modified. This value is compared against the internal last update value for the data table as provided by the Client.
 - *Local String* (<<local>>): This selection will create a local data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping. When selected, the attribute String Name will be available to name the local data tag created. This is the equivalent to calling the SDML function tag <<local . . .>>.
 - *Local XML* (<<localXML>>): This selection will create a local XML data tag available to subsequent data mappings in the same parent HTTP Request. The value of this data tag will be the value extracted by the response mapping.
 - *Notification [Cancel] Button Label:* This selection specifies that the value extracted by the response mapping should be used to label the cancel button displayed in the Notification Dialog displayed by push definitions. If the step is used in any other manner than push processing, this selection will have no effect.
 - *Notification [OK] Button Label:* This selection specifies that the value extracted by the response mapping should be used to label the OK button displayed in the Notification Dialog displayed by push definitions. If the step is used in any other manner than for push processing, this selection will have no effect.

- *Notification Text:* This selection specifies that the value extracted by the response mapping should be used as the message text displayed in the Notification Dialog displayed by push definitions. If the step is used in any other manner than for push processing, this selection will have no effect.
- *Notification Title:* This selection specifies that the value extracted by the response mapping should be used as the title bar text of the Notification Dialog displayed by push definitions. If the step is used in any other manner than for push processing, this selecting will have no effect.
- *Parent Object Key Property:* This selection specifies that value extracted by the response mapping should be matched with the key property of the parent object to the object being synchronized. This is primarily used in fetches and object read steps, where the data for a nested collection is being retrieved. When this option is selected, the **Key Property** attribute will be enabled to allow for the selection of the parent key property to match with the value.
- *Property Path:* This selection specifies that value extracted by the mapping is assigned to a property within the object definition for which the step has been defined. This selection has no meaning for steps defined for fetch definitions. For steps defined for transactions this option is selected when the parent step is expected to return the key property of the object targeted by the transaction. This is only the case when the transaction step usage definition defines a client response of Update Client Key Property.
- *User ID:* This selection will map the value extracted by the response mapping to the user's ID. This value is the equivalent to the SDML data tag `<<user.id>>`. If a previous response mapping in any HTTP Request processed by the Server set the user ID, setting it here will override that value. This will then be the value available for all HTTP-XML system connection processing where the data tag `<<user.id>>` is referenced.
- *Validation:* This selection allows for validation within of the user during the request made by the step. When the validation item is selected, the XPath defined for the step must successfully locate and XML element. The failure to locate the element is treated as failed validation.
- **String Name:** This attribute is enabled when the **Maps To** attribute is set to "Local String (`<<local>>`).". This attribute contains the name of the local data tag to be created. This name can be set to any character consisting of alphanumeric characters.
- **XML Name:** This attribute is enabled when the **Maps To** attribute is set to "Local XML (`<<localXML>>`).". This attribute contains the name of the local XML data tag to be created. This name can set to any string consisting of alphanumeric characters.
- **Key Property:** This attribute is enabled when the **Maps To** attribute is set to "Parent Object Key Property." The **Key Property** attribute is set to the key property of an object that is an ancestor to the object being synchronized.
- **Property:** This attribute is enabled when the **Maps To** attribute is set to "Property Path." The **Property** attribute is set to the property whose value will be set to the one extracted by

the parent step definition. The property selected here should be defined within the definition for which the parent step was defined

Step Type: File Command Line Step

A File Command Line step is defined for a File System Connection and defines a command to be executed by the Agentry Server on the host system of the Server. This command is contained in a script file, with a default file extension of `.bat`, which is the Windows batch file extension. This extension may be changed to match the script language used in the file. The command executed by the Agentry Server can be monitored for its return value. The script file is processed by the Server to expand any SDML tags it may contain prior to execution against the host system.

The contents of the script file for this step type may be accessed directly on the Agentry Development Server for the application project. The file path listed for this file is relative to the installation location of this Server. If this file is modified it is not necessary to publish the application project for the change to be exhibited. In a production environment this file is not directly accessible in this manner and must be modified through the Agentry Editor. Changes made in a production environment must be published.

As an alternative to storing the command in an external file, it may be contained in the Command attribute of the step. Such a command must consist of a single line. By default the Command attribute is set to the SDML tag `<<script>>`, which expands at run time to the file referenced in the steps File attribute.

A file command step can be defined to wait for the command it calls to complete execution. When defined in this manner, the back end synchronization for a user will not continue until the command returns, or until the defined wait period expires. If the wait period is exceeded, the Agentry Server will log an error and the synchronization will be halted.

If the step is not defined to wait for the command to complete, an error will only be logged if the defined command cannot be executed by the Server for any reason.

The script file or the text in the Command attribute for this step is processed by the Agentry Server, which runs it through the Server's SDML pre-processor before executing the step. The results of this SDML expansion are written to a temporary directory, based on the Server's configuration.

File Command Line Step Attributes

- **Name:** Contains the unique internal name of the step definition. This value must be unique among all step definitions within the same module.
- **Connection:** This attribute references the system connection for which the step is defined. This attribute is set when the step is initially created and cannot be modified. For a File Command Line step this must be a File system connection type.
- **File:** This attribute contains the path and file name of the script file containing the step's commands. This path is relative to the path

```
ServerDirectory\Application\Development\Scripts
```

where `ServerDirectory` is the installation location of the Agentry Development Server.

- **Command:** This attribute is set the SDML data tag `<<script>>` by default. This tag expands to the script referenced in the File attribute. If the command does not contain this data tag, its contents are assumed to be the command to be executed by the Server. In this case the command must be a single line, which may contain SDML tags.

Wait Attributes

- **Wait:** This attribute specifies whether or not the Agentry Server should wait for the command executed by the step to complete before processing the next step in the synchronization process. When set to true, the Server will wait for the duration of time specified in the **Wait Period Limit** attribute. If the command does not complete within this limit, the Server will attempt to kill the command process. It will then log an error message and halt synchronization.
- **Wait Period Limit:** This attribute specifies the duration of time the Agentry Server is to wait for the command executed by the step to complete. This attribute is available only when the **Wait** attribute is set to true.
- **Delete Script File:** This attribute specifies whether or not the script file created by the Agentry Server as result of processing the script file for SDML expansion should be deleted or kept. This attribute is available on when the **Wait** attribute is set to true.

Step Type: File Document Management Step

A File Document Management step is defined for a File System Connection and defines a command to be executed by the Agentry Server on the host system of the Server in support of transferring files between the Server and the Agentry Client. The command for this step is stored in a text file executed as a script by the Server. This step type also includes a child definition to encapsulate mappings between the file data and the data members of the mobile application. A File Document Management step is defined for a specific definition type within the same module.

A file document management step can define a command to be executed to retrieve a file from a file system or version control system so that it may be transferred to the Agentry Client. The child definition document mapping can then associate this file with an object property, normally of type External Data. It may also define a command that moves a file referenced by an External Data property within a transaction to a permanent location on the file system or version control system. The file is also associated with the property via the document mapping child definition.

The definition for which the step is defined may be an object, transaction, or fetch. The property referencing the file to be transferred should be a child property of the selected definition.

A component of this definition is the Document Management Script. This script contains the command or commands the Agentry Server will execute in support of the file transfer

behavior. This script is by default a Windows batch script (.bat). The file extension for the script may be changed to reflect the type of script language it contains.

The document management step can be defined to wait for the command it executes to return, or it can execute the command without waiting. If defined to wait for the command, the next step to be processed in the synchronization will not be run until the command has completed execution, or until a defined wait period has been exceeded. If the wait period is exceeded, the Agentry Server will log an error and synchronization will stop.

For downstream synchronization, i.e. fetch, push, or object read step processing, the command is expected to produce a file to be transferred to the Agentry Client. For upstream synchronization, i.e. transaction processing, the command is expected to process the file after it has been transferred from the Agentry Client to the Server's host system. This may include moving it to another location on the file system, or checking it in or updating it to a version control system, or any other post-transfer processing that should occur for the file.

In addition to the file itself, it is also possible to capture values from the document management command run by the step. This behavior is also defined in the child definition document mapping. Return values, error codes, and similar data can be assigned to properties of the appropriate data type.

The contents of the script file for this step type may be accessed directly on the Agentry Development Server for the application project. The file path listed for this file is relative to the installation location of this Server. If this file is modified it is not necessary to publish the application project for the change to be exhibited. In a production environment this file is not directly accessible in this manner and must be modified through the Agentry Editor. Changes made in a production environment must be published.

File Document Management Step Child Definitions

Document Mapping: A document mapping definition is a child to a file document management step and defines the correlation between the file produced by that step to a property definition, normally of type External Data.

File Document Management Step Attributes

- **Name:** Contains the unique internal name of the step definition. This value must be unique among all step definitions within the same module.
- **Connection:** This attribute references the system connection for which the step is defined. This attribute is set when the step is initially created and cannot be modified. For a File Document Management step this must be a File system connection type.

Wait Attributes

- **Wait:** This attribute specifies whether or not the Agentry Server should wait for the command executed by the step to complete before processing the next step in the synchronization process. When set to true, the Server will wait for the duration of time specified in the **Wait Period Limit** attribute. If the command does not complete within this

limit, the Server will attempt to kill the command process. It will then log an error message and halt synchronization.

- **Wait Period Limit:** This attribute specifies the duration of time the Agentry Server is to wait for the command executed by the step to complete. This attribute is available only when the **Wait** attribute is set to true.
- **Delete Script File:** This attribute specifies whether or not the script file created by the Agentry Server as a result of processing the script file for SDML expansion should be deleted or kept. This attribute is available only when the **Wait** attribute is set to true.

Document Management Script Attributes

- **File:** This attribute contains the path and file name of the script file containing the step's commands. This is relative to the path

```
ServerDirectory\Application\Development\Scripts
```

where `ServerDirectory` is the installation location of the Agentry Development Server.

- **Command:** This attribute is set to the SDML data tag `<<script>>` by default. This tag expands to the script referenced in the File attribute. If the command does not contain this data tag, its contents are assumed to be the command to be executed by the Server. In this case the command must be a single line, which may contain SDML tags.

Document Mapping

A document mapping definition is a child to a file document management step and defines the correlation between the file produced by that step to a property definition, normally of type External Data. The specific behavior of a document mapping differs depending on the type of definition for which the parent step was defined. For objects and fetches, the document mapping defines where and how to access the file produced by the parent step's command. For a transaction, the document mapping defines how and where the file should be provided to the step's command.

Because of the differences between a document mapping for an object and fetch, and one for a transaction, there are different attributes for this definition type depending on how the parent step has been defined.

When a document mapping is defined within a file document management step for an object or fetch, the purpose of the document mapping is to capture output from the parent step definitions document script and map it to a property within the object or fetch. This output is primarily intended to be a file that is mapped to an external data property. This file will be transferred down to the Agentry Client. Other outputs may be captured from the document script, including output written to standard out and standard error, as well as the command's exit code as returned to the operating system.

When a document mapping is defined within a file document management step for a transaction, the purpose of the document mapping is to provide the contents of a property to the parent step definition's document script. This input to the command may be provided from

an external data property and passed to the command by either writing the file to the file system, or by piping it to the commands standard input. When piped to standard input, the option exists to pass the EOF character to that command after all file data has been passed in. When writing the file to the file system, the command is then expected to look for the file at that location and process it accordingly. When the command has completed processing the file, the option exists to delete the file from the file system. Note that this option will not be available if the parent document management step has been defined to not wait for the document script to complete execution.

Document Mapping Attributes - Object and Fetch

- **Property:** This attribute specifies the property to which the output from the parent step's command will be mapped. For a file produced by the command this should be a property of type External Data. For other output types, the proper data type of the property will vary.
- **Output Type:** This attribute specifies which output from the command to map to the selected property. The options to this attribute are:
 - *Command Exit Code:* This selection specifies that the exit code returned by the command to the operating system should be captured and mapped to the selected property.
 - *File Created By Command:* This selection specifies that a file created by the command should be mapped to the selected property. For this output type the **Property** attribute should be set to a property of type External Data.
 - *STDERR:* This selection specifies that any output written by the command to standard error should be mapped to the selected property. This may be done to determine if an error has occurred, and the nature of that error.
 - *STDOUT:* This selection specifies that any output written by the command to standard out should be mapped to the selected property.
- **File Name:** This attribute is enabled when the selected **Output Type** is "File Created By Command." The **File Name** attribute specifies the name of the file to be mapped to the selected property. This value may include SDML tags, with the default being `<<script>>-1.tmp`.
- **Delete File:** This attribute is enabled when the selected **Output Type** is "File Created By Command." The **Delete File** attribute specifies whether to keep the file created by the command after it has been transferred, or if it should be deleted.

Document Mapping Attributes - Transaction

- **Property:** This attribute specifies the transaction property containing the value to be passed to the document command. If the **Input Type** is "File Input to Command Line," this should be an External Data property.
- **Input Type:** This attribute specifies how the value or file referenced by the selected property will be passed to the document command of the parent step. This can be set to one of the following options:
 - *File Input to Command Line:* This selection specifies that file referenced by the selected property should be written to the file system and that the command will the

read it in from that location. The **File Name** attribute is enabled when this option is selected, and specifies the file name to which the file will be saved.

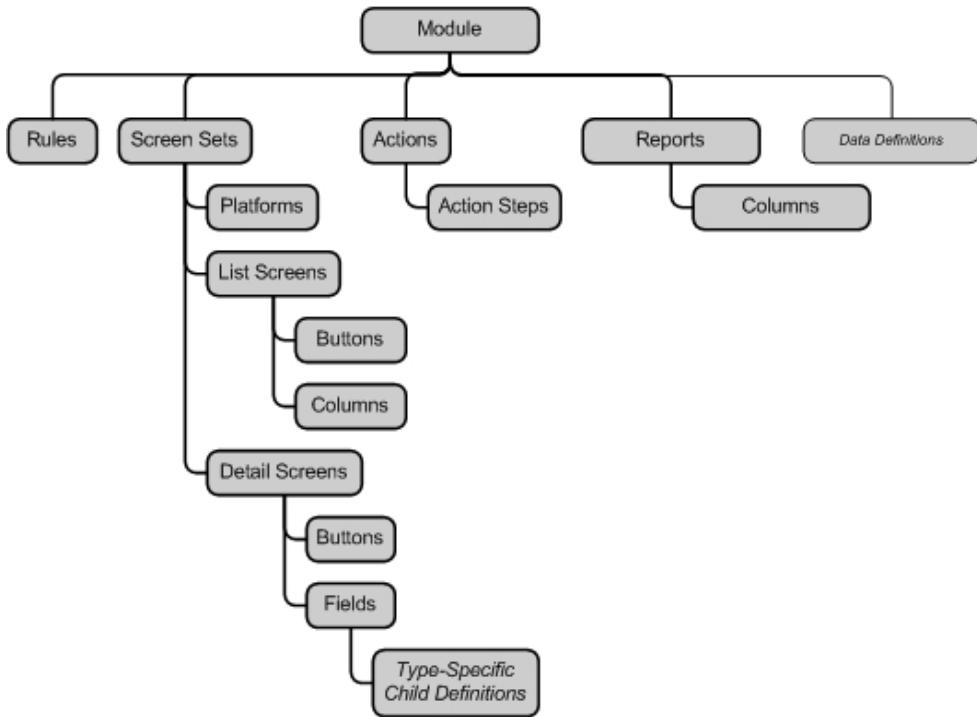
- **STDIN:** This selection specifies that the value of the selected property should be piped to the document command through standard input. For external data properties the file data will be streamed directly to the command without being written to the file system. When this selection is made the **Send EOF** attribute is enabled, indicating whether the EOF character should be sent to the command after the property data as been piped to the command.
- **File Name:** This attribute is enabled when the **Input Type** attribute is set to “File Input to Command Line.” This attribute contains the name to be given to the file when it is saved to the file system. This value may include SDML tags. It is set to <<script>>-1 . tmp by default.
- **Delete:** This attribute is enabled when the **Input Type** attribute is set to “File Input to Command Line.” The **Delete** attributes specifies whether the file saved to the file system by the Server should be deleted after the document command has finished processing it.
- **Send EOF:** This attribute is enabled when the **Input Type** attribute is set to “STDIN.” This attribute specifies whether or not to send an End of File character to the document command at the end of the file data. This is provided in support of those processes that require this character to indicate no further input is being sent.

Module-Level User Interface Definitions Overview

Within the module level of the application project in Agentry there are definitions for both data and user interface encapsulation. The user interface definitions encapsulate the screens and behaviors that expose the functionality within the application to the mobile users on the Agentry Client. These definitions do not have any direct impact on the behavior or functionality of the application as defined for the Agentry Server.

Of the user interface definitions, those that encompass the screens displayed on the Client are the most robust. The structure of these definition types is deeper than any of the other module-level definitions within the application.

Following is a graphic illustrating the module-level user interface definitions and their child and descendent definition types. This includes definitions that encapsulate the screens and screen controls displayed on the Agentry Client, the behaviors and functionality, and other similar user interface-related application components. Excluded from this graphic are the data definitions within the module. Note that this separation is for discussion purposes only. Within the application project structure, all child definitions to the module exist at the module level with no distinction made between them in the Agentry Editor in relation to whether they are data or user interface definitions.



The rule definition type within the module is actually one that crosses the line between a user interface and data definition. It is organized here with the user interface definition types, as a large portion of the rules written for a module affect this aspect of the behavior. However, rules can also be written and used within data definition types. The rule definition is described in this reference manual in its own section.

As indicated in this graphic, the screen set definition type is a deep structure, with several levels of child definitions below it. Note that, while separated in the above graphic, the list screen and detail screen items are both the same definition type, screen. A screen is a child definition to the screen set and, when defined, is either a list or detail screen. Each screen type has distinct child definitions, and thus are separated in the graphic shown here.

The field child definition to the detail screen can have child definitions of its own. This is dependent on the type of field defined, or the field's "edit type." The edit type of a field impacts the fields appearance and behavior on the Agency Client. Certain field edit types include child definitions that support their intended behaviors. Field edit types are discussed individually within this section of the manual and those that include child definitions are noted.

Overall the user interface definitions within the module display, expose, and provide the means to capture data to and from the mobile users. User interface definitions can display not only data from the module, but also data stored in the application level definitions data table and complex table.

User Interface Definition Types

The definition types within Agentry that define the Agentry Client's user interface are the screen set, platform, and screen.

- **Screen Set:** The screen set is the main Client user interface definition and defines what definition type its child screens display.
- **Platform:** The platform definition defines how the screens it uses within the same screen set appear on a specific device type.
- **Screen:** A screen definition defines how the property values in the definition being displayed are presented to the user on the Agentry Client. There are two possible screen types that may be defined, list screens and detail screens. Screen definitions have additional child definitions for the controls they display. These child definitions are dependent on the type of screen (list or detail) and the definition type displayed by the parent screen set.

Each of these definition types provide a separate portion of the UI functionality to the application and are broken out into these separate, but related definitions primarily to provide the separation of data and interface. This separation allows for the multi-device support by a single Agentry application. The overall structure of the definition hierarchy within Agentry, and the UI definitions' place within it, allows the business logic of an application to be separate from the UI. This also allows the UI to be defined to take full advantage of the capabilities of each device type.

Screen Set

A screen set definition defines the Agentry Client's user interface. The screen set defines the definition type to be displayed, which can be an object, transaction, or fetch within the same module. The properties of this definition type can then be displayed by the screen definitions within the screen set. Screen sets contain the child definitions screen and platform.

The type of data definition a screen set is defined to display will have an effect on the types of screens it may contain and how those screens are presented on the Agentry Client. When a screen set is defined to display an object it may contain both detail and list screens. Each screen within the screen set is displayed within the same window, with the screens represented by tab controls. In most cases the fields displayed on these screens are read-only.

When the screen set is defined to display a transaction or fetch it can only contain detail screens. These screens are displayed in a wizard format, with each screen displayed one at a time and containing navigation buttons to advance, reverse, cancel, or complete the wizard. Note that this navigation will also be affected by the action that displays the screen set. The fields of these screens can be read-only or editable based on each field's definition.

When a new module is added to an application project a single screen set will be defined within it automatically. This will be the main screen set for the module, making it the first screen set displayed on the Agentry Client when that module is viewed by the user. There is only one main screen set per module. This screen set definition can be altered but cannot be deleted.

Screen Set Child Definitions

The following definitions are child definitions to the screen set:

- **Platform:** The platform definition defines how the screens it uses within the same screen set appear on a specific device type.
- **Screen:** A screen definition defines how the property values in the definition being displayed are presented to the user on the Agentry Client.

Screen Set Attributes

- **Displays:** This is a two part attribute consisting of the definition type and the specific definition of that type the screen set will display. Screen sets can be defined to display objects, transactions, or fetches. The selection made here makes the data (properties) within that definition available to the screens defined within the screen set.
- **Name:** This is the unique internal name of the screen set that identifies the definition within the module. This value must be unique among all other screen sets in the same module and can contain no white space.
- **Main Screen Set:** This attribute cannot be set by the developer and is displayed in the add screen set wizard and properties screen in the Agentry Editor for reference purposes only. The main screen set for a module is created automatically by the Editor whenever a new module is defined.

Platform

A platform definition defines how a screen set's screens will appear on a specific device type. A platform is defined to use one or more screens within the same parent screen set. There are different platform types, each corresponding to a different type of client device. The platform affects the placement of buttons and the form factor of the screens it uses.

The most important attribute to the platform definition is the Platform Type. This attribute specifies the platform upon which the screens it uses will be displayed and how those screens will appear. A given screen set can contain one or more platform definitions. At least one platform must be defined before screens can be added to the screen set. During publish, at least one screen must be used by at least one platform within the screen set or an error will be returned and the publish will not be allowed to proceed.

A platform can use more than one screen within the same screen set. A screen can be used by more than one platform as well. At run time, when a screen set definition is sent to a client, the client's device type will determine which screens that client receives based on the platform using the screens.

Platform Attributes

General Attributes

- **Platform Type:** This is the type of device platform to be supported by the screens used by the platform and will affect the form factor and behavior of those screens.
- **Caption:** This is the title text displayed in the window on the Client at run time for the screen set. Since this is at the platform level, the screen set's window can contain a different caption on different target devices. This value may be set statically or via a rule definition for more dynamic text. A rule used here is expected to return a string value and is evaluated in the context of the object displayed by the screen set.
- **Size:** This attribute only applies to platform definitions for the Windows desktop, laptop, and tablet operating systems. For this type of platform the Size attribute specifies the initial display size of the screens it uses. For other **Platform Types** this attribute is disabled and all screens used by the platform are displayed in the full screen size of the device type. Note that this attribute may be affected or negated by the application definition's **Screen Size** attribute.
- **Button Placement:** This attribute contains four possible settings: Bottom, Top, Left, and Right. This attribute specifies where the buttons for all screens used by the platform are displayed.
- **List Navigation:** This attribute controls whether or not the object displayed by the screens used by the platform definition can be changed via navigation buttons drawn automatically on the Client. When true, these buttons will allow a user to change the object displayed in the current screen set based on a list of objects in the previous screen in the navigational flow. In this previous list, the previous or next item in the list is selected and the action executed to display the current screen set is executed again. This attribute has no effect on platforms for the module main screen set or for platforms within screen sets displaying a transaction or fetch. This behavior is applicable when the previous screen was a list screen, or when it was a detail screen containing a list view or list tile view field.
- **Screen Navigation:** By default a screen sets screens are displayed as tabs on the Client at run time. Selecting this option removes the tabs and instead displays a menu button containing the caption value of each screen definition to allow the user to select different screens.

Platform Screen Type

The following attributes are only valid for platforms of type iPad or Android and support the display of a pop up screen using the platform's screens.

- **Screen Types:** The options for this attribute include **Full Screen** and **Overlay View**. Full Screen creates standard screens on the Client. Overlay View creates popup screens that overlay the screen from which the user navigated. These screens can be used for both read only information as well as for data capture. When Overlay View is selected, the Height and Width attributes are enabled to specify the size of the overlay screen displayed.
- **Height:** The vertical size of the overlay screen in pixels.
- **Width:** The horizontal size of the overlay screen in pixels.

Platform Style Attributes

The style attributes of a platform specify the styles applied to different aspects of the screens used by the platform. Style definitions must exist before these attributes can be set. The final

appearance of the screen will be affected by the overall application of styles according to the style hierarchy. There are three groups of style elements for the platform: Screen Styles, Detail Screen Styles, and List Screen Styles. Screen styles affect all screens used by the platform regardless of screen type. Detail screen and list screen styles affect only those screens of the corresponding type.

All style attributes for the platform definition may be set statically by selecting the style from a list, or by returning the name of a style to apply from a rule definition. Rules evaluated for style attributes are expected to return a string value containing the name of the style to apply and are evaluated in the context of the object displayed by the parent screen set.

Screen Styles

- **Tabs:** The style to apply to the tab controls representing each screen within an object screen set. Has no affect on screens within a transaction or fetch screen set.
- **Buttons:** The style to apply to all button definitions for screens used by the platform.
- **Focused Buttons:** The style to apply to the button that currently has the focus.

Detail Screen Styles

- **Screen:** The style to apply to the screen as a whole. This will affect all portions of the screen not displaying a field or button.
- **Fields:** The style to apply to all fields displayed on the screen.
- **Focused Fields:** The style to apply to the field that currently has the focus.
- **Hyperlinks:** The style to field labels defined to be hyperlinks.

List Screen Styles

- **Screen:** The style to the list screen as a whole. This will affect all portions of the screen not displaying a list, header label, detail pane, or button.
- **Header Label:** The style to apply to the list screen's header label. If no header label is defined this attribute has no affect on the screen.
- **Column Labels:** The style to apply to the column labels on the screen's list control.
- **Rows:** The style to apply to all rows on the list screen. The Hyperlinks optional style will override the Rows style for cells with hyperlinks.
- **Alternate Rows:** The style to apply to every other row in the list, beginning with the second row. The Hyperlinks optional style will override the Alternate Rows style for every other row, specifically cells containing hyperlinks within the row.
- **Highlight Rows:** The style to apply to a row for the purposes of drawing attention to that row. This style should always be returned via a rule definition that evaluates the object being listed. The optional Hyperlinks style will be applied to the highlighted row's cells containing a hyperlink.
- **Selected Rows:** The style to apply to the row currently selected by the user in the list control. The optional Hyperlink style will be applied to any cells within the selected row containing a hyperlink.

- **Detail Pane:** The style to apply to both the foreground (text) and background of the list screen's detail pane. If no detail pane is defined this attribute has no affect on the screen.

Platform Button Attributes

Platforms defined within a screen set displaying a transaction or fetch include an additional set of attributes related to the buttons displayed on screens used by the platform. Screens within this type of screen do not contain button definitions, but rather contain buttons added to each screen automatically by the Client based on the action that displayed the parent screen set and the position of each screen within the screen set.

- **Cancel Action Details:** The label for cancel buttons that will cancel the action currently being executed.
- **Previous Screen Details:** The label for buttons that allow users to navigate to the previous screen in the current screen set.
- **Previous Record Details:** The label for buttons that allow users to navigate to the previous transaction instance. This button is displayed on the first screen of a screen set when being displayed by an action with looping behavior.
- **Next Screen Details:** The label for buttons that allow users to navigate to the next screen in the current screen set.
- **Next Screen (no back up) Details:** The label for buttons that allow users to complete the current instance of a wizard in a loop and start the next iteration; or to move from one wizard to the next when multiple wizards are displayed by the action.
- **Complete Action Details:** The label for buttons displayed on the last screen of a screen set, when there are no additional screen sets displayed by the action and when the current screen set is not being displayed in a loop.
- **Complete Action Details:** The label for buttons displayed on the last screen of screen set being displayed in a loop and that will end that loop.

Platform Screens List

The Properties view for a platform definition within the Agentry Editor includes a Screens tab. This tab lists all screens within the same parent screen set of the platform. Within this list the screens to be used by the platform can be selected. The screens listed here are not child definitions to the platform, but rather a children of the screen set. If a new screen is added to the screen set by starting the Add Screen Wizard from the platform view, that screen will automatically be used by that platform.

List Screen

A list screen definition displays an object collection property on the Agentry Client. Object instances from the collection are displayed as rows in the list. A list screen contains the child definitions column and button. A column is defined to display the property value for each object instance in the collection. Buttons are defined to execute actions related to the object instances. List screens include definable behaviors related to filtering, scanning, and sorting, as well as other screen enhancements for displaying data stored in the object instances of the target collection property.

The list screen may or may not display a header label above the list control. A header label can contain static or dynamic text about the items displayed in the list. A list screen may also display a detail pane containing static or dynamic text. The detail panes intended usage is to display the property values of the currently selected object in the list control, reducing the need for horizontal scrolling on the Agentry Client.

List screens can be defined to include double-click actions, executed when the user double-clicks an item in the list control, scanning actions and scan filtering, and include rules to determine what items are displayed in the list. A list screen can also be enabled or disabled via a rule definition. Disabled screens are not displayed in the screen set on the Agentry Client.

List Screen Child Definitions

- **Column Definition:** A list screen column defines what object property is displayed for each record in a list control and how it is formatted on the screen.
- **Button Definition:** A button definition defines a button control to be displayed for the screen that will execute an action or display a menu when selected.

List Screen General Attributes

The General Screen attributes set the basic behavior of the List Screen, including how Styles can be applied to the List Screen.

General Attributes

- **Name:** The internal name of the list screen. This value must be unique among all screen definitions, regardless of type, within the same parent screen set.
- **Caption:** Labels the tab on the Agentry Client for the list screen. This value may or may not be displayed when there is only one screen displayed within the parent screen set, depending on the client device type.
- **Screen Icon:** This is a reference to an image definition within the application. This image is used as the icon displayed for this screen in tabs.
- **Collection:** References the object collection property the list screen is to display. This collection is normally a property of the object definition the parent screen set is defined to display.
- **Enable Rule:** References a rule definition expected to return a Boolean value and that is evaluated in the context of the object definition for the parent screen set. When false is returned, the screen will be disabled and no tab for it will be displayed within the screen set window. If all screens within a screen set are disabled, that screen set will not be displayed and any actions defined to display it will also be disabled. If the main screen set for a module is disabled, that module cannot be displayed on the Agentry Client.
- **Include Rule:** References a Rule definition expected to return a Boolean value and that is evaluated once for and in the context of each object in the collection displayed by the list screen. When an include rule is specified, only those objects for which the rule evaluates to true will be listed in the screen's list control.
- **Icons Image:** References an image definition to be displayed on the tab for the list screen, to the left of the screen's caption text, within the screen set window on the Agentry Client.

The name of this image may be selected from a list, or it may be returned from a rule. When a rule is referenced, it is expected to return a string value and is evaluated in the context of the object displayed by the parent screen set.

List Screen Styles

- **Screen:** The style to apply to the list screen as a whole. This will affect all portions of the screen not displaying a list, header label, detail pane, or button.
- **Header Label:** The style to apply to the list screen's header label. If no header label is defined this attribute has no affect on the screen.
- **Column Labels:** The style to apply to the column labels on the screen's list control.
- **Rows:** The style to apply to all rows on the list screen. The Hyperlinks optional style will override the Rows style for cells with hyperlinks.
- **Alternate Rows:** The style to apply to every other row in the list, beginning with the second row. The Hyperlinks optional style will override the Alternate Rows style for every other row, specifically cells containing hyperlinks within the row.
- **Highlight Rows:** The style to apply to a row for the purposes of drawing attention to that row. This style should always be returned via a rule definition that evaluates the object being listed. The optional Hyperlinks style will be applied to the highlighted row's cells containing a hyperlink.
- **Selected Rows:** The style to apply to the row currently selected by the user in the list control. The optional Hyperlink style will be applied to any cells within the selected row containing a hyperlink.
- **Selected No Focus Rows:** The style to apply to the selected rows in a list view control or list screen where the list control does not have the input focus. The optional Hyperlinks style will be applied to any cells within the selected row containing a hyperlink.
- **Detail Pane:** The style to apply to both the foreground (text) and background of the list screen's detail pane. If no detail pane is defined this attribute has no affect on the screen.
- **Buttons:** The style to apply to all button definitions on the screen.
- **Focused Buttons:** The style to apply to the button that currently has the focus.

Actions/Sorting Attributes

The Action/View/Selection attributes control how the user interacts with the List Screen, including double-clicking on or off an item in the list and behaviors related to sorting and reordering the columns.

- **Double-Click On Item - Action:** Specifies the action to execute when the user double-clicks a list control record.
- **Double-Click On Item - Target:** Specifies the target of the Double-Click On Item Action. A target must always be specified for the action and is typically the selected object in the list.
- **Double-Click Off Item - Action:** Specifies an action to be executed when the user double-clicks the list without clicking on an item. This is most commonly used to execute an action that instantiates an add transaction for the object type being listed.

- **Double-Click Off Item - Target:** Specifies the target of the Double-Click off Item Action. A target must always be specified for the action. Typically the target is the parent object of the object collection property displayed by the list screen.
- **Fixed Sort Property:** Specifies the property definition within the object type being listed used to sort the records in the list. The user will not be allowed to re-sort the list when this attribute is set. The Order option to this attribute is set to specify the sort order, either ascending or descending.
- **Allow Sort:** Specifies if the user can resort the list by clicking on a column header. This is enabled by default, and is disabled if a **Fixed Sort Property** is set.
- **Initial Sort Column:** Specifies a column definition by which the list will be sorted upon initial display of the screen. This attribute requires that a column definition exists before it can be set. The Order option to this attribute is set to specify the sort order, either ascending or descending. If the list screen allows the list to be sorted (**Allow Sort** is true) the list will be displayed sorted to the order of the last sort action. If a **Fixed Sort Property** is set, this attribute is disabled.
- **Allow Reorder:** Specifies whether or not the user can reorder the columns displayed in the list by dragging and dropping the column headers in the list. This is enabled by default.
- **Allow Filter:** Specifies whether or not the user can filter the items in the list. A filter icon is displayed at the bottom of the screen when enabled. The user can click this icon to select sorting options. This sets the filter behavior for the entire list screen. This is enabled by default. Individual column definitions may be defined to prohibit filtering on those columns.
- **Allow Multi-Row Select:** Specifies if the user can select more than one record in the list at the same time. If multiple items are selected in a list, actions that target the selected object in the list will be executed once for each selected object. The default for selecting multiple objects requires a `Ctrl+Click` combination or a click and drag operation by the user, depending on the device type. The **Enable Single Click** option may be set to allow multiple records to be selected with a single click by the user. Deselecting a record requires the user to click it again. This feature is normally most useful on touch screen devices using a stylus, as it allows non-sequential records in the list to be selected.

Header/Detail Pane Attributes

Using these attributes, you can display Header text and a Detail Pane in addition to the main list control of the List Screen.

Header and Detail pane attributes are set to display additional information about the list as a whole or about the currently selected item in the list. The Header Label is a static line of text displayed above the list. This text may be static, set via certain available format strings, or set via a rule. A rule referenced for this purpose is expected to return a string value and is evaluated in the context of the object displayed by the parent screen set.

The Detail Pane is redrawn each time a new object is selected in the list and almost always contains either format strings or is set via a rule's return value. Rules are evaluated in the context of the selected object in the list and are expected to return a string value.

- **Header Label:** Specifies the Header text for the list screen. A common use for this header label is the total number of objects displayed in the list vs. the total number of objects in the collection, which may be different when a filter is enabled. The format strings used for this purpose are `%DisplayedCount` and `%TotalCount`.
- **Detail Pane:** Displays a text box on the list screen. The detail pane is updated each time the user changes their selection in the list screen.
- **Position:** Controls where the detail pane is displayed on the screen in relation to the list control. You can position the detail pane below it or to its right.
- **Size:** Sets the pixel size of the detail pane on the screen. The default is 50. If the **Position** is “Bottom” the detail pane will span the width of the screen and the **Size** will set its height. If the **Position** is “Right” the detail pane will span the height of the screen and the **Size** will set its width.
- **Word Wrap:** When enabled, lines of text longer than the width of the detail pane will be wrapped to the next line. When disabled, text will continue off the screen. The user will need to scroll the detail pane to view the text.
- **Format:** Sets the values displayed in the Detail Pane. This pane can be set to a combination of static text and format strings, which take the form `%propertyName`. The `propertyName` is the name of a property defined within the selected object and will be updated with the value of that property each time a different object is selected. It may also be set to the return value of a rule, which is evaluated in the context of the selected object instance and is expected to return a string.

Scanner Attributes

The scanner attributes for a list screen affect only those list screens used by a scanner platform within the screen set and only when the list screen is displayed on a client device with a barcode scanner. At least one column definition within the list screen must be defined to support scan filtering.

A scanned value will be compared to the column(s) defined for scan filtering and only those matching this value will then be displayed. Actions may be defined when only one record matches the scan filter and when no records match.

- **Single Match Action:** Specifies what action is executed when a scanned barcode value matches one of the records displayed in the list screen. The target of the action will always be the record found to match.
- **No Match Action:** Specifies what action is executed when the scanner filter criteria does not match any records on the list. This is optional. The target of the action is the object that is the parent to the collection property displayed by the list.
- **Label Type:** Specifies what barcode types are accepted by the Agentry Client. If no Label Type is specified, all types supported by the client device will be supported.
- **Minimum Value:** The minimum number of characters accepted by the Agentry Client from the device scanner.

- **Maximum Value:** The maximum number of characters to be accepted by the Agentry Client from the device scanner. If the value scanned in contains more characters, it will be ignored.

List Screen Column

A column definition defines what object property is displayed in a list control column. The column definition also controls behaviors such as formatting, sorting the list on the column, whether or not the column can be resized or moved, and whether or not the list can be filtered on the column. Columns may also be defined to execute an action via hyperlink control.

In addition to or in place of a property value, a column may also display an image definition as an icon, which can be different for each record based on a Rule definition.

Column Attributes

- **Name:** Internal name for the column definition. This value must be unique among all columns definitions in the list screen.
- **Label:** Specifies the label for the column header. This text is displayed at the top of the column on the Agentry Client to identify the contents of the column.
- **Object Property:** Specifies the property to display in the column on the list screen. Set this to None, to display either a value derived from a format string or only an icon image. Selecting both an Object Property and specifying an icon image will display both in the column.
- **Enable Rule:** References a rule definition evaluated in the context of the object displayed by the screen set and expected to return a Boolean value. When the rule returns true, the column is enabled and displayed on the Agentry Client. When it returns false, the column is disabled and not displayed.
- **Icon Image:** References an Image definition within the application to specify an icon for the column. The image name can also be returned using a rule definition to dynamically determine the image to display for each record. This rule is evaluated in the context of the object instance for the record and is expected to return the name of an image definition as a string. Note that not using a rule for this attribute will display the same image for all records in the list
- **List Filter:** Specifies if the column should be included in those listed in the filter dialog for the list. This attribute is ignored in filtering has been disabled for the list screen.
- **Scanner Filter:** Enables scan filtering functionality for the column. When this attribute is enabled, the value scanned in by the device will be compared to the values of the column to create a filter. Multiple columns can be defined for this behavior. However, the values in the columns should be mutually exclusive. The order of the columns evaluated against the scanned value is undefined. This attribute is only supported for screens used by a scanner platform and displayed on a scanner-enabled device.
- **Format:** Can contain a format string to display one or more property values from the object type being displayed by the list in a different format than the default for the property's data type. This text can also be set via a rule definition, where the expected

return value is a string and is evaluated in the context of the object instance for the record in the list. To set the format attribute set the **Object Property** attribute must be set to None.

- **Column Width:** Specifies the initial size of the column on the Agentry Client. The user can resize the columns if the list screen definition has not disabled this behavior. If the user changes the width of a column, the new width is saved by the Agentry Client and will override the **Column Width** attribute.
- **Hyperlink:** Specifying a hyperlink action enables each cell within the column to execute an action when the user single or double clicks on the hyperlink drawn in that column. The text of the hyperlink will be the value the column is defined to display. This functionality can include columns with images. Hyperlink contains two attributes:
 - **Hyperlink Action:** Specifies the action that will be executed when a user single-clicks a column in a populated row in the list.
 - **Hyperlink Target:** Specifies the target of the Hyperlink Action.

Detail Screen

A detail screen definition displays a single instance of an object, transaction, or fetch on the Agentry Client. The properties of the definition instance are displayed in fields, a child definition to the detail screen. Definable behaviors of a detail screen are predominantly controlled by the screen's child field and button definitions, which can include read-only or read-write values within the fields, as well as numerous field type behaviors. Detail screens for transactions and fetches do not have the child definition button.

The detail screen definition contains attributes for the screen's caption, enabling and disabling the screen, and the initial focus of the screen. The detail screen is separated into multiple rows and columns, based on the definition. These row and column positions are used to specify the location of fields on the screen.

The values of the definition instance displayed by the detail screen are exposed to the user via the field definitions.

Detail Screen Child Definitions

- **Detail Screen Fields:** A detail screen field defines field controls for display on a detail screen to display data to and capture data from the Agentry Client user.
- **Buttons:** A button definition defines a button control to be displayed for the screen that will execute an action or display a menu when selected. Detail screens only have button definitions when the parent screen set is defined to display an object.

Detail Screen Attributes

General Settings

- **Name:** Internal name for the screen definition. This value must be unique among all screen definitions within the same parent screen set, regardless of screen type.
- **Caption:** Labels the tab on the Agentry Client for the detail screen when a part of an object screen set. For transaction a fetch screen sets, the detail screen caption text is displayed in

the title bar of the window on the Agentry Client. This value may be set to a rule. This rule is evaluated in the context of the definition instance being displayed, and is expected to return a string value.

- **Screen Icon:** This is a reference to an image definition within the application. This image is used as the icon displayed for this screen in tabs.
- **Enable Rule:** References a rule definition evaluated in the context of the definition displayed by the parent screen set and expected to return a Boolean value. When the return is false, the screen will be disabled and will not be displayed to the user.
- **Rows:** Sets how many rows the screen will contain. This attribute is used to divide the screen into rows, which are referenced by the field definitions to determine the position of each field on the screen. The default settings will vary depending on the platform using the screen. The grid created by the **Rows** and **Columns** attributes is not displayed on the screen at run time, but is visible in the Agentry Editor for development purposes.
- **Columns:** Sets how many columns the screen will contain. This attribute is used to divide the screen into columns, which are referenced by the field definitions to determine the position of each field on the screen. The default settings will vary depending on the platform using the screen. The grid created by the **Rows** and **Columns** attributes is not displayed on the at run time, but is visible in the Agentry Editor for development purposes.s
- **Initial Focus:** Sets the field to be the initial focus when the screen is first displayed on the Agentry Client. This attribute requires that fields have been defined for the detail screen.
- **Label Position:** Specifies the position of the label text for all fields displayed on the detail screen. The options for this attribute are either Left or Top, with the Left being the default.

Detail Screen Style Attributes

- **Screen:** The style to apply to the screen as a whole. This will affect all portions of the screen not displaying a field or button.
- **Fields:** The style to apply to all fields displayed on the screen.
- **Focused Fields:** The style to apply to the field that currently has the focus.
- **Hyperlinks:** The style to apply to any labels define to be hyperlinks.
- **Buttons:** The style to apply to all buttons on the screen.
- **Focused Buttons:** The style to apply to the button that currently has the focus.

Images Attributes

- **Screen Background Image:** This attribute allows for the selection of an image definition within the application which is displayed as the background image for the detail screen. This behavior is currently only supported on detail screens used by iOS and Android platforms.
- **Fit to Screen (Lock Aspect Ratio):** This attribute specifies that the image should be resized to fit within the viewable area of the screen. The aspect ratio of the original image is maintained. This attribute is mutually exclusive from **Fit to Screen (Stretch)** and **Crop to Screen**.

- **Fit to Screen (Stretch):** This attribute specifies that the image is to be resized to fit within the viewable area of the screen. The aspect ratio of the original image is not maintained and the image will always fill the entire viewable area of the screen. This attribute is mutually exclusive from **Fit to Screen (Lock Aspect Ratio)** and **Crop to Screen**.
- **Crop to Screen:** This attribute specifies that the image is to be cropped to fit within the viewable area of the screen. Images larger than the viewable area of the screen will not be fully displayed if this attribute is selected. This attribute is mutually exclusive from **Fit to Screen (Lock Aspect Ratio)** and **Fit to Screen (Stretch)**.
- **Background Image Position:** This attribute specifies the position of the image within the viewable area of the screen. There are nine radio buttons displayed for this attribute, each corresponding to the position of the image on the screen both vertically and horizontally.
- **Field Opacity - Fields Cover Images:** This attribute sets the opacity of the image displayed on the screen. If selected, fields on the detail screen will always be displayed on top of the background image. If not selected, the image will overlay the fields on the screen.

Button

A screen button defines a button control to be displayed on a Client screen. The button may be displayed as a standard button control, a tool bar button, a menu or menu item, or as a separator. A button is defined to execute an action when clicked or tapped, unless defined as a menu or separator. When executing an action the button also defines the target object instance provided to the action for processing.

The button definition itself allows for different Button Types. These include a traditional button, called an Action Button; an item to be added to the Action menu displayed on the Agentry Client's menu bar, called an Application Menu; a Toolbar Button, which is displayed on the Agentry Client's tool bar; and a Separator button, which places extra space between other button definitions, or a separator line in a menu.

In addition, an Action Button can be defined to be a Popup Menu button. In this case, the button displayed on the screen will not execute an action, but instead display a menu when clicked. The contents of the menu are other button definitions for the same screen that will execute actions when selected. These other buttons must meet the criteria of, first, being positioned after the menu button, and second, the Popup Menu attribute must be set to true.

All button types except for the separator are defined to execute an action when clicked. The action definition to execute must exist before creating the button definition. Buttons also include a target attribute where the object instance targeted by the action being executed is specified. The selected target object type must be the same as the object type selected for the action being executed, with the exception of those actions defined with a For Object attribute setting of None.

Button Attributes

Following are the attributes for a button definition. Some of these attributes are not applicable to a button definition based on the selection made in the **Button Type** attribute. The attribute descriptions in this list specify this information:

- **Button Type:** This attribute specifies the type of button to define for the screen. The options are:
 - **Action Button:** Displays a button control on the screen at the position specified by the platform using the screen at run time.
 - **Application Menu:** Adds a menu item to the Agentry Client's Action menu. This menu item will only be a part of the Action menu when the parent screen to the button has the focus.
 - **Separator:** Places extra space between Action Buttons, or a separator in a popup menu, depending on where the separator button is displayed Separators cannot be added to the Agentry Client's Action menu.
 - **Toolbar Button:** Places a button on the Agentry Client's toolbar. This button type must have an image as it will not have a label.
- **Name:** The unique name for the button definition. This value must be unique among all buttons within the same screen.
- **Image:** This attribute references an image definition within the application to be used as the icon for the button control displayed on the screen. For action button types the icon is displayed to either the left or right of the button's label depending on the device's OS shell. The image icon for toolbar buttons is required. This will be the image used to identify the toolbar button. For both Separator and Application Menu Button Types, or if the Action is set to Popup Menu, the image attribute field is disabled.
- **Label:** This attribute specifies the label to identify the button on the screen. This value is the label for Action Button Types, or the text listed as a menu item for both Action Menu Button Types or Action Buttons included on a popup menu. This attribute is disabled for both Toolbar and Separator Button Types.
- **Action:** This is the action to execute on the Agentry Client when the button is clicked or tapped by the user. This action must be defined before creating the button definition. At run time if this action is disabled, the button will also be disabled. This attribute may also be set to Popup Menu. In this case the button will not execute an action, but rather will display a popup menu when clicked or tapped. The items in this menu will be other button definitions within the same screen defined to be drawn on the popup menu. Popup menu buttons do not have an image or a target object instance. Also, the Popup Menu attribute is not available, as a popup menu button cannot be placed within another popup menu.
- **Target:** This attributes specifies the target object instance of the button to be passed to the action the button executes. The object type selected here must match the definition type defined in the For Object attribute in the action the button is defined to execute. At run time if the selected Target object instance is not currently in scope, the button will be disabled. As an example, if the target is the selected object in a list screen, and no object is currently selected, there is not valid target in scope and the button will be disabled. Separator Button

Types and buttons with an Action attribute setting of Popup Menu do not have a target as they do not execute an action.

- **Popup Menu:** This attribute specifies whether the button should be displayed in a popup menu on the Agentry Client. If this attribute is set to true, and of a button definition positioned before the current definition is defined with Action attribute of Popup Menu, the current button definition will be added as a menu item rather than a button control. This attribute may only be set for Action Button Types.
- **Style:** This attribute specifies a style to apply to the button definition. The Style attribute is only available for Action Button Types.
- **Focused Style:** This attribute specifies a style to apply to the button when the button has the focus. This attribute is not available for Separator Button Types.
- **Shortcut Key:** This attribute specifies whether a shortcut key is associated with the button and the specific key or key combination. This attribute includes the ability to set combinations of the `Ctrl`, `Alt`, and `Shift` keys, as well as any alphanumeric keys, function keys from `F1` through `F24`, or hardware buttons (Button 1 through Button 5) on mobile devices. When setting this attribute, verify the key combination selected is not configured for any other shortcut, either within the current screen of the mobile application or for any system shortcuts configured on the client device.

Detail Screen Fields

A detail screen field defines a field control for display on the parent screen. The field displays data to the user and, when displaying a transaction or a fetch, can capture data from the user. A field can be defined to have one of several edit types that will affect both the appearance and behavior of the field on the screen, especially when capturing data.

There are several different edit types that may be selected for a field definition. This edit type will significantly impact the field's behavior on the Agentry Client. Despite this, however, there are several attributes that are common among most fields regardless of edit type. For many field edit types these common attributes are the only attributes. For others there are additional attributes specific to the edit type selected for the field definition.

In many use cases a field definition will target a property within the definition it displays. The value of that property will be displayed to the user and, for transaction and fetch screens, the user may be able to edit that value. In these situations, the value of the field will be assigned to the property when the user advances the wizard past that screen. This may be the case when the user clicks a next button or finish button.

The field may also target other definitions within the application. If the target of a field is not a property definition, the value of the field will not be copied to that definition. It will only use it as a data source for the value to display. These targets can be selected using the target browser and can include other fields on the current screen or other screens within the same screen set.

With the release of the Agentry Mobile Platform version 5.1, when the target is another screen field that is one of the edit types for displaying complex tables, it is possible to select a complex table record field from the currently selected record in that target screen field. In

previous releases it was necessary to define an update rule for the field that would retrieve the complex table record and field to display in the screen field. The additional target browser behavior negates the need to define such a rule. Existing applications using an update rule for this purpose will still behave correctly, and can be modified to use the new behavior or left as is with the same result.

The target for a screen field can also include a field on screens in other screen sets, provided those screen sets currently exist on the Agentry Client, but are hidden from view due to the focus being on the current screen.

A field on a wizard screen displaying a property value will enforce the data limits of that property. This means minimum and maximum values or string lengths defined for the target property will be enforced by the field definition. For strings, no more than the maximum number of characters may be entered. For numeric values, the target property's attributes related to precision and maximum values will be enforced. For minimum values the user will receive an error message when trying to advance the wizard of either the minimum number of characters or the minimum numeric value has not been entered.

The labels for a field may be defined as static text or as a hyperlink. Hyperlink labels may only be defined for a field displayed on a detail screen that displays an object instance. When a label is defined as a hyperlink, an action is defined to be executed when the user clicks that label.

Fields may have their displayed value set through an update rule. These fields can still target a property, normally for transactions and fetches, in which case the value of the field as set by the update rule will be the value assigned to the property when the user advances the wizard. When displaying an object, there is normally no reason to target a property with a field definition whose value is set via an update rule.

Fields may also be hidden and/or disabled via rule evaluation. A hidden field will not be displayed on the detail screen. An optional behavior related to a hidden field is disabling that field when it is hidden. A field may also be disabled via a separate rule independent or in lieu of a hidden rule. A disabled field on a wizard screen will not enforce any required values as defined by the target property.

Fields are positioned and sized on the detail screen using the columns and rows into which the detail screen is broken up. The position of a field is set based on the upper left corner of the field and is specified using the row and column position. The width of the field is specified in columns, and the height is specified in rows, counting from the position in the field which its placed.

Common Field Attributes

The following attributes are common to most or all field edit types and result in the same behaviors for most of the different types of fields.

- **Object/Transaction Property:** Sets the property definition or other definition whose value is displayed by the field and/or that is updated with the field's value. This definition

is said to be “targeted” by the field. This attribute can be set to “-- None --”, in which case the value displayed by the Field must come from some other source.

- **Name:** The unique internal name of the Field definition. It must be unique among all fields within the same detail screen. This is commonly set to match the name of the property the field targets.
- **Label:** Sets the label for the field. This text is displayed on the left side of the field. This label text will be rendered as a hyperlink if that behavior is also defined. This value is optional and if not set no label nor the space for one will be displayed on the screen.
- **Placeholder:** This attribute references a rule definition which returns a string value used as the place holder for the field definition.
- **Edit Type:** Sets the edit type for the field, selected from a list. This may also be set to “-- Default --”, in which case the edit type of the field will match the data type of the property being displayed.
- **Read-only:** Sets the field to be read-only or read/write. Fields targeting an object property are always read-only and are not affected by this attribute. Fields with any other target will respect the **Read-only** attribute setting.
- **Shortcut Key:** Sets a key or key combination that, when entered by the user, will set the focus to the field on the detail screen. This can include both keyboard keys and hardware keys on the client device.
- **Format:** Sets any format text for the value displayed in the field. If using a format string the Object/Transaction Property attribute should be set to “-- None --”.
- **Label Width:** Sets the number of characters that can be displayed in the space given to the label on the Screen. Character size will vary depending on the font used for the label text. The total size of a field on the screen does not change based on the Label Width. The amount of space within the specified size that is given to display the field itself is decreased as the Label Width is increased. Label text longer than the space provided based on the Label Width is word wrapped on the screen.
- **Position - Column & Row:** Sets where the upper-left corner of the field will be displayed. The column and row specified correspond to the number of columns and rows the detail screen is defined to contain.
- **Size - Width & Height:** The Size attributes specify the Width and Height of the field. The Width is set to the number of columns the field should span and the Height is set to the number of rows.

Rules/Hyperlink/Special Value Attributes

- **Change Focus:** Sets if the field will keep the focus of the screen. If this attribute is checked, when focus is set to the field, it will automatically be redirected to the next field on the screen. When unchecked, the field will keep the screen focus until the user selects another control.
- **Update Rule:** References a rule definition evaluated in the context of the definition being displayed and expected to return a string value. This rule is evaluated each time the user interacts with any part of the detail screen. The value returned by the rule is displayed in the

field. Note that this rule will not change the value of the field if it returns the same value for two or more consecutive evaluations.

- **Hidden Rule:** References a rule definition evaluated in the context of the definition being displayed and expected to return a Boolean value. If the rule returns true, the field will be hidden on the detail screen. If false, the field will be displayed.
- **Disable When Hidden:** When checked, the field will be disabled whenever the Hidden Rule returns true. The Enable Rule will not be evaluated. If unchecked, then the Enable Rule will determine whether the field is enabled independently of whether or not the field is hidden.
- **Enable Rule:** References a rule definition evaluated in the context of the definition being displayed and expected to return a Boolean value. When the rule returns true the field is enabled. A false return will disable the field. A disabled field will appear grayed out, and the user will not be able to interact with it. A disabled field will also not update its target property and no attributes related to the required property value will be enforced.
- **Clear When Disabled:** When checked, the field will clear any value in the transaction property it targets if the field is disabled. Disabled fields include are those disabled by the **Enable Rule**; or those with the **Disable When Hidden** attribute is true and the field is hidden by its **Hidden Rule**. This attribute only affects fields with the following edit types:
 - Boolean
 - Date
 - Date and Time
 - Decimal Number
 - Duration
 - Identifier
 - Integral Number
 - String
 - Time
- **Pattern Recognizer:** This attribute enables or disables the behavior of recognizing certain patterns within text values of the field; e.g., e-mail addresses or phone numbers. When set to true, the user can hold down the hyperlink text to invoke some OS-defined operation. Examples may include allowing the user to compose and send an e-mail; or to send a text message or place a call to a phone number. This functionality is available on iOS Agentry Clients version 6.0.6 or later and only when the application is configured in an Agentry Editor version 6.0.8 or later.
- **Hyperlink - Action:** This attribute references an action and when set will enable the hyperlink behavior for the field's label. The label itself will be displayed as hyperlink and the user will be able to click on the label to execute the defined action. This behavior is only valid for fields displayed on an object screen.
- **Hyperlink - Target:** Sets the target object for the Hyperlink Action.
- **Hyperlink - Shortcut Key:** Sets a shortcut key for the hyperlink. When this key combination is entered on the Client, the defined Hyperlink Action is executed.
- **Special Value:** Sets a default value for the field. When a field has a Special Value defined, a radio button is displayed on the detail screen as a part of the field's definition. It is drawn

between the label for the field and the actual field control. A second radio button is also drawn to the immediate left of the field control. When the first radio button is selected, the Special Value defined for the field is set as the field's actual value, which will then update the property targeted by the field. When the second radio button is selected, the field control itself becomes enabled, and the user can enter a value.

- **Display Value:** The value to display in the field when the property value is equal to the field's special value. This only impacts fields on detail screens displaying an object instance.
- **Auto Label and Width:** This attribute can contain a label for the first radio button displayed for special value fields. This label is displayed to the right of the radio button and can indicate to the user that they are selecting the default value.
- **Edit Label and Width:** This attribute can contain a label for the second radio button that enables the field control on the Agentry Client. This label is displayed to the right of the second radio button and can indicate to users that its selection requires them to enter a value.

Detail Screen Field Edit Types

Following are the different field edit types that may be selected from the Edit Type attribute's list. All field edit types include the Common Field Attributes as a part of their definition. Many edit types also include additional attributes related to their edit type-specific behavior. These edit types are denoted as such with an asterisk(*) in this list. Look to the additional information provided for these field edit types for information on their type-specific attributes.

- **Default:** Selecting this edit type option will force the field to take on the edit type matching the data type of the property it targets. If a field has a default edit type and does not target a property, the field will be a string field.
- **Barcode Scan*:** The barcode scan field edit type receives input from a barcode scanner. Use of this field type requires the device to have a barcode scanner, and for the parent detail screen to be used by a platform that supports scanning. This type of field may also accept manual input from the user, depending on how it is defined.
- **Button*:** The button field edit type defines a detail screen field with button behaviors to execute actions and capture values. This field type will draw a button control on the detail screen in any position where a field can be placed. For object screens this button may execute an action. For wizard screens the button can set the value of a property. The type of button displayed may be a push button, check box, or radio button. Check boxes and radio buttons may be grouped (meaning only one can be selected at a time) by all targeting the same property. A value can be defined for this field that will be set to the property the Field targets when the user clicks this button. There are three types of buttons: Radio Button; Check Box; and Push Button. This is the default for displaying a Boolean Property.
- **Calendar View*:** The calendar view field edit type provides an interactive calendar to display an object collection property, with each object treated as a calendar event. The objects in the collection property displayed by this field must include properties for start and end date and times, as well as other calendar related values.

- **Complex Table Drop Down*:** The complex table drop down field edit type displays unique values from a defined record field from a complex table in a drop down list. Using a succession of fields with this edit type can create a cascade. This is a representation of parent-child values where the users will be required to select a parent value first, and then select from only those values that are children of the selected parent. Use of this edit type requires the supporting structure be first defined in the complex table the field displays. A cascade can also be created using a combination of this field edit type and Complex Table List fields.
- **Complex Table List*:** The complex table list field edit type displays the records of a complex table in a list control on the detail screen. Using a succession of fields with this edit type can create a cascade. This is a representation of parent-child values where the users will be required to select a parent value first, and then select from only those values that are children of the selected parent. Use of this edit type requires the supporting structure be first defined in the complex table the field displays. A cascade can also be created using a combination of this field edit type and Complex Table Drop Down fields.
- **Complex Table Search*:** The complex table search field edit type displays the records of a complex table in a searchable list of records. This screen is displayed when the user clicks the associated button for this field type. This is a built-in screen within the Client and will display the records of the complex table in rows and columns. The user may select any index for a string field and enter search text to locate a record within the table.
- **Complex Table Tree*:** The complex table tree field edit type displays the records of a complex table in a tree control, providing a parent-child relationship to the records. This screen is displayed when the user clicks the associated button for this field type. This is a built-in screen within the Client and will display the records of the complex table in a tree control. The records are organized in this tree using the parent-child index relationships defined in the complex table.
- **Data Table Selection*:** The data table selection field edit type lists the records of a data table in a drop down list control on the detail screen. The code value of the record selected by the user is returned to the field. If the number of records is too large to fit in a drop down control, a popup dialog will display the records in a list box.
- **Date:** The date field edit type allows the user to enter a date value selected from a calendar control. The user may also manually type a date value into this field. When using the calendar control, the user clicks the ellipsis button drawn to the right of the field on the detail screen. This will display the calendar where the user can select a date by scrolling through the months. It is recommended that this edit type only be used with properties defined to be date values.
- **Date And Time:** The date and time field edit type allows the user to enter a date and time value selected from calendar and time controls, respectively. In this type of field, the user can enter a date value by selecting it from a calendar and enter a time value in the time portion of the field. It is recommended this field edit type only be used with date and time properties.

- **Decimal Number:** The decimal number field edit type captures decimal values, allowing only numeric values, a single decimal value, and a negative sign. Any other characters will not be accepted by this field type.
- **Duration:** The duration field edit type allows the user to enter a duration value in hours, minutes and seconds. This field displays a control similar to a time entry, but the values entered represent a duration of time, rather than a time of day.
- **Embedded Image*:** The embedded image field edit type displays an image definition on the detail screen that can be interactive. A transparent grid can overlay this image and each section, or “cell” within this grid can have an action associated with it. When a given cell is clicked on the Client that action will be executed. Fields with this edit type have a child definition called Cell that represents each cell in the grid overlaying the image.
- **External Data:** The external data field edit type displays controls to show the Windows File Dialog on the client to allow a file to be selected for an external data property.
- **External Field - Active X Control*:** The external field-ActiveX Control field edit type is defined to call out from a field to an active X control, passing values to the control. Use of this Edit Type also requires use of the Active X interface available with the Agentry Mobile Platform.
- **HTML:** The HTML field edit type supports the formatted display of HTML markup text, or the display of a defined URL for internet navigation.
- **Identifier:** The identifier field edit type requires the user to enter only positive integers. This edit type is intended to support the capture and storage of values intended to uniquely identify some business entity.
- **Image Capture*:** The image capture field edit type provides integration with the client device’s built-in digital camera, allowing for images to be captured and stored in properties of the application.
- **Integral Number:** The integral number field edit type allows the user to enter only whole numeric values and an optional negative sign. Any other characters will not be accepted in this field.
- **Label:** The label field edit type displays only the label portion of a field definition, excluding any actual field control. The Label edit type prevents any editing, and no field is drawn on the detail screen. As will be readily apparent in the Agentry Editor, many of the common field attributes are disabled for fields with an edit type of Label.
- **List Selection*:** The list selection field edit type displays a drop down list of values, the source of which may be an object collection, data table, or complex table. Using an include rule, you can also list a sub-set of the source items. The values listed in this field edit type are treated as a temporary data table that exists only in working memory and only for as long as the field is displayed.
- **List Tile View:** The list tile view field edit type displays an object collection property in a tiled view allowing for add and edit interaction with the collection through the field. This field type will use other screen sets containing detail screens within the same module to display each object in the collection in a list with each object displayed in it’s own tile. This can include different screen sets to display, add, or edit objects within the collection.

- **List View*:** The list view field edit type displays an object collection property in a list control on a detail screen with the same functionality as provided by a list screen. A field of this edit type contains the child definition column, matching the column child definition to the list screen definition.
- **Location:** The location field edit type is intended to display the value of a location property, displaying the latitude and longitude in degrees for the location. This field can be read-only or editable, allowing the user to manually enter latitude and longitude values. When the field targets a transaction property, it will automatically retrieve the latest location value from the GPS unit. When this field edit type displays an object property, or if it does not target any property, a rule can be written to update the field using the @GPS_LOCATION rule function to update the field.
- **Password Validation*:** The password validation field edit type requires users to enter their client password and validates the value entered against the password for the client. This entered value is hidden with character placement, displaying asterisks in place of each entered character. Includes the ability to define a message to the user when the password entered is not valid.
- **String:** The string field edit type allows the user to enter any printable character values. This field type can be used to provide a large text field to capture user input by spanning multiple columns and rows on the detail screen.
- **Signature Capture:** The signature field edit type allows for the entry of a signature on a client's screen that is stored as a bitmap image. The signature edit type cannot be selected from the edit type list. This is, rather, the default edit type for a field when that field targets a property with a data type of Signature. To display such a property the correct edit type selection is "Default." No other edit type should be used when targeting a signature property as the behavior of such a combination is undefined.
- **Tile Edit*:** The tile edit field type displays object properties in a tiled view allowing for add and edit interaction without starting a wizard screen.
- **Tile Display*:** The tile display edit type displays an object instance in a tiled view.
- **Time:** The time field edit type allows the user to enter a time of day value using a time control. This edit type will display three controls for the field to display and capture the hours, minutes, and seconds portions of the time value. It is recommended that this field edit type only be used with time property types.

Field Edit Type - Property Data Type Cross Reference

When the edit type of a field is set to "Default," the field will take on the behavior of the edit type that matches the data type of the property the field targets. The table provided here contains the cross reference between the field's edit and the property data type. This will then be the type of field displayed on the detail screen when it targets a property of the type listed and the edit type of the field is set to "Default." This refers only to the field edit type on wizard detail screens. Object detail screens will always display property values in read-only string fields.

Property Data Type	Default Field Edit Type
Boolean	Button (check box type)
Collection	read-only string field (field should have edit type specified).
Complex Table Selection	Complex Table Search List
Data Table Selection	Data Table Selection
Date	Date
Date and Time	Date and Time
Decimal Number	Decimal Number
Duration	Duration
External Data	External Data
Identifier	Identifier
Integral Number	Integral Number
Object	read-only string field (field should have edit type specified)
Signature	Signature
String	String
Time	Time

Field Definitions With Edit Type-Specific Attributes

Many of the field edit types available include attributes beyond those common to all fields. These edit type-specific attributes are necessary to define the behaviors specific to a given field's edit type. As an example, a field defined to display complex tables will require attributes that specific the complex table to be displayed, the index used to sort the records, and so on.

The following section lists each of these field edit types and describes their type-specific attributes.

Barcode Scan

The barcode scan field edit type receives input from a barcode scanner. This type of field may also be defined to behave like a string field, accepting input from the device's keyboard. The scanning functionality is only available on detail screens used by scanner platforms on devices equipped with a barcode scanner.

Barcode Scan Attributes

Following are the attributes specific to a barcode scan field edit type. These are in addition to the common field attributes:

- **Label Types:** This attribute specifies the name of the barcode label type or types to support for this field. If the barcode being scanned is not one of these types, it will not be scanned. If this attribute is left blank, any label type supported by the device will be scanned.
- **Minimum Length:** This attribute specifies the minimum number of characters to scan in. If the number of characters is less than this minimum, the value will be ignored. The default for this attribute is no minimum. This value must be less than or equal to the **Maximum Length** attribute value.
- **Maximum Length:** This attribute specifies the maximum number of characters to scan in. If the number of characters is greater than this maximum, the value will be ignored. The default for this attribute is no maximum. This value must be equal to or greater than the **Minimum Length** attribute value.
- **Allow Typing:** This attribute specifies whether or not the user can type a value into the field in addition to scanning one in. If true, the user can type a value directly into the field.
- **Show Scan Button:** This attribute specifies whether or not a **Scan** button is drawn to the right of the barcode scan field. This button is labeled “Scan” and will activate the device’s scanner just as if the hardware scanner button is pressed.
- **Maintain Scan Focus:** This attribute specifies whether or not the scan focus should always exist for the field when displayed on the current screen. When selected, and when the user scans a barcode the value scanned in will be set to the barcode field regardless of where the current input focus may be on the screen.

Button Field Edit Type

The button field edit type defines a detail screen field with button behaviors to execute actions and capture values. A value can be defined for this field to set the value of the target transaction property of the field. When displaying an object the button can execute actions. Part of the definable behavior is the type of button to display, which may be a radio button, check box, or push button. This is the default edit type for fields targeting Boolean properties.

Included in the functionality of a button field is the ability to group multiple button fields on a detail screen. If multiple button fields are defined for the same detail screen, and they also target the same property, these button fields are then grouped. The resulting behavior of such a configuration is that only one of the buttons may be selected at the same time. This is most commonly the case when the button fields are defined to display radio button controls. However, this same behavior will be exhibited for any of the button display types.

Buttons fields defined for object detail screens should be used to execute actions. Button fields for wizard detail screens displaying a transaction or fetch should be defined to set the value of a property.

An image can be defined for display in place of one of the available button controls. In this situation, the image referenced can be an image list, with each image in the list being a square

and the same size. Which image is used is based on the state of the button field. These images are then used based on their position, as follows:

1. Enabled, not selected
2. Enabled, selected
3. Disabled, not selected
4. Disabled, selected

Button Field Edit Type Attributes

Following are the attributes specific to a Field with an Edit Type of Button.

- **Button Type:** This specifies the kind of button control to be drawn for the field. This can be a radio button, check box, or a push button.
- **Value When Selected:** This attribute specifies the value to be assigned to the target property of the field when that field is displayed on a transaction or fetch detail screen. For a button field on a transaction, fetch, or object detail screen, if the target property matches the value set for this attribute, the button will be in a selected state.
- **Action When Selected:** This attribute is only enabled for button fields when the parent detail screen displays an object. The action referenced is executed when the button is selected. Normally the **Button Type** for this situation is a Push Button. The **Target** attribute references the target object of the action referenced in the action attribute.
- **Button Image:** This attribute references an image definition to be displayed for this field. In this situation the **Button Type** attribute will be ignored and the image selected in the **Button Image** attribute will be displayed in its place. The image itself will behave as if it is a button and will either execute an action when selected or set the value of the field's target property.

Calendar View

The calendar view field edit type provides an interactive calendar to display an object collection property, with each object treated as a calendar event. Properties of the object type used should represent an event title, an event start date and time and an event end date and time. Definable behaviors include allowing users to change viewing options, setting start and end times for a work day, and days included in a work week.

As a part of the calendar view field edit type, actions may be defined for double-clicking a calendar event as well as for double-clicking a calendar day and time that currently contains no events. When defining this field edit type, it is likely the field will comprise most, if not all of the viewable screen area.

The resulting display will be a calendar control that may include events. A given event is represented by an object instance containing the event's title, and its start and end date and times. The event will then be represented in the calendar view as a block spanning the days and times as provided by the start and end date and time values. Users will be able to interact with these events and/or with empty time periods on the calendar based on double-click actions that can be set for the field.

Note that this field edit type was only supported on the desktop builds of the Windows operating system supported by the Agentry Client in versions of the Agentry Mobile Platform prior to 5.1. Release version 5.1 and later of the Agentry Mobile Platform provide support for the mobile Windows operating systems supported by the Agentry Client.

Calendar View Data/Style Attributes

The following attributes for a field with a calendar view edit type define the collection property to use, and the properties within each object instance contained in the object collection to use for displaying events in the calendar view. They also include the styles that may be applied to the calendar view events.

Calendar Data

- **Collection:** Sets the object collection property that will be used as the data source for calendar events. Each object instance in this collection will be treated as a calendar event by the field.
- **Event Title:** This attribute specifies the value to display as the title of a given event. This may be either an object property within the objects of the collection, or the return value of a rule. If set to a rule, the rule is evaluated in the context of the object instance for the event and is expected to return a string value. This rule will be evaluated once for each object currently displayed in the calendar view. Changing the view options of the calendar on the Agentry Client at run time will result in the rule being evaluated again for each object displayed.
- **Event Start:** This attribute specifies the value to treat as the start date and time for an event. This may be either an object property within the objects of the collection, or the return value of a rule. If set to a rule, the rule is evaluated in the context of the object instance for the event and is expected to return an integral number treated as an Agentry date and time value. This rule will be evaluated once for each object currently displayed in the calendar view. Changing the view options of the calendar on the Agentry Client at run time will result in the rule being evaluated again for each object displayed.
- **Event End:** This attribute specifies the value to treat as the end date and time for an event. This may be either an object property within the objects of the collection, or the return value of a rule. If set to a rule, the rule is evaluated in the context of the object instance for the event and is expected to return an integral number treated as an Agentry date and time value. This rule will be evaluated once for each object currently displayed in the calendar view. Changing the view options of the calendar on the Agentry Client at run time will result in the rule being evaluated again for each object displayed.
- **Include Rule:** Sets the name of the rule that can be used to limit the objects in the collection that will be displayed.
- **Tool Tip Rule:** Sets the rule that can be used to format a tool tip text when a user hovers the cursor over an event displayed in the calendar as an event.

Calendar Styles

- **Highlight Events:** This attribute provides the style to apply to events that should be highlighted in the calendar view. This style should be set based on a rule in any real-world

use cases. The rule is evaluated in the context of the object instance representing the event and is expected to return a string value containing the name of the style to apply to those events that should be highlighted. An empty string will use the default style.

- **Selected Events:** This attribute provides the style to apply to a selected event in the calendar view. The style to be applied may be selected from those that are defined, or the name of the style can be returned from a rule. If a rule is used, it will be evaluated in the context of the object instance representing the event when the user selects the event. The rule is expected to return a string value containing the name of the style to apply.

Calendar View Options Attributes

The calendar view options attributes define behaviors related to the options a user may set, including whether or not to allow the user to set those options. These include the different views the calendar supports, the days of the week to treat as work week days vs. weekends, and whether or not those weekend days should be compressed in the month view.

Calendar Options

- **Allow User to modify Options:** This attribute controls whether or not users can change the calendar options on the Agentry Client. If set to true, the remaining attributes set here are treated as the defaults for the calendar behavior, which the users can then override. If this attribute is set to false, the options set here will define the behaviors exhibited at all times for each view supported by the calendar.
- **View:** Sets the default view for the Calendar:
 - **Day:** Displays the current day
 - **Month:** Displays the entire month, including weekends.
 - **Week:** Displays the entire week, Sunday through Saturday, including both work and not work days.
 - **Work week:** Displays the current work week. The days in the work week are defined in the Work Week section.

Day View - These attributes affect the appearance and behavior of the calendar view when set to the Day View on the Agentry Client.

- **Time Scale:** Sets the time scale, in minutes, for the time rows in the calendar. Options for Time scale are in minutes, can be in increments of: 60, 30, 15, 10, 6, or 5.
- **Start Time:** Sets the calendar start time in increments of 30 minutes. The calendar uses a white background color when displaying times between the start time and end time.
- **End Time:** Sets the calendar end time in increments of 30 minutes. The calendar uses a white background color when displaying times between the start time and end time.

Work Week View - These attributes affect the appearance and behavior of the calendar view when set to the Week or Work Week view on the Agentry Client.

- **Sunday - Saturday check boxes:** Sets which days of the week are included in a Work Week. Unchecked days are treated as weekends.

- **First Day of Week:** Sets the first day of the week displayed on the calendar as the left-most day.

Month View - This attribute affects the calendar only when the field is set to the month view.

Compress weekends: Check to compress Saturday and Sunday in the calendar month view. Unchecked, and Saturday and Sunday will display like the other days in the week.

Calendar View Actions Attributes

The attributes for actions in a calendar view allow for the definition of actions to be executed when the users double-click an event in the calendar, and when double-clicking an open time slot within the calendar.

- **Double-Click on Event Action and Target:** These attributes allow for the definition of an action to be executed when a user double-clicks an event in the calendar, and the target object of that action. This target is normally the object representing the event just selected. In most use cases the action may display that event and/or allow the user to edit that event.
- **Double-Click off Event Action and Target:** These attributes allow for the definition of an action to be executed when a user double-clicks a time slot within the calendar that does not currently contain an event. This target is normally the parent object of the collection being displayed by the calendar view field. In most use cases the action will allow the user to add a new event starting at the day and time slot double-clicked in the calendar. The selected begin or end date may be retrieved via the rule function `SCREENFIELDVALUE` by passing the name of the calendar view field and either of the parameters `SelectedBeginDate` or `SelectedEndDate`.

Complex Table Drop Down

The complex table drop down field edit type displays a drop down list of unique values from a defined complex table field. This screen field edit type is normally used in a cascade control series, allowing users to drill down through records within the table that have a parent-child relationship.

A cascade is a series of multiple fields all displaying the same complex table. Each cascade field displays a different complex table field. The records displayed in a field are the child records to the selected record in the field before it in the cascade. The parent-child relationship is determined by the structure of the indexes for the complex table being displayed.

The overall behavior of a cascade will force a user to make a selection in the first field in the cascade, which is a top-level parent record in the complex table. The next field in the cascade will not be enabled until this selection is made. At this point, the values listed in the second field will be only those complex table records that are children to the record selected in the first screen field.

This behavior repeats for each field in the cascade. Defining such a cascade requires that the complex table displayed by these fields have the needed parent-child indexes defined. Each cascade screen field must then have a matching child table index at the same level.

As an option to displaying the values of a complex table, this field type can display a complex table search dialog. Within this dialog the records of the complex table will be listed within multiple columns, one for each complex table field. The records displayed in this dialog will be dependent on the selections made in previous fields in the cascade control series. Also as definable behaviors in this dialog are options to specify which indexes to allow a user to search on and the specific complex table fields to display in the list.

This field edit type also supports scanning as input. When this behavior is enabled, the value scanned in must be one that can return a record using the index specified for the screen field.

Both the complex table drop down and complex table list field edit types support cascade behavior. Fields of both types may be used in the same cascade series of fields on a given detail screen.

Complex Table Drop Down Attributes

Following are the attributes specific to fields with an edit type of complex table drop down. These are in addition to the common field attributes:

- **Complex Table:** This attribute specifies the complex table the field is to display. In a cascade it is possible for each field to display a different complex table, provided the values selected in one can be used to search the next. This is not the recommended method for using cascades, as it is more efficient to use a single complex table for all cascade fields.
- **Table Index:** This attribute specifies the complex table index that should be used to search that table. For the first field in a cascade, this index should be the top-level index, that is, an index that does not have a parent index. For subsequent cascade fields, the selected index should be the child index to the index selected in the field that precedes it in the cascade.
- **Cascade Parent:** This attribute references another screen field on the same detail screen to use as the cascade parent for the field. If left set to Auto, the cascade will be determined based on the selected Table Index. The field whose Table Index is set to the parent index of the selected Table Index for the field will be treated as the cascade parent screen field. If the proper index structure is in place in the complex table, the cascade parent can be left set to auto.
- **Display Field:** This attribute specifies the complex table field to display for the table records. Leaving this attribute set to Auto will display the field upon which the index selected in Table Index is defined. This is the recommended selection for this attribute. Only unique values of this field will be listed.
- **Return Field:** This attribute contains the complex table field to return from the selected record. When left set to Auto, the return field will be the field upon which the index selected in Table Index is defined. This is the recommended selection for this attribute, as this value will be the one passed to the next field in the cascade.
- **Selection Method:** This attribute specifies how the records of the complex table being displayed by the screen field are displayed to the user. Following are the options for this attribute:
 - *Always Drop Down Menu* - Always list the records of the complex table in a drop down list.

- *Always Open Dialog* - Always list the records in a popup dialog.
- *Always Open Dialog with Search* - Always open a complex table search list dialog that provides the user with the ability to enter search text to locate the desired record. This dialog will list all fields, or only those selected in the Search Dialog Indexes list, for each record in the complex table.
- *Open Dialog if Needed* - Allows to the specification of record threshold. For this option, the default is to display the records in a drop down list. If the number of records to list exceeds the defined threshold, a popup dialog is displayed listing the records in a list box control.
- **Open Threshold:** This attribute can be set only if the **Selection Method** is set to Open Dialog if Needed. This attribute can be set to the Default, which will vary from one client device to another, or to a specific number of records. When the Open Threshold is exceeded, the popup dialog is displayed, listing the records from the table.
- **Scanning:** This attribute, when set to true, will enable the client device's scanner (if present). The user can scan a barcode value, which will be used to search the complex table by that value using the defined Table Index. If a single matching record is found, that will be the selection for the field. This attribute only has an impact if the client device has a scanner, and if the parent detail screen is used by a scanner platform.
- **Handle Special Value By:** This attribute is available only if a special value has been defined for the screen field. This attribute specifies what to do when the cascade parent value changes. You can define the field to then set itself back to the special value, change to the default text of "Please select," or to change to the "Please select text" only if the current selection is not the special value.

Search Dialog Indexes

For detail screen fields defined with an edit type of complex table drop down, there are two lists of items shown in the properties view of the Editor. The first is the Search Dialog Indexes tab. Listed in this tab will be one item for each top-level index defined in the complex table the screen field is displaying. For complex table drop down lists the selected search indexes will only affect behavior when the **Selection Method** attribute is set to "Always Open Dialog with Search." Each item contains a check box which, when selected, will display that index to the user as one that can be searched on. Those indexes not checked in this list will not be displayed to the user.

Search Dialog Fields

For detail screen fields defined with an edit type of complex table drop down, there are two lists of items shown in the Properties view of the Editor. The second is the Search Dialog Fields tab. Listed in this tab will be one item for each field defined in the complex table being displayed by the screen field. For complex table drop down lists the selected fields will only affect behavior when the **Selection Method** attribute is set to "Always Open Dialog with Search." Each item contains a check box which, when selected, will display that field to the user in the list of records. Fields that are not selected will not be shown to the user.

Complex Table List

The complex table list field edit type displays the records of a complex table in a list control on the detail screen. Definable behaviors include the complex table fields to display, the complex table field value to return when a record is selected, and an action to execute when a record is double-clicked. This field edit type is normally used in a cascade control, though this is not a requirement.

A cascade is a series of multiple fields all displaying the same complex table. Each cascade field displays a different complex table field. The records displayed in a field are the child records to the selected record in the field before it in the cascade. The parent-child relationship is determined by the structure of the indexes for the complex table being displayed.

The overall behavior of a cascade will force a user to make a selection in the first field in the cascade, which is a top-level parent record in the complex table. The next field in the cascade will not be enabled until this selection is made. At this point, the values listed in the second field will be only those complex table records that are children to the record selected in the first screen field.

This behavior repeats for each field in the cascade. Defining such a cascade requires that the complex table displayed by these fields have the needed parent-child indexes defined. Each cascade screen field must then have a matching child table index at the same level.

Both the complex table drop down and complex table list field edit types support cascade behavior. Fields of both types may be used in the same cascade series of fields on a given detail screen.

Complex Table List Attributes

Following are the attributes specific to the complex table list field edit type. These attributes are in addition to the common field attributes:

- **Complex Table:** This attribute specifies the complex table the field is to display. In a cascade it is possible for each field to display a different complex table, provided the values selected in one can be used to search the next. This is not the recommended method for using cascades, as it is more efficient to use a single complex table for all cascade fields.
- **Table Index:** This attribute specifies the complex table index that should be used to search that table. For the first field in a cascade, this index should be the top-level index, that is, an index that does not have a parent index. For subsequent cascade fields, the selected index should be the child index to the index selected in the field that precedes it in the cascade.
- **Cascade Parent:** This attribute references another screen field on the same detail screen to use as the cascade parent for the field. If left set to Auto, the cascade will be determined based on the selected Table Index. The field whose Table Index is set to the parent index of the selected Table Index for the field will be treated as the cascade parent screen field. If the proper index structure is in place in the complex table, the cascade parent can be left set to auto.

- **Fields to Display:** This attribute can contain the name of each complex table field to display in the list. Each table field name is listed here, separated by a comma. If no fields are listed here, all fields are displayed in the list. Each field displayed in the list is represented by a list column.
- **Return Field:** This attribute contains the complex table field to return from the selected record. When left set to Auto, the return field will be the field upon which the index selected in Table Index is defined. This is the recommended selection for this attribute, as this value will be the one passed to the next field in the cascade.
- **Double-Click Action:** This attribute references an action to execute if the user double-clicks a record in the list. The target of this action will always be the object instance the parent detail screen is displaying. The complex table record the user double-clicks is the current record and can be accessed as such through the target browser. This attribute can only be set when the field is displayed on an object detail screen and will have no effect on a wizard detail screen for a transaction or fetch instance.
- **Handle Special Value By:** This attribute is available only if a special value has been defined for the Field. In this case, this attribute specifies what to do when the cascade parent value changes. You can define the Field to then set itself back to the special value, change to the default text of Please select, or to change to the Please select text only if the current selection is not the special value.

Complex Table Search

The complex table search field edit type displays the records of a complex table in a searchable list. This field edit type displays a field with an ellipses button. When the ellipses button is clicked, the searchable list screen is displayed. By default users may search the records of the complex table on any defined top-level index for a string field. Alternately, a single search index may be specified as a part of the screen field's definition. This field edit type also supports scanner functionality to select a record.

When scanner functionality is enabled, the scanned value will be used to search the complex table on the selected search index. Only those records that match will be listed and the user may make a selection from this filtered list.

Complex Table Search Attributes

Following are the attributes specific to the complex table search field edit type. These attributes are in addition to the common field attributes:

- **Complex Table:** This attribute specifies the complex table whose records will be listed in the search screen.
- **Search Index:** This attribute can be set to restrict the index used to search the complex table. If an index is selected for this attribute, the user will only be able to search the complex table using that index. By default, all top-level indexes on string fields can be used to search the records. The Search Index can be defined any index, parent or child, which will then be used for all searches of the complex table when using this field.

- **Parent Value:** This attribute can be set to the value by which the records should be filtered when the **Search Index** is set to an index that is a child to another index.
- **Initial Value:** This attribute can be set to a property of the definition. This will set the initial value of the field to the value of this property. The user can still select a complex table record to change this value. By default, there is no Initial Value.
- **Display Field:** This attribute can be set to any field within the complex table and specifies the field value to display in the screen field for the selected table record. By default the value displayed in the screen field is the field upon which the complex table's primary index has been defined, i.e., the field containing the unique value for each table record.
- **Return Field:** This attribute can be set to any field within the complex table and specifies the table field to return to the screen field from the selected table record. By default the value returned is the table field upon which the complex table's primary index has been defined, i.e., the field containing the unique value for each table record.
- **Allow Scanning as Input:** This attribute can enable scanner functionality for the search screen. When enabled, the user can scan a value that will be used to search the complex table using the selected search index. Only those records matching this search will be listed. This attribute will only affect screen fields for detail screens used by a scanner platform displayed on client devices with barcode scanners.

Search Dialog Indexes

For detail screen fields defined with an edit type of complex table search, there are two lists of items shown in the properties view of the Editor. The first is the Search Dialog Indexes tab. Listed in this tab will be one item for each top-level index defined in the complex table the screen field is displaying. Each item contains a check box which, when selected, will display that index to the user as one that can be searched on. Those indexes not checked in this list will not be displayed to the user.

Search Dialog Fields

For detail screen fields defined with an edit type of complex table search, there are two lists of items shown in the Properties view of the Editor. The second is the Search Dialog Fields tab. Listed in this tab will be one item for each field defined in the complex table being displayed by the screen field. Each item contains a check box which, when selected, will display that field to the user in the list of records. Fields that are not selected will not be shown to the user, unless all fields are not selected, in which case all fields of the complex table are displayed as columns in the list.

This dialog also allows the developer to specify the order in which columns should be displayed in the list control of the search dialog. For the selected fields, a position value is assigned and can be adjusted by moving the field up or down in the list.

Complex Table Tree

The complex table tree field edit type displays the records of a complex table in a tree control, providing a parent-child relationship to the records. Each node in the tree control represents a complex table record. This edit type displays a field on the detail screen with an ellipses

button. When this button is clicked the screen containing the tree control is displayed. Definable behaviors include the table indexes to be treated as the parent and child indexes, the starting point of the records, the number of levels below the start point to display, and the complex table field values to display from each record in each node of the tree control.

The complex table tree field edit type allows for the creation of parent-child relationships that do not exist in the complex table's structure. Part of the definition of a field of this type is the selection of two indexes in the complex table, both of which are top-level indexes. One will be used as the parent index and the other the child. This relationship will only exist while the tree control screen is displayed.

Complex Table Tree Attributes

Following are the attributes specific to the complex table tree field edit type. These attributes are in addition to the common field attributes:

- **Complex Table:** This attribute references the complex table whose records will be displayed in the tree control screen.
- **Parent Index:** This attribute references the index within the complex table to use as the parent index. Records will be organized in the tree control according to their common parent based on this index. Each node will contain child nodes with the same value in the field for which this index is defined.
- **Child Index:** This attribute references the index within the complex table to use as the child index. Each record with a unique value in the field upon which this index was created will be listed in the tree control under the parent record.
- **Search Index:** This attribute can be set to restrict the index used to search the complex table. If an index is selected for this attribute, the user will only be able to search the complex table using that index. By default, all top-level indexes on string fields can be used to search the records.
- **Parent Root:** This attribute can be set to a value found in the complex table field for which the selected Parent Index is defined. Any records with the Parent Root value in this field will be treated as the top-level parent records by the complex table tree field. The resulting behavior will be that these records will be listed as the root nodes in the tree control.
- **Display Field:** This attribute references a complex table field whose value will be displayed for each node in the tree control. This same value will also be the one returned to the field for display on the detail screen containing the complex table tree field definition. If this value is not set, the default is to display the field for which the complex table's primary index was defined. The Display Format attribute to this screen field edit type can also affect the appearance of the nodes in the tree control.
- **Return Field:** This attribute specifies the complex table field to return for the selected record for the purpose of setting the property targeted by the screen field. By default, the complex table field for which the primary index was defined is the value returned.
- **Display Type:** This attribute can specify how each node in the tree control will be displayed. The default is to display the table field value from the **Display Field** attribute for the record each node represents. The other alternatives are to display not only the value for that record, but also the values of each ancestor to that record. This can be in either a

parent-to-child order (Root to Selected Item), or in a child-to-parent order (Selected Item to Root). The value from each record displayed in the node can be separated in the display using the Connect Items With attribute (discussed below). The **Display Field** attribute contains the field value displayed for each record.

- **Display Format:** This attribute can contain format strings to format the display of the selected record in the detail screen field. To access the values of the selected record, use the format string syntax of `%fieldName` where `fieldName` is the name of the complex table field whose value is to be displayed.
- **Tree Format:** This attribute can contain format strings to format the display of the nodes in the tree control. Using these format strings, you can display additional complex table fields for each record in its respective node. This will be in addition to the value selected in the display field attribute. To reference a complex table field, the syntax is `%fieldName` where `fieldName` is the name of the complex table field.
- **Connect Items With:** This attribute can contain a character that will be placed in between each of the values displayed in a single node. This attribute is only available if the **Display Type** is set to either Root to Selected Item, or Selected Item to Root. The character(s) contained in the **Connect Items With** attribute will be placed between the values for each record in the hierarchy within a single node of the tree control.
- **Word Wrap:** This attribute, when set to true, will wrap the text of the nodes to the next line, if it spans beyond the viewable area of the screen. The default is to not wrap the node values, requiring the user to horizontally scroll the tree control for longer values.
- **Sort:** This attribute controls how child nodes are sorted in relation to nodes that begin with a hyphen. In many cases, a complex table will contain a default record, such as `--None--`. The Sort attribute can specify that such records are sorted either before or after the other nodes. The default setting will place these items wherever they may be sorted according to the locale settings of the client device.
- **Depth:** This attribute can specify how many levels of the hierarchy within the complex table you to display in the tree control. The number of levels refers to the number of descendents to display below the root node. This value is relative to the Parent Root, if one is specified. Note that this value does not specify the actual level, but rather the number of levels counting from the Parent Root.
- **Search:** If this attribute is enabled the search controls in the complex table tree screen will be hidden.
- **Scanning:** This attribute can enable barcode scan searches of the complex table records within the tree control. The complex table will be searched using the specified Child Index for the value scanned in. The first matching record will be selected in the tree control. This attribute only affects complex table tree fields defined for detail screens used by a scanner platform and displayed on a device equipped with a barcode scanner.

Data Table Selection

The data table selection field edit type lists the records of a data table in a drop down list control on the detail screen. Definable behaviors of this list include the data table field to display, the sort order for display, and the value by which to sort. A popup screen may be

displayed based on the number of records in the data table. This threshold is different for each supported device type.

When displaying the records from the data table in the drop down list, the code, value, or both may be displayed. Additionally, format strings may be used to format the text for each record. When a selection is made in the list, the value returned to the field is always the code portion of the selected record. This will be the value set to the target property of the field.

Data Table Selection Attributes

The following attributes are specific to the data table selection field edit type and are in addition to the common field attributes.

- **Data Table Name:** This attribute references the data table whose records will be listed in the drop down list for this field.
- **Sort By:** This attribute allows you to sort the values listed in the drop down list by one of several options: Code, which is the code field in each table record; Value, which is the value field in each record; Displayed Text, which is the text displayed for each record in the field; and Order in Data Table, which is the order in which the records are listed in the data table itself.
- **Sort Order:** This attribute specifies whether the records displayed are sorted in ascending or descending order. This is a string sort.

Field Attributes

- **Display Type:** This attribute specifies which fields from the data table records should be displayed in the drop down list. The options are: Code, meaning the code field in each table record; Value, which displays the value field from each record; Code - Value, which displays both fields from each record, separating them with a hyphen; and Format Text, which allows you to specify format strings to format the values displayed in the list for each record.
- **Format Text:** If the **Display Type** attribute is set to “Format Text” this attribute will be enabled. Format strings can then be entered in this attribute to format each record from the data table. The valid format strings for this field are %code, %value and %position. This last will display the position number of each record as stored in the data table. This last option is used mostly for testing purposes and is generally not found in the production version of an application. This attribute can also contain any other printable characters, excluding tabs and carriage returns, to format the display of the table’s records.
- **Editable:** This attribute specifies whether or not users can manually enter text values in the field for values not found in the data table displayed by the field. When this attribute is set users can either select from the list or enter a value manually. When not set, users will be required to select an item from the list. This attribute can be set if the field is defined to be read-only if the field also has an update rule defined, if that rule can return values not found in the data table.

Popup Dialog Attributes

- **Define separate display type for popup dialog:** This attribute allows you to display the records from the data table differently in the popup dialog vs. the drop down list for the field. If set to false, the display and format attributes listed above will also affect the popup dialog. If set to true, the attributes listed next will provide separate display behaviors for the popup dialog.
- **Display Type (Popup Dialog):** The options for this attribute are the same as the Display Type options listed previously. The option selected here will impact the appearance and behavior of the popup dialog displayed for larger data tables.
- **Format Text (Popup Dialog):** The format strings for this attribute are the same as the Format Text options listed previously. The format text entered here will impact the appearance and behavior of the popup dialog displayed for large data tables.

Embedded Image Field

The embedded image field edit type displays an application-level image definition on the detail screen that can be interactive. Definable behaviors include whether or not to resize the image to fit in the space allocated for the field, the cropping behavior of the image displayed, and the ability to divide the image into cells to elicit different behaviors when different portions of the image are selected.

Each cell in an embedded image field is represented by a child definition to the field in the Editor. This definition type is called an image cell. There will be as many of these image cells as there are cells in the image, which is a multiple of the rows and columns defined for the field.

Note that the embedded image field edit type was named the image field edit type in versions of the Agentry Mobile Platform. Starting with version 5.1 and going forward, this field edit type has been named embedded image. This is to distinguish this field edit type from the image capture field, which displays the contents of image properties. The embedded image field is provided to display image definitions at the application level of the application project hierarchy.

Child Definitions

- **Image Cell:** The image cell definition is a child definition to fields with an edit type of embedded image and represents a specific portion of the image being displayed. An image cell defines the action to execute or the value to set when the corresponding cell of the image field is selected by the user.

Image Field Edit Type Attributes

The following attributes are specific to the embedded image field edit type. These are in addition to the common field attributes:

- **Image:** This attribute specifies the image definition within the application that this field is to display.

- **Grid:** This attribute contains two numeric values, rows and columns. The product of these two values determines the number of image cell definitions for the field. A one by one grid will create a single cell representing the entire image field.
- **Resize to Fit:** This attribute specifies whether the image should be resized to fit in the space allotted to the field.
- **Lock the Aspect Ratio:** Available only when resize to fit is true, this attribute specifies whether or not the aspect ratio of the image should remain the same. If true, the aspect ratio will be locked. If false, an image too large for the field will be resized to the size and shape of the field, regardless of its affect on the appearance of the image.
- **Crop to Fit:** This attribute is only available when the resize to fit attribute is false. If **Crop to Fit** is true, the image will be cropped on its right and bottom edges to fit within the field.
- **Position:** This attribute specifies the position of the image within the space allotted to the field on the detail screen. This can be one of: upper-left; top; upper-right; left-center; center; right-center; bottom-left; bottom; or bottom-right.
- **Highlight:** This attribute specifies whether or not the currently selected cell on the image field is highlighted. If this attribute is true, you can specify how to highlight the cell(s). This may be either in 3D, or by specifying a mask color to be applied to the selected cell(s). Cells are considered selected if either the user selects them on the screen or if the value the cell is defined to set is equal to the value of the property targeted by the field.
- **Highlight Cells on Hover:** This attribute specifies whether or not the cell over which the mouse cursor is currently hovering is highlighted. If this is true, you can specify whether to highlight the cell in 3D or by specifying a mask color to be applied to that cell. This attribute only affects the Windows PC platforms.

Image Cell

The image cell is a child definition to a field with an edit type of image. The image cell definition represents a cell for the parent image field. A cell definition can define an action to execute or a value to assign to the field's target property when the cell is selected. Actions may be executed from detail screens for objects. Values may be assigned to properties from detail screens for transactions or fetches.

The editor allows for a single image cell to be edited within the image, or to edit multiple cell images at the same time. This is accomplished using the layout view for the image field. The grid will overlay the defined image in this view, and the cells may be selected and edited via right-clicking a cell. Multiple cells may be selected using `Ctrl+Click`. A single cell can be defined to set a value or execute an action, and then additional cells can be defined to be the same as that cell. This allows for multiple cells to be combined to define a region of the image, based on its appearance.

Image Cell Attributes

The following attributes define the behavior of the image field child definition image cell:

- **Cell:** This attribute specifies which cell the definition represents. This is a numeric value displayed in the format (*Row, Column*), where *Row* and *Column* are the points where the row and column intersect to create the cell.
- **Name:** This attribute is the name of the cell definition, which is set by default to `Cell_R_C`, where *R* and *C* are the row and column that make up the Cell.
- **Value When Selected** (Transactions and Fetches): This attribute is available for transaction and fetch detail screens and defines the value to be set to the property targeted by the cell's parent field definition when the cell is selected.
- **Action** (Objects): This attribute references the action to execute on object detail screens when the user selects the cell.
- **Action Target:** This attribute specifies the object instance that is targeted by the action executed.
- **Tooltip When Hovered Over:** This text field can contain any text value. This will be the text displayed on the client when the user hovers the mouse cursor over the cell. This attribute only impacts image fields displayed on detail screens for the Windows PC platforms.

External Field - ActiveX Control

The external field-ActiveX control edit type is defined to call out from a field to an ActiveX control. Values may be passed to this control from the Agentry Client.

Use of this field requires an ActiveX control exist on the client devices and that control be built using the Agentry ActiveX Control API, including the implementation of all Expected Methods.

Using the **Agentry Data** and **Actions** tabs allows an ActiveX control to query Agentry for data and for an ActiveX control to call for Agentry to execute actions. Agentry can also query the ActiveX control for any values listed in the **External Values** tab.

External Field - Active X Control Attributes

The following attributes are specific to the External Field - ActiveX control field edit type. These are in addition to the common field attributes:

- **ActiveX Class Name (Prog ID):** This attribute contains the class name that the Agentry Client will interface with for the ActiveX control.
- **Allow Scanning as Input:** This attribute specifies whether or not the field displayed will accept barcode scan values as input. This attribute will only impact fields displayed on detail screens used by a platform that supports scanner behavior and on client devices equipped with a barcode scanner. When value is scanned for the field, the ActiveX control expected method `AgentryUpdateScanData` to pass the barcode value to the ActiveX control.
- **External Values Tab:** The External Values tab is a list of values provided by the ActiveX Control. This will allow the Agentry Client to query the control for data. From the tab, you can add and delete value names from the list. The ActiveX control referenced by the detail screen field must include the proper processing within the

AgentryGetSpecificValue method to return the value(s) associated with each of the External Values listed in this tab.

- **Agentry Values Tab:** The Agentry Values tab is a List of names and target paths for values within Agentry, made available to the ActiveX Control. From the tab, you can link Agentry data with the external values for the ActiveX Control. Both primitive data types as well as object instances and collection properties can be made available to the ActiveX control. The name associated with the selected data item is the identifier exposed to the ActiveX control, which can call the `GetPropertyFromMappings` Agentry Client-Side API method, passing the name to retrieve the desired value.
- **Actions:** Allows the ActiveX control to call for Agentry to execute actions. The Properties tab gives you a list of Actions and target paths. Within this list actions can be added and deleted. When an action is added it must also specify a target object for the action. The ActiveX control can call the `ExecuteAgentryAction` Agentry Client-Side API method, passing the name of the action to be executed.

HTML

The HTML field edit type supports the formatted display of HTML markup text, or the display of a defined URL for internet navigation. Definable behaviors for this field include whether or not to display the navigation toolbar, a list of parameters to be passed to a URL, and the ability to provide either a list of permitted or prohibited URL's to restrict the navigation allowed by the user.

Included in this field edit type are two child definitions, which are the Domain List and the URL Parameters. The domain list items can be used to specify to which URL's users can navigate. The URL parameters can define the parameters to pass to a URL. These values are derived from a rule and can therefore be dynamic.

The HTML field edit type can also display HTML pages or text retrieved from some source, such as the back end system. This allows for a web page to be displayed within the field on the detail screen, which can then provide links to internal or external pages.

HTML Child Definitions

- **Domain:** The domain definition is a child definition to detail screen fields with an edit type of HTML, and can specify the URL's to which users can navigate or those they should be prevented from viewing.
- **URL Parameter:** The URL parameter is a child definition to detail screen fields with an edit type of HTML, and can specify an argument value to be passed to the URL the field is defined to display.

Attributes

Navigation Bar

- **Initial State-Show Navigation Bar:** This attribute specifies whether or not to display the navigation bar for the HTML field when the parent screen is initially displayed. If this

option is not set, the user can display the navigation bar by right-clicking the field and selecting the popup menu item to display it.

Domain List

- **Domain List Contains:** This attribute specifies whether the domains added to the HTML field as child definitions specify those URL's to which the user is allowed to navigate, or those URL's to which they should be prevented from viewing.

Domain and URL Parameter

The domain definition is a child definition to detail screens with an edit type of HTML, and can specify the URL's to which users can navigate or those they should be prevented from viewing. This definition type contains a single attribute of Name, which contains the URL for the domain definition. The parent HTML field then specifies whether all child domain definitions are those that are allowed to be viewed, or those that should be blocked.

The URL parameter is a child definition to detail screen fields with an edit type of HTML, and can specify an argument value to be passed to the URL the field is defined to display. The value for each URL parameter is specified via a rule definition, making the values dynamic.

Image Capture

The image capture field edit type provides integration with the client device's built-in digital camera, allowing for images to be captured and stored in properties of the application. Using this field type it is also possible to select an image file on the client device to store in the property. This field edit type is intended for use only with properties of type image.

The image capture field edit type interacts with the client device's camera, if one is available. When displayed on the detail screen, the field will include up to two buttons. One will allow the user to select a file from the client device's file system. The other will interact with the camera, taking a picture that will be captured and displayed in the field. This behavior is exhibited only on detail screens displaying a transaction or fetch. For object screens, the image capture field will display the image stored in an image property in a read-only field.

When the camera button for the image capture field is clicked, a dialog is displayed allowing the user to take a picture using the device's camera. When the image is captured it is displayed as a thumbnail in the image capture field. The user can then click this image to display a popup screen of the image. The size of the image displayed in this popup is dependent on the Initial Popup Mode attribute for the image capture field. This option allows for the image to initially be displayed in either full size or to be scaled to fit within the popup screen. This screen will be no larger than the viewable area of the client device's display. Within this popup screen the user can click the image to switch between the scaled and full size image views. The dialog will contain scroll bars to allow the user to scroll the image if it is larger than the viewable display area.

Image Capture Attributes

- **Allow Image Camera Capture:** This attribute specifies whether or not to allow the field to interact with the device's camera, if one is available. When this attribute is set, the field will include a button control that will activate the camera to take a picture
- **Allow Image File Capture:** This attribute specifies whether or not to allow the field to capture an image file stored on the device's file system. When this attribute is set, the field will include button control that will display the Windows file dialog, allowing the user to select an image file from the file system.
- **Initial Popup Mode:** This attribute specifies how the image should be initially displayed in the popup screen when the image capture field is clicked by the user. The options are display the captured image in full size or to scale the image display to the size of the popup dialog. This is the initial display mode and the user can switch between the two by clicking the image in the popup dialog.
- **Image Location:** Specifies the location in which the image should be stored once it has been captured.
- **Image Name Prefix:** Specifies a string value to affixed to the beginning of the image file name.

List Tile View

The list tile view field edit type displays an object collection property in a tiled view allowing for add and edit interaction with the collection through the field. For a given object, the properties of that object can be displayed in the list tile view in tiles within that object's record. The values of a given object can be edited directly in this list, and new object instances can also be added. This field edit type also supports scan filter functionality.

The list tile view will make use of an existing screen set defined to display the same object type as is stored in the collection being listed by the field. As a part of a list tile view's definition, a screen set is selected to display the objects within the collection. This screen set must be defined to display the same object type as is found in the collection, and must contain a single detail screen. When the list tile view field is displayed, each object instance within the collection will be displayed within the field in a list. Each tile within the list will be shown in the detail screen from the selected screen set. Two screen sets can be used for read-only display of the objects within the collection. One is used for all rows within the list. The selected row screen set can be defined, with a detail screen containing more fields. This will then be the screen set used to display the selected object.

Similar to this behavior is the ability to add and edit objects for the collection from within the tile view. A transaction for the add and edit behaviors must exist, as must a screen set to be used to display the transactions. When an item is selected in the list, the user can click the add icon button. A new tile will be displayed at the bottom of the list. The screen set in which it will be displayed will be the one defined for the add behavior. For an edit, the user can select an object in the list tile view and click the edit icon button. In this case the currently selected tile will

change to use the edit screen set, displaying the edit transaction. The user can change the values on the screen. They can then either cancel or accept the changes they have made.

As alternatives to this behavior, an action can be specified for both add and edit behaviors. When the action is executed it will dictate the behavior, displaying the add or edit transaction in the wizard screen set just as with any other action.

Filtering can be enabled or disabled for the entire list tile view field. When enabled, the properties of the object type being listed are selected. The user will then only be able to filter the list on these properties. When a property is selected for this purpose, a Tile Filter child definition is added to the list tile view. Users will then only be able to filter the items in the list on one of the selected property values.

Related to the manual filtering, this field edit type also supports scanner filtering. A tile filter can be defined to support scanner filtering. When a barcode value is scanned in it will be compared to the values of that tile filter's property. Only those items that match will be listed. The parent field can then be defined to execute an action when a single item in this list matches the scan filter, and a separate action to execute when no items match.

List Tile View Child Definitions

- **Tile Filters:** The tile filter is a child definition to a detail screen field with an edit type of list tile view, defining the values upon which the items listed in the parent field can be filtered.
- **Sort Properties:** This child definition is a simple list of the object properties by which the list tile view can be sorted at run time on the Agentry Client. When adding a sort property a selection is made from the properties defined in the object type for the collection which the List Tile View field is defined to display.

List Tile View - Collection/Styles Attributes

The list tile view field edit type does not support the following general field attributes:

- Object/Transaction Property
- Format
- Field Style
- Focused Field Style
- Change Focus
- Update Rule
- Special Value

The list tile view data and style attributes set the basic behavior of the view, including how styles can be applied to the list tile view field.

General Settings

- **Collection:** References the object collection property the list tile view is to display. This collection is normally a property of the object definition the parent screen set is defined to display.
- **Include Rule:** References a rule definition expected to return a Boolean value and that is evaluated once for and in the context of each object in the collection displayed by the list view. When an include rule is specified, only those objects for which the rule evaluates to true will be listed in the list tile view.

Styles Settings

- **Header Label:** The style to apply to the list tile view's header label. If no header label is defined this attribute has no affect on the screen.
- **Rows:** The style to apply to all rows on the list tile view.
- **Alternate Rows:** The style to apply to every other row in the list, beginning with the second row.
- **Highlight Rows:** The style to apply to a row for the purposes of drawing attention to that row. This style should always be returned via a rule definition that evaluates the object being listed.
- **Selected Rows:** The style to apply to the row currently selected by the user in the list control.

List Tile View - Settings Attributes

Selection Settings

- **Allow Multi-Row Select:** Specifies if the user can select more than one record in the list at the same time. If multiple items are selected in a list, actions that target the selected object in the list will be executed once for each selected object. The default for selecting multiple objects requires a `Ctrl+Click` combination (mouse input) or a click and drag operation (stylus input) by the user, depending on the device type. The **Enable Single Click** option to this attribute may be set to allow multiple records to be selected with a single click by the user. Deselecting a record requires the user to click it again. This feature is normally most useful on touch screen devices, as it allows non-sequential records in the list to be selected. If this option is enabled the attributes related to editing the objects in the list tile view will be disabled. These objects may still be edited as the selected object in the list tile view, but the action must be executed from a control on the same screen as the list tile view field, rather than from within the list tile view itself.

Action Settings

- **Allow Tile Adds:** This attribute specifies whether or not users will be able to add a new object to the collection being displayed by the list tile view field from within the field. When this option is selected, the **Add Screen Set** and **Add Transaction** attributes must also be set.
- **Allow Tile Edits:** This attribute specifies whether or not users will be able to edit an object within the collection displayed by the list tile view field. When this option is selected, the

Edit Screen Set and **Edit Transaction** attributes must also be set. **Allow Tile Edits** is disabled if the attribute **Allow Multi-Row Select** is enabled for the list tile view field.

- **Allow Single Click Action:** This attribute specifies whether to allow for an action to be executed when a tile is selected in the list with a single click. If this attribute is set to true, all default behaviors of the list tile view field for a single click of a tile are disabled, as are the related attributes within the definition. This includes the following attributes:
 - Allow Tile Edits
 - Allow Multi-Row Select/Enable Single Click Selection
 - All attributes in the section Edit Actions/Tiles

Screen Sets

- **Row:** This attribute specifies the screen set containing the detail screen to display each object contained in the collection being listed in the list tile view field. The screen set selected here will be used for each tile in the list that is not currently selected. The screen set referenced must be defined for the same object type as is contained in the collection being listed by the list tile view. The screen set must also contain a single detail screen used by the same platform as the parent screen of the list tile view.
- **Selected:** This attribute specifies the screen set containing the detail screen to display each selected tile in the list tile view field. The screen set selected here will be used only for a selected tile in the list. The screen set referenced must be defined for the same object type as is contained in the collection being listed by the list tile view. The screen set must also contain a single detail screen used by the same platform as the parent screen of the list tile view.

Add Actions/Tiles

- **Add Screen Set:** This attribute is enabled when the **Allow Tile Adds** attribute is set. Add Screen Set is set to the screen set in which the **Add Transaction** will be displayed within the list tile view field. This screen set is displayed when the user selects clicks the add icon button for the field, allowing the user to add the values for the new object instance.
- **Add Transaction:** This attribute is enabled when the **Allow Tile Adds** attribute is set. **Add Transaction** is set to the transaction that will capture the values from the user for the new object instance to be added to the collection being displayed by the list tile view field. The transaction will be displayed in the list tile view field, with the tile using the screen set selected in **Add Screen Set**.
- **Add Action:** This attribute is enabled when the **Allow Tile Adds** attribute is not set. **Add Action** can be set to the action to execute when the user clicks the add icon button for the list tile view field. This action will be executed, targeting the object selected in **Add Target**. The purpose of the **Add Action** attribute is to execute an action that will add a new object instance to the collection being displayed by the list tile view field.
- **Add Target:** This attribute is enabled when the **Allow Tile Adds** attribute is not set. **Add Target** is set to the object instance that the **Add Action** should target when executed. In

almost all scenarios the **Add Target** should be set to the parent object of the collection being listed by the list tile view field.

- **Add Shortcut Key:** This attribute is set to the shortcut key combination that will allow the user to add an object to the collection being displayed by the list tile view field. The shortcut key will exhibit the same behavior as if the add icon button for the list tile view field were clicked or tapped by the user, meaning either the defined **Add Action** will be executed, or the defined **Add Screen Set** and **Add Transaction** will be displayed in a new tile in the list tile view field.

Edit Actions Tiles

- **Edit Screen Set:** This attribute is enabled when the Allow Tile Edits attribute is set. Edit Screen Set is set to the screen set in which the Edit Transaction will be displayed within the list tile view field for the selected tile. This screen set is displayed when the user selects a tile in the list and clicks the edit icon button for the field, allowing the user to edit the values of the selected object instance.
- **Edit Transaction:** This attribute is enabled when the **Allow Tile Edits** attribute is set. **Edit Transaction** is set to the transaction that will capture the values from the user to modify the object instance selected in the list tile view field. The edit transaction will be displayed in the list tile view field with the tile using the screen set selected in **Edit Screen Set**.
- **Edit Action:** This attribute is enabled when the **Allow Tile Edits** attribute is not set. **Edit Action** can be set to the action to execute when the user clicks the edit icon button for the list tile view field. This action will be executed, targeting the object selected in **Edit Target**. The purpose of the **Edit Action** attribute is to execute an action that will allow the user to edit the selected object instance in the collection being displayed by the list tile view field.
- **Edit Target:** This attribute is enabled when the **Allow Tile Edits** attribute is not set. **Edit Target** is set to the object instance that the **Edit Action** should target when executed. In almost all scenarios the **Edit Target** should be set to the selected object instance of the collection being listed by the list tile view field.
- **Edit Shortcut Key:** This attribute is set to the shortcut key combination that will allow the user to edit the selected object in the list tile view field. The shortcut will exhibit the same behavior as if the edit icon button for the list tile view field were clicked or tapped by the user, meaning either the defined **Edit Action** will be executed, or the defined **Edit Screen Set** and **Edit Transaction** will be displayed in the selected tile of the list tile view field.

Single Click Action - These attributes are enabled only if the attribute **Allow Single Click Action** is set to true.

- **Single Click Action:** This attribute specifies the action to be executed when the user selects a tile in the list.
- **Single Click Target:** This attribute specifies the object to be targeted by the Single Click Action when it is executed.

List Tile View - Filter/Sort Attributes

General Settings

- **Fixed Sort Property:** Specifies the property definition within the object type being listed by which to sort the objects in the list tile. The **Order** option to this attribute is set to specify the sort order, either ascending or descending. For the list tile view it is recommended that this attribute be set, as the list tile view cannot be sorted by the user. If a Fixed Sort Property is not set, the order of the objects in the list will be the order in which they are stored in the collection.
- **Enable Groups:** Enables or disables the group and indexing behavior available in iOS Agentry Clients. When selected, the defined **Fixed Sort Property** is used to group the objects listed in the List Tile View field. Tiles will be sorted based on this selection and grouped by those with the first *x* number of characters (defined in No. Chars option) sorted relative to each other and exclusive to those in other groups. If the **Fixed Sort Property** is a string property, the **No. Chars** option is enabled where the number of characters to group on is defined. For numeric types, grouping is based on the first (highest order) digit. This value should be less than the maximum length of the selected string property. When **Enable Groups** is set to true, the attribute **Allow Filter** is disabled.
- **Show Group Index:** This attribute is only available when Enable Groups is selected. When set to true, this attribute will result in the display of a group index on the right side of the List Tile View field. The user can select one of the items in this list to filter the List Tile View to only the matching items.
- **Allow Sort:** This attribute enables or disables sorting of the List Tile View's tiles on the Agentry Client by the user. When enabled, a button is displayed on the top of the list tile view field that displays a sort dialog when clicked by the user. The user can select a property within the object type being listed and the sort order of either ascending or descending. This attribute is disabled if a Fixed Sort Property is defined.
- **Initial Sort Property:** This attribute allows for the selection of property to sort the list tile view field on during its initial display on the Agentry Client. If a property is selected for initial sorting, the option Order is available to define whether or not the initial sort order should ascending or descending. This attribute is not available unless **Allow Sort** is set to true.
- **Allow Filter:** Specifies whether or not the user can filter the items in the list tile view. A filter icon is displayed for the list tile view field when enabled. The user can click this icon to select filter options. Only those properties for which tile filters have been defined within the list tile view field can be selected by the user in the displayed filter dialog.
- **Shortcut Key:** This attribute specifies the shortcut key combination the user can enter on the Agentry Client to display the filter dialog for the list tile view field. This attribute will have no affect if **Allow Filter** is not set.

Header

- **Header Label:** Specifies the header text for the list tile view. A common use for this header label is the total number of objects displayed in the list vs. the total number of

objects in the collection, which may be different when a filter is enabled. The format strings used for this purpose are %DisplayedCount and %TotalCount.

List Tile View - Scanner Attributes

The scanner attributes for a list tile view affect only those list tile view fields defined for a detail screen that is used by a scanner platform within the screen set and only when the screen set is displayed on a client device with a barcode scanner. At least one tile filter must be defined within the list tile view to support scan filtering.

Single Match

- **Use Edit Row:** This attribute specifies whether or not to use the defined edit behavior for the single object that matches the scan filter settings. When set, the selected object will be edited via either the defined **Edit Action**, or the defined **Edit Screen Set** and **Edit Transaction** in the List Tile View Settings attributes. If this attribute is set, the Single Match Action attribute will be disabled.
- **Single Match Action:** Specifies what action is executed when a scanned barcode value uniquely matches an object in the list tile view. The target of the action will always be the object instance found to match. This attribute will be disabled if the Use Edit Row attribute is set.

No Match

- **Use Add Row:** This attribute specifies whether or not to use the defined add behavior of the list tile view field. When set, defined Add Action will be executed, or the defined Add Screen Set and Add Transaction in the List Tile View Settings attributes will be displayed in a new tile added to the list. If this attribute is set, the No Match Action attribute will be disabled.
- **No Match Action:** Specifies what action is executed when the scan filter criteria does not match any records in the list. The target of the action is the parent object to the collection property displayed by the list tile view. This attribute will be disabled if the Use Add Row attribute is set.

Label

- **Label Types:** Specifies what barcode types are accepted by the Agentry Client. If no **Label Type** is specified, all types supported by the client device's scanner will be supported. To restrict the label types, enter the name of each label type to support, separated by a comma. Barcodes not listed will not be processed by the Agentry Client.
- **Minimum Value:** The minimum number of characters accepted by the Agentry Client from the device scanner. If the value scanned in contains fewer characters, it will be ignored.
- **Maximum Value:** The maximum number of characters to be accepted by the Agentry Client from the device scanner. If the value scanned in contains more characters, it will be ignored.

- **Shortcut Key:** This attribute allows a shortcut key combination to be defined to activate the device's barcode scanner. This should be set to a key combination not already defined as a shortcut for any other items on the current screen or any system-level shortcut keys.

Tile Filter

The tile filter is a child definition to a detail screen field with an edit type of list tile view. A tile filter defines the property within the object type being listed upon which the items listed in the parent field can be filtered. This includes both manual, user defined filters as well as barcode scan filters.

A tile filter targets an object property within the object type being listed by the parent list tile view field. For this property, the tile filter then defines whether or not the user can filter on this property manually, and whether or not scan filtering is enabled for this value within the list tile view.

Tile Filter Attributes

- **Object Property:** This attributes contains the target path to the object property for the tile filter.
- **Allow Filter:** This attribute specifies whether or not the user can select this property from the list of object properties displayed in the filter dialog on the Agentry Client.
- **Scan Filter:** This attribute specifies whether or not the value scanned in by the client device's barcode scanner should be compared to the value of the property targeted by the tile filter. It is considered a best practice to set this attribute to true for only one tile filter within the same list tile view field, but this is not a requirement.

List Selection

The list selection field edit type displays a drop down list of values, the source of which may be an object collection, data table, or complex table. This list is treated as a temporary data table created at run time. Part of the definition of this edit type is to specify the values to be treated as the code and value fields for each record. Definable behaviors include whether to display the code, value, or both for each item listed. The code field is always the value returned from the selected item in the list.

A field with this edit type is displayed as a drop down list on the detail screen. If the number of records displayed in the list is large, a popup dialog will displayed when the user selects the field.

To use this edit type, either an object collection property or complex table is selected as the source for the items listed. Within this selected source two data members (object properties or complex table fields) are selected as the code and value for the records in the temporary data table.

Another aspect of this edit types behavior is the option to define an include rule. If used, this rule will be evaluated for each object or record in the defined source and only those items for which the rule returns true will be listed in the field. Note that this rule evaluation should be

made as efficient as is possible when working with complex tables with large numbers of records.

List Selection Attributes

The following attributes are specific to the list selection field edit type and are in addition to the common detail screen field attributes.

- **Source:** This attribute specifies the source object collection property or complex table for the field. The collection or complex table may be returned via a rule, or it may be selected from the target browser. Within the target browser, options exist for selecting object instances or a range of complex table records via a rule. Note that this is separate from using an include rule, which is another attribute to the field type. In most situations simply the object collection or complex table is selected here. It may be desirable for complex tables to return a range of records based on a table index. In this case the search value is provided in the target browser, which may come from an object property or a rule. Specifying a search value for the complex table to reduce the number of records for the field can significantly reduce the number of evaluations needed for the include rule, if one is used.
- **Include:** The Include attribute can reference a rule definition that will filter the records listed to the user. This rule is evaluated in the context of each data instance (object or complex table record) returned by the Source attribute and is expected to return a Boolean value. Only those data instances for which this rule returns true will be listed in the field. If no Include rule is selected, all instances of the selected source will be listed.
- **Key:** This attribute specifies the data definition within the **Source** instances to be used as the code (also known as the Key) field for each record in the temporary data table. This will either be an object property or complex table field, depending on the selected source.
- **Value:** This attribute specifies the data definition within the Source instances to be used as the value field for each record in the temporary data table. This will be either an object property or complex table field, depending on the selected source.
- **Sort By:** This attribute allows you to sort the values listed in the drop down list by one of several options: Code, which is the code field in each table record; Value, which is the value field in each record; Displayed Text, which is the text displayed for each record in the field; and Order in Data Table, which is the order in which the records are listed in the data table itself.
- **Sort Order:** This attribute specifies whether the records displayed are sorted in ascending or descending order. This is a string sort.
- **Display Type:** This attribute specifies which fields from the data table records should be displayed in the drop down list. The options are: Code, meaning the code field in each table record; Value, which displays the value field from each record; Code - Value, which displays both fields from each record, separating them with a hyphen; and Format Text, which allows you to specify format strings to format the values displayed in the list for each record.
- **Format Text:** If the **Display Type** attribute is set to "Format Text" this attribute will be enabled. Format strings can then be entered in this attribute to format each record from the

data table. The valid format strings for this field are %code, %value and %position. This last will display the position number of each record as stored in the data table. This last option is used mostly for testing purposes and is generally not found in the production version of an application. This attribute can also contain any other printable characters, excluding tabs and carriage returns, to format the display of the table's records.

- **Editable - Allow User-Entered Values:** This attribute specifies whether or not the user can manually enter values not found in the source of the list. When set users will be able to either select an item from the list or manually enter a text value in the field. When not set the users will be required to select from the items in the list. Note that if the field is set to read-only, this attribute should still be set if the field has an update rule defined, and if that rule can return a value not found in the data source for this field.
- **Define separate display type for popup dialog:** This attribute allows you to display the records from the data table differently in the popup dialog vs. the drop down list for the field. If set to false, the display and format attributes listed above will also affect the popup dialog. If set to true, the attributes listed next will provide separate display behaviors for the popup dialog.
- **Display Type (Popup Dialog):** The options for this attribute are the same as the Display Type options listed previously. The option selected here will impact the appearance and behavior of the popup dialog displayed for larger data tables.
- **Format Text (Popup Dialog):** The format strings for this attribute are the same as the Format Text options listed previously. The format text entered here will impact the appearance and behavior of the popup dialog displayed for large data tables.

List View

The list view field edit type displays an object collection property in a list control on a detail screen. This list contains all of the same definable behaviors as a list screen, but is contained within a detail screen. Multiple list views may be displayed on a single detail screen. This is the default edit type for a field targeting an object collection property.

When a field is defined with a list view edit type, that field will have column child definitions. A list view field can be defined on a wizard detail screen for a transaction or fetch. However, the attributes for the double-click actions will be disabled, as actions may not be executed from a wizard.

List View Child Definitions

- **Column Definition:** A list screen column defines what object property is displayed for each record in a list control and how it is formatted on the screen.

List View Data/Styles Attributes

The list view data and style attributes set the basic behavior of the view, including how styles can be applied to the list view field.

List Data

- **Collection:** References the object collection property the list view is to display. This collection is normally a property of the object definition the parent screen set is defined to display.
- **Include Rule:** References a Rule definition expected to return a Boolean value and that is evaluated once for and in the context of each object in the collection displayed by the list view. When an include rule is specified, only those objects for which the rule evaluates to true will be listed in the list view.
- **Icons Image:** References an image definition containing an image list to be displayed in a column on the list view. This is an image list with the positions of the images in each list then referenced by the child column definition's Icon attribute. Note that columns may also reference image definitions to use for this same purpose, though they may not be image lists.

List Styles

- **Header Label:** The style to apply to the list view's header label. If no header label is defined this attribute has no affect on the screen.
- **Column Labels:** The style to apply to the column labels on the screen's list control.
- **Rows:** The style to apply to all rows on the list view. The Hyperlinks optional style will override the Rows style for cells with hyperlinks.
- **Alternate Rows:** The style to apply to every other row in the list, beginning with the second row. The Hyperlinks optional style will override the Alternate Rows style for every other row, specifically cells containing hyperlinks within the row.
- **Highlight Rows:** The style to apply to a row for the purposes of drawing attention to that row. This style should always be returned via a rule definition that evaluates the object being listed. The optional Hyperlinks style will be applied to the highlighted row's cells containing a hyperlink.
- **Selected Rows:** The style to apply to the row currently selected by the user in the list control. The optional Hyperlink style will be applied to any cells within the selected row containing a hyperlink.
- **Detail Pane:** The style to apply to both the foreground (text) and background of the list view's detail pane. If no detail pane is defined this attribute has no affect on the screen.

List View Actions/Sorting Attributes

The list view actions and sorting attributes control how the user interacts with the list view, including double-clicking on or off an item in the list and behaviors related to sorting and reordering the columns. Note that the double-click attributes will be disabled for list view fields defined on wizard detail screens.

Double-Click Actions

- **Double-Click On Item - Action:** Specifies the action to execute when the user double-clicks a list view record.

- **Double-Click On Item - Target:** Specifies the target of the **Double-Click On Item - Action**. A target must always be specified for the action and is typically the selected object in the list view.
- **Double-Click Off Item - Action:** Specifies an action to be executed when the user double-clicks the list view without clicking on an item. This is most commonly used to execute an action that instantiates an add transaction for the object type being listed.
- **Double-Click Off Item - Target:** Specifies the target of the **Double-Click off Item - Action**. A target must always be specified for the action. Typically the target is the parent object of the object collection property displayed by the list view.

Sorting and Selection

- **Fixed Sort Property:** Specifies the property definition within the object type being listed by which to sort the items in the list. Selecting a property here prevents the user from resorting the list on any other column. The **Order** option to this attribute is set to specify the sort order, either ascending or descending.
- **Allow Sort:** Specifies if the user can sort the list by clicking on a column header. This is enabled by default, and is disabled if a **Fixed Sort Property** is set.
- **Initial Sort Column:** Specifies a column definition by which the list will be sorted upon initial display of the list view. This attribute requires that a column definition exist before it can be set. The **Order** option to this attribute is set to specify the sort order, either ascending or descending. If the list view allows the list to be sorted (**Allow Sort** is true) the list will be displayed sorted in the order of the last sort action. If a **Fixed Sort Property** is set, this attribute is disabled.
- **Allow Multi-Row Select:** Specifies if the user can select more than one record in the list at the same time. If multiple items are selected in a list, actions that target the selected object in the list will be executed once for each selected object. The default for selecting multiple objects requires a **Ctrl+Click** combination (mouse input) or a click and drag operation (stylus input) by the user, depending on the device type. The **Enable Single Click** option to this attribute may be set to allow multiple records to be selected with a single click by the user. Deselecting a record requires the user to click it again. This feature is normally most useful on touch screen devices using a stylus, as it allows non-sequential records in the list to be selected.
- **Allow Reorder:** Specifies whether or not the user can reorder the columns displayed in the list view by dragging and dropping the column headers. This is enabled by default.
- **Allow Filter:** Specifies whether or not the user can filter the items in the list. A filter icon is displayed at the bottom of the list view field when enabled. The user can click this icon to select filter options. Individual column definitions may be defined to prohibit filtering on those columns.

List View Header/Detail Pane Attributes

Using these attributes, a header label or a detail pane may be added to the list view field. Header label and Detail pane attributes are set to display additional information about the list as a whole or about the currently selected item in the list. The Header Label is a static line of text displayed above the list view, within the area given to the field. This text may be static, set

via certain available format strings, or set via a rule. A rule referenced for this purpose is expected to return a string value and is evaluated in the context of the object displayed by the parent screen set.

The Detail Pane is redrawn each time a new object is selected in the list and almost always contains either format strings or is set via a rule's return value. Rules are evaluated in the context of the selected object in the list and are expected to return a string value. The detail pane drawn on the screen is a multi-line, read-only text box that may be scrolled horizontally or vertically if needed. The detail pane is drawn within the area given to the list view field and will reduce the amount of space for the list items.

Header

- **Header Label:** Specifies the header text for the list view. A common use for this header label is the total number of objects displayed in the list vs. the total number of objects in the collection, which may be different when a filter is enabled. The format strings used for this purpose are `%DisplayedCount` and `%TotalCount`.

Detail Pane

- **Detail Pane:** When true, a text box on the list view. The detail pane is updated each time the user changes their selection in the list view.
- **Position:** Controls where the detail pane is displayed on the screen in relation to the list control. This may be below the list or to its right.
- **Size:** Sets the pixel size of the detail pane within the list view field. The default is 50. If the **Position** is "Bottom" the detail pane will span the width of the space given to the field and the **Size** will set its height. If the **Position** is "Right" the detail pane will span the height of the space given to the field and the **Size** will set its width.
- **Word Wrap:** When enabled, lines of text longer than the width of the detail pane will be wrapped to the next line. When disabled, text will continue off the detail pane. The user will need to scroll the detail pane to view the text.
- **Format:** Sets the values displayed in the detail pane. This pane can be set to a combination of static text and format strings, which take the form `%propertyName`. The `propertyName` is the name of a property defined within the selected object and will be updated with the value of that property each time a different object is selected. It may also be set to the return value of a rule, which is evaluated in the context of the selected object instance and is expected to return a string.

List View Scanner Attributes

The scanner attributes for a list view affect only those list view fields defined for a detail screen that is used by a scanner platform only when the detail screen is displayed on a client device with a barcode scanner. At least one column definition within the list view must be defined to support scan filtering.

A scanned value will be compared to the column(s) defined for scan filtering and only those matching this value will then be displayed. Actions may be executed automatically when a single record matches the scan filter, or when no records match.

- **Show Button:** This attribute specifies whether or not a button is displayed to activate the device's barcode scanner.
- **Single Match Action:** Specifies what action is executed when a scanned barcode value matches one of the records displayed in the list view. The target of the action will always be the object instance found to match.
- **No Match Action:** Specifies what action is executed when the scan filter criteria does not match any records in the list. The target of the action is the parent object to the collection property displayed by the list view.
- **Label Types:** Specifies what barcode types are accepted by the Agentry Client. If no Label Type is specified, all types supported by the client device's scanner will be supported. To restrict the label types, enter the name of each label type to support, separated by a comma.
- **Minimum Value:** The minimum number of characters accepted by the Agentry Client from the device scanner. If the value scanned in contains fewer characters, it will be ignored.
- **Maximum Value:** The maximum number of characters to be accepted by the Agentry Client from the device scanner. If the value scanned in contains more characters, it will be ignored.
- **Shortcut Key:** This attribute can define a shortcut key combination to activate the device's barcode scanner. This shortcut cannot be the same as any other shortcut defined for the current screen or any system level shortcuts configured on the client device.

List View Column

A column definition defines what object property is displayed in a list control column. The column definition also controls behaviors such as formatting, sorting the list on the column, whether or not the column can be resized or moved, and whether or not the list can be filtered on the column. Columns may also be defined to execute an action via hyperlink control.

In addition to or in place of a property value, a column may also display an image definition as an icon, which can be different for each record based on a rule definition.

Column Attributes

- **Object Property:** Specifies the property to display in the column on the list view. Set this to None, to display either a value derived from a format string or only an icon image. Selecting both an Object Property and specifying an icon image will display both in the column.
- **Name:** Internal name for the column definition. This value must be unique among all columns definitions in the list view.
- **Label:** Specifies the label for the column header. This text is displayed at the top of the column on the Agentry Client to identify the contents of the column.
- **Enable Rule:** References a rule definition evaluated in the context of the object displayed by the screen set and expected to return a Boolean value. When the rule returns true, the

column is enabled and displayed on the Client. When it returns false, the column is disabled and not displayed.

- **Format:** Can contain a format string to display one or more property values from the object type being displayed by the list in a different format than the default for the property's data type. This text can also be set via a rule definition, where the expected return value is a string and is evaluated in the context of the object instance for the record in the list. To set the format attribute set the **Object Property** attribute must be set to None.
- **Icon Image:** References an Image definition within the application to specify an icon for the column. The image name can also be returned using a rule definition to dynamically determine the image to display for each record. This rule is evaluated in the context of the object instance for the record and is expected to return the name of an image definition as a string. Note that not using a rule for this attribute will display the same image for all records in the list
- **Column Width:** Specifies the initial size of the column on the client. The user can resize the columns if the list view definition has not disabled this behavior. If the user changes the width of a column, the new width is saved in the registry on the client device and will override the Column Width attribute.
- **List Filter:** Specifies if the column should be included in those listed in the filter dialog for the list. This attribute is ignored in filtering has been disabled for the list view.
- **Scanner Filter:** Enables scan filtering functionality for the column. When this attribute is enabled, the value scanned in by the device will be compared to the values of the column to create a filter. Multiple columns can be defined for this behavior. However, the values in the columns should be mutually exclusive. The order of the columns evaluated against the scanned value is undefined. This attribute is only supported for screens used by a scanner platform and displayed on a scanner-enabled device.
- **Hyperlink:** Specifying a hyperlink action enables each cell within the column to execute an action when the user single or double clicks on the hyperlink drawn in that column. The text of the hyperlink will be the value the column is defined to display. This functionality can include columns with images. Hyperlink contains two attributes:
 - **Hyperlink Action:** Specifies the action that will be executed when a user single-clicks a column in a populated row in the list.
 - **Hyperlink Target:** Specifies the target of the Hyperlink Action.

Password Validation

The password validation edit type requires users to enter their password on a detail screen. The value entered is validated against the password stored for the Agentry Client for the current user. The characters entered in this field are replaced with asterisks. This field edit type is used primarily with transaction authentication functionality.

The value entered in this field is validated against the user's password when the wizard or authentication screen set is advanced. If the value entered is not a valid password a message will be displayed. This message may be the default message provided by the Agentry Client, or it may be defined as a part of the screen field using the **Message** attribute.

Password Validation Attributes

The following attributes are specific to the password validation field edit type. These are in addition to the common field attributes:

- **Password Failure Message:** This attribute specifies the message to display to the user if the password entered in the field is invalid. This may be the default message, which is displayed when Auto is selected, or it may be a message entered in the text box for this attribute field.

Tile Edit

The tile edit field type displays object properties in a tiled view allowing for add and edit interaction without starting a wizard screen. For a given object, the properties of that object can be displayed in the tile edit view in tiles within that object instance. The layout of the tile edit is defined in a separate screen set and detail screen, used by the tile edit field. The values of a given object can be edited directly, and new object instances can also be added.

Prior to defining a field with this edit type, the screen set and the transaction it is to use must be defined. Both are required information when defining a new tile edit field. The screen set must be defined to display the transaction to capture the data. The screen set must contain a single detail screen displaying the properties from the transaction.

At run time this field type is displayed within its own detail screen. Within the field is then the single detail screen from the separate screen set. The fields of this detail screen are displayed within the tile edit field in the same manner in which they are laid out in the detail screen. The user can edit any fields defined in the separate detail screen that are not read-only. Within the tile edit field, if the detail screen displayed is larger than the space given to the field, a vertical scroll bar is displayed to allow the user to scroll up and down to display fields not immediately shown.

The tile edit field type does not support the following common field attributes:

- Object Property
- Read-only
- Format
- Change Focus
- Update Rule
- Special Value

Tile Edit Attributes

The following attributes are specific to the tile edit field type. These are in addition to the common field attributes:

- **Tile Edit Screen Set:** This attribute specifies the screen set to display for edits. This screen set should be defined to display the transaction definition specified in the Tile Edit Transaction attribute.

- **Tile Edit Transaction:** This attribute specifies the edit transaction instantiated to capture data entered by the user in the Tile Edit Screen Set. An instance of this transaction is created, applied, and saved as a pending transaction when the user enters data changes.
- **Tile Target:** This attribute specifies the object instance that is targeted by the transaction.
- **Modify Row Height By:** This attribute allows for all rows displayed on the screen within the Tile Edit field to be modified by the value set in this attribute.
- **Hide Buttons:** This attribute will hide the OK and Cancel buttons displayed when the tile is being edited. These are displayed by default. When hidden, values entered by the user are automatically applied, as is the define transaction, when the Tile Edit field no longer has the input focus.
- **In Progress Edit:** This attribute will enable the In Progress Edit style to be applied to the field when it is currently being edited and the changes it contains have not been applied. This is a visual indicator to the user that the Tile Edit field currently has the focus and is actively being edited.

Tile Display

The tile display edit type displays an object instance in a tiled view. The layout and appearance of the values is defined in a separate screen set and its detail screens. This separate screen set is used by the tile display field, with its detail screens displayed within the tile display field as a tab control.

Prior to defining a field with an edit type of tile display, the separate screen set it is to display must be defined. This screen set must be defined to display the object type desired for display in the tile display field. The separate screen set can contain a single detail screen. The fields of this detail screen display the property values of the selected object type.

When the detail screen containing the tile display field is displayed on the client, the separate screen set and its detail screen are displayed within the viewable area of the tile display field. These values are read-only.

The tile display edit type does not support the following general field attributes:

- Object Property
- Read-only
- Format
- Change Focus
- Update Rule
- Special Value

Tile Display Attributes

The following attributes are specific to the tile display edit type. These are in addition to the common field attributes:

- **Tile Display Screen Set:** This attribute specifies the screen set to display the object instance within the tile display field. This screen set can contain one detail screen. The

fields of this detail screen are displayed within the tile display field. The screen set must be defined to display the object definition specified in the Tile Target attribute.

- **Tile Target:** This attribute specifies the object instance targeted by the tile display field. This object instance must be of the type the Tile Display Screen Set is defined to display.
- **Modify Row Height By:** This attribute specifies the rows within the screen being displayed by the Tile Display field be modified by the value set in this attribute.
- **Display Single Screen:** This attribute forces the Tile Display field to display only a single screen from the selected screen set. Otherwise each screen in the screen set is displayed within the Tile Display, with a tab control displayed for each screen.

Detail Screen Fields With Implicit Edit Types

In addition to those detail screen field edit types already addressed, there is a small handful of edit types which are implicitly set based on the data type of the property the field is defined to display. These implicit field edit types do not contain any edit type-specific attributes. The general field behaviors, e.g., position, size, read-only, etc., are defined just as any field would be. The edit type-specific behaviors are typically taken driven by the definition of the property being displayed.

These edit types cannot be selected from the Edit Type attribute for the field definition. Instead, when a field is defined to display one of the property data types with which the field edit type corresponds, the field definition's Edit Type should be left set to --Default--. As an example, when displaying a signature capture property type, the field to display this property will not contain a corresponding Signature Capture edit type. Rather, it is left set to an Edit Type of --Default--. At run time, the field displayed on the client will be a signature capture field, and the behavior of the field is driven by the definition of the signature property.

Signature

The signature field edit type allows for the entry of a signature on a client's screen that is stored as a bitmap image. This is an implicit edit type in that it cannot be selected when defining a field definition. Any detail screen field with an edit type of "Default" and targeting a property with a data type signature will be a signature field. This field edit type has no additional attributes beyond those of the common field attributes. Much of the behavior of this field is dictated by the signature property it targets.

Action

An action defines navigation and user interaction for the Agentry Client. Actions are composed of a series of action steps of varying types. An action is defined to allow the user to interact with the application in some way.

The action defines the object its steps will act upon, any may also define whether or not users will be permitted to cancel the action once it has been executed, and also an optional separate action to execute if the action is cancelled.

The behavior of the action is dictated primarily by its child action step definitions. There are different types of action steps for different types of Agentry Client behaviors. Each action step defines a specific task to be performed on the Agentry Client.

Actions can be referenced by several different components of the user interface, such as buttons, list screens, and others. Whenever an action is executed it will be passed an object instance. This object instance is determined by the user interface component executing the action. The action must be defined for an object type. This object type for the action and the type of object passed to the action on the Agentry Client must be the same. The exception this is when the action is not defined for any object. Such actions are limited in use and normally pertain to performing transmits between the Agentry Client and Agentry Server, or actions that close screen sets but do not open others.

Action Child Definitions

Action Step: An action step defines a single task within an action that is a part of the overall action execution on the Agentry Client.

Action Attributes

- **Name:** This is the unique internal name for the action within the application project. This value must be unique among all actions defined within the same module.
- **Display Name:** This attribute contains the name displayed for the action on the Agentry Client.
- **Group:** This attribute specifies the group into which the action will be organized within the application project. This attribute has no impact on the action's behavior at run time.
- **For Object:** This attribute specifies the object for which the action is defined. An instance of this object must be passed to the action by the Agentry Client user interface component executing the action. Therefore, both this attribute and UI component must have the same type of object defined or in scope when the action is executed.
- **Enable Rule:** This attribute references a rule definition called in the context of the object currently in scope on the user interface and is expected to return a Boolean value. When the rule returns true the action will be enabled and can be executed. When the rule returns false the action will be disabled and cannot be executed. Any buttons defined to execute a disabled action will be displayed as disabled controls.
- **Disable Cancel:** This attribute specifies whether or not users can cancel an action once its execution begins. This setting primarily affects the behavior of screen sets defined to display transactions or fetches and are displayed by the action. When Disable Cancel is set to true, screen sets will not contain a cancel button, preventing the user from canceling the action. When set to false (default) the wizards will contain a cancel button.
- **Cancel Action:** This attribute references another action within the same module to be executed when the parent action is canceled. The cancel action will be executed in the same context as the action that was executed first and then canceled by the user. The action selected here must exist prior to making a selection.

Action Step

An action step defines a single task within an action that is a part of the overall action execution on the Agentry Client. There are multiple action step types. Each type of step is defined for a different type of task. These can include navigation, transaction instantiation and display, transmit initiation, and other behaviors. Each action step type contains its own type-specific attributes.

The action step definition encapsulates a single task to be performed within the action as a whole. A given action can contain one or more action steps. Much of the client-side functionality and behavior that may be defined for a mobile application is exposed in the action steps.

Action Step Types

Following are the different types of action steps that may be defined:

- **Apply:** The apply action step type applies all transactions instantiated and completed before it in the same action.
- **Exit Application:** The exit application client action step will close the Agentry Client application when executed.
- **External Field Command:** The External Field Command action step issues a command to an ActiveX control when executed.
- **List Selection:** The List Selection action step type selects the specified row or item in the selected screen set and screen.
- **Message:** The message action step type displays a message screen on the Agentry Client to the user that can contain one or two buttons.
- **Navigation:** The navigation action step type displays an object screen set on the Agentry Client.
- **Open URL:** The Open URL action step type defines a URL to be opened by the client device's web browser.
- **Print Report:** The print report action step type will print the defined report definition on a printer connected to the client device.
- **Save Tile Transactions:**
- **SubAction:** The SubAction action step type executes an action definition from within another action.
- **Transaction:** The transaction action step type instantiates a transaction on the Agentry Client and defines what screen set to display the transaction instance in.
- **Transmit:** The transmit action step type initiates communications between the Agentry Client and Agentry Server.
- **Windows Command:** The Windows command action step type executes a command on the client device.

Action Step Type: Apply

The apply action step type applies all transactions instantiated and completed before it in the same action. An apply step is required in any action containing one or more transaction steps in order for those transactions to affect their target objects and to be saved on the Client.

The apply step definition itself contains no attributes other than a name. However, it is an important part of transaction behavior within the Agentry Client. The absence of an apply step within an action that also includes a transaction step will result in the transaction not being save or applied on the Agentry Client.

The intended purpose of the separate apply step to apply and save a transaction is to allow for actions continuing multiple transaction steps followed by a single apply step. This allows for the requirement that multiple transactions be instantiated and completed by a user within a single action, and to not save any data until all transactions have been finished.

Apply Step Attributes

This action step type has only a **Name** attribute, which must be unique among all steps within the same parent action.

Action Step Type: Exit Application

The exit application client action step will close the Agentry Client application when executed. When this step is performed within an action it will result in the same behavior as if the Exit menu item is selected in the File menu of the Client. A step of this type should only be defined as the last step to be executed within an action, as no other action steps that follow it will be executed.

The primary purpose of this action step type is to support the true and clean shutdown of the Agentry Client when running on client devices that do not support this behavior easily. Many client devices and the shells they run will no truly exit an application when the user clicks the title bar close button. Rather, the application is simply hidden from view. It remains running on the client device. Additionally, the application does not exhibit any behaviors defined to occur when the application exits, such as check for, and notifying the user of any pending transactions.

To support a cleaner shut down, the users should always be instructed to use either the Exit menu item in the client's File menu, or to execute an action defined with an Exit Application action step type.

Exit Application Step Type Attributes

This action step type has only a **Name** attribute, which must be unique among all steps within the same parent action.

Action Step Type: External Field Command

The External Field Command action step issues a command to an ActiveX control when executed. It references the External Field - ActiveX Control field to specify the control to which the command is to be issued. The action step passes the value of the defined command string to the ActiveX control, which is then responsible for receiving and processing the string command accordingly.

The defined command string within this action step type is passed by the Agentry Client to the ActiveX control through the expected method `AgentryExecuteCommand`. This method should be implemented to process the provided command string in the manner deemed appropriate for that control.

External Field Command Step Attributes

- **Step Name:** This attribute contains the unique internal name of the action step definition. This must be unique among all steps within the same parent action.
- **Screen Set:** This attribute specifies the screen set containing the detail screen within which the External Field - ActiveX Control field is defined. Valid selections for this attribute include any screen set defined to display an object definition. Screen sets for transactions and fetches are not valid.
- **Screen:** This attribute specifies the detail screen containing the External Field - ActiveX Control field.
- **External Control:** The External Field - ActiveX Control detail screen field that references the ActiveX control to which the command string is to be issued.
- **Command:** The string to be passed to the ActiveX control's `AgentryExecuteCommand` method. This attribute value can be entered into the attribute field directly, or can be set to the return from a rule definition. A rule referenced by this attribute is evaluated in a string context and in the context of the action to which the action step is being added and the object for which that action is defined.

Action Step Type: List Selection

The List Selection action step type selects the specified row or item in the selected screen set and screen. The specific list control on the screen must also be specified if more than one type of list field is defined for that screen. The action allows for the specification of record to select by one of several options, as described in the Select Rows attribute of the field definition.

List Selection Step Attributes

- **Step Name:** This attribute contains the unique internal name of the action step definition. This must be unique among all steps within the same parent action.
- **Screen Set:** This attribute specifies the screen set containing the screen and list in which a selection is to be made.

- **Screen:** This attribute specifies the screen containing the list in which a selection is to be made.
- **List Control:** This attribute specifies the list on the selected detail screen in which a selection is to be made.
- **Select Rows:** This attribute specifies how the item in the list is to be selected. Options for this attribute include:
 - **By Position:** Selecting this option enables the Position attribute, where the position number of the item to be selected can be specified. This is a numeric value that must be one or greater and indicates the item to select from either the top or bottom of the list.
 - **By Rule:** This option specifies a rule is to be used to determine the item to be selected. The object being listed sets the context for the rule being evaluated, with either a true or false value returned by the rule. The first item for which the rule returns true will be the one selected in the list.
 - **First Row:** The first row in the list based on it's current sort order.
 - **Last Row:** The last row in the list based on it's current sort order.
 - **Next Row:** The row immediately following the row currently selected in the list.
 - **None (Clear selection):** This clears the selection state of any items that may be currently selected in the list.
 - **Previous Row:** The row immediately preceding the row currently selected in the list.

Action Step Type: Message

The message action step type displays a message screen on the Agentry Client. This screen can contain a defined title, message text, and either an OK or an OK and Cancel button. When a Cancel button is clicked in a message screen the parent action of the step is canceled. No subsequent steps within the action will be executed.

This step type can provide the user with the ability to cancel an action based on some decision. This step type is commonly used within actions that will delete an object instance on the client. A message step with two buttons can be defined to confirm the delete of the object prior to executing the transaction step that will delete it.

When a message step is displayed with only a single button, the user will not be able to cancel the action within the message displayed. Rather, the single button is displayed for the user to confirm they have read the message. Once clicked, the action will continue execution with its next defined action step.

Message Step Attributes

- **Step Name:** This attribute contains the unique name for the action step. This value must be unique among all steps within the same parent action.
- **Caption:** This attribute contains the text to display in the title bar of the message dialog displayed by the Message Step.
- **Message Text:** This attribute contains the text to display in the main portion of the message dialog displayed by the Message Step. Format strings may be used within this

text, or the entire message may be built and returned by a rule. A rule definition referenced here is evaluated in the context of the object passed to the step by the action. The rule is expected to return a string value.

- **OK Label:** This attribute contains the text to label the OK button in the message dialog. Regardless of the label, clicking this button will always confirm the message, or be considered a positive response to the message, continuing execution of the action.
- **Cancel Label:** This attribute can enable or disable the cancel button behavior in the message log. When disabled, no cancel button is displayed. When enabled, the cancel button will be displayed and this attribute also then contains the label for that button. Regardless of the label text, clicking this button will always be considered a negative response and cancel the parent action's execution.

Action Step Type: Navigation

The navigation action step type displays an object screen set on the Agentry Client. It includes optional definable behaviors to specify the screen and control on the screen to which the initial focus is set. It may also be defined to close the previous screen set displayed. Screen sets may be defined to display an object, transaction, or fetch. A navigation action step is defined to display only those screen sets defined to display an object.

When selecting a specific screen within a screen set to be the first one displayed, you will typically select any of the screens not at position one within the screen set. The screen at position one within the screen is displayed first by default. If the navigation step displays a screen set with multiple platforms, and the selected screen definition is not used by one or more of the platforms, the screen at position one within the screen set is displayed first on those device platforms.

Similar behavior is exhibited when a specific field is selected within a detail screen to have the initial focus. If the field does not exist on a screen for a given platform, the default focus field, as defined within the screen, will contain the initial focus.

If the initial screen defined to be displayed is a list screen, the optional behavior of selecting one or more rows by default within that list can be defined. There are several options for selecting the rows, including: by position within the list; conditionally based on a rule; or to not select any record within the list.

The definable behavior of closing the previous screen set, or closing all open screen sets, can be used when navigating from one screen set to the next. However, it is recommended that closing the previous screen behavior not be defined when navigating from a module main screen set, as closing the main screen set is considered undesirable user interface behavior except in rare circumstances. Closing all open screen sets will never close the current module's main screen set.

Closing the previous screen set or all non-main screen sets may also be defined in a navigation step that does not display a new screen set. This definition option results in the user being returned to the previously displayed screen set, i.e. the one from which they navigated to the current screen set, or the module main screen set. These behaviors are supported to ensure that

a screen set and its screens are truly destroyed when the user wishes to close them. Certain client devices and their shells do not close a screen once opened. Rather, when a user clicks the close button (or sometimes an OK button) displayed in a title bar, the screen itself is simply hidden, but still remains in the background. The navigation step will close a screen set by destroying the screen object in memory. Closing all open screen sets other than the module main screen set provides an easy means of returning the user to the module main screen set if they are multiple levels deep into the application's screen flow.

Navigation Step Attributes

- **Step Name:** Contains the unique internal name of the action step definition. This must be unique among all steps within the same parent action.
- **Screen Set:** Specifies the object screen to be displayed by the navigation step. This screen must be defined and exist within the module prior to defining the navigation step. This may be set to "Do No Display Screen Set," which will also set the **Close Screen** attribute to "Close the screen you are leaving when this navigation step runs."
- **Close Screen:** Specifies that the screen set currently displayed when the action is executed be closed, or alternately that all screen sets other than the module main screen set be closed. The default setting for this attribute is to not close any screen sets. Valid options for this attribute include:
 - *None:* Do not close any screen sets. Display the screen set defined in the **Screen Set** attribute.
 - *Close the screen you are leaving when this navigation step runs:* The currently displayed screen set will be closed when the action step is executed. The screen set defined in the **Screen Set** attribute, if any, will then be displayed.
 - *Close all screens except main when this navigation step runs:* All open screen sets except the module main screen set will be closed. The screen set defined in the **Screen Set** attribute, if any, will then be displayed. In an application with multiple modules, the main screen set for all modules except the current module will be closed.
- **Screen:** Specifies the screen within the screen set to display first. By default, the first screen displayed is the screen at position one within the screen set. If a detail screen is selected, the Initial Focus attribute field is enabled. If a list screen is selected, the Select Rows attribute field is enabled.
- **Initial Focus:** Available only when the Screen attribute is set to a detail screen. The Initial Focus can specify a field on the selected detail screen that will have the initial focus when the navigation step displays the screen set. By default, the field at position one within the screen will have the initial focus.
- **Select Rows:** Available only when the Screen attribute is set to a list screen. The following are the options for this attribute, each of which specifies which row or rows within the list screen should be selected automatically when the list screen is displayed:
 - *All Rows:* This selection will result in all rows in the list screen being selected initially. This selection is only applicable to list screens for which the multi-row select behavior has been enabled.

- *Auto*: This selection is the default and will not change the selected row on the list screen.
- *By Position*: This selection will enable the Position attribute field where the selected row in the list is specified by its position within the list.
- *By Rule*: This selection specifies that the initially selected row or rows in the list will be determined by a rule. When this option is selected, the Row attribute field will be enabled where the rule can be selected.
- *First Row*: This selection specifies that the first row in the list will be selected. This is always the first row from the top of the list.
- *Last Row*: This selection specifies that the last row in the list will be selected. This is always the last row from the bottom of the list.
- *None (Clear Selection)*: This selection specifies that no rows will be selected in the list.
- **Position**: Available when the Select Rows attribute is set to By Position. The Position attribute can then be set to a numerical value specifying the row at this position will be selected. As options to this attribute, the position can be determined by counting from the top of the list down or counting from the bottom up.
- **Rule**: Available when the Select Rows attribute is set to By Rule. The Rule attribute can then be set to the First, Last, or All Rows, where the selected Rule returns either True or False. All rows is applicable only to list screens for which the multi-row select behavior has been enabled. The rule referenced here is evaluated once for and in the context of each object listed on the screen. It is expected to return a Boolean value.

Action Step Type: Open URL

The Open URL action step type defines a URL to be opened by the client device's web browser. As a option to this defined URL it is possible to also specify one or more URL parameters to be passed to the URL. These parameters can be set via a rule, which allows for the specification of dynamic values obtained from the applications data.

Child Definitions

- **URL Parameters**: Contains a value to be passed to the defined URL as a parameter (such as a CGI argument or similar). This can be set via a rule definition to expose access to any value within the mobile application.

Open URL Attributes

- **Step Name**: Contains the unique internal name of the action step definition. This must be unique among all steps within the same parent action.
- **URL**: Specifies the URL to be passed to the client device's default web browser. This can be either a constant value set directly in the field, or returned by a rule definition.

Action Step Type: Print Report

The print report action step type will print the defined report definition on a printer connected to the client device. This step can control which objects for the report are printed via an include

rule. It can also be defined to allow the user to skip printing the report. At least one report definition must exist within the same module before a print report action step can be defined.

Print Report Step Attributes

- **Step Name:** This attribute contains the unique internal name of the action step definition. This value must be unique among all steps within the same action.
- **Report:** This attribute references the report to be printed by the action step. The report definition must be defined prior to selecting it for this attribute.
- **Include Rule:** This attribute allows for optionally including only certain objects within the collection targeted by the report being printed by the action step. When a rule is referenced here it is evaluated once for, and in the context of each object instance in the report's targeted collection. The rule is expected to return a Boolean value. Only those objects for which the rule returns true will be printed in the report.
- **Allow Skip:** This attribute can allow the user to skip printing the report. When this attribute is set to false (default) the report will always be printed. When set to true, the user will be prompted to continue with the print job or to cancel.

Action Step Type: Save Tile Transactions

The Save Tile Transactions action step applies all transactions begun in a tile edit or tile list detail screen field that have not been applied. This can occur based on the overall screen flow and navigational options defined within the application project.

Save Tile Transactions Action Step Attributes

- **Step Name:** This attribute contains the unique internal name of the action step definition. This value must be unique among all steps within the same action.
- **Save Option:** This attribute specifies which transactions are to be applied when the step is executed. The options include:
 - **Active Screen Set:** Any unapplied transactions from any tile controls on any screen in the current screen set.
 - **All Screen Sets:** Any unapplied transactions from any tile controls on any screen in any screen set within the current module.

Action Step Type: SubAction

The SubAction action step type executes an action definition from within another action. When the sub-action has completed execution the parent action will continue. A SubAction step can execute an action once or iteratively based on various available criteria. A SubAction step is also used to execute an action in a different module.

The SubAction step type supports modularity within the actions of an application, providing for the reuse of actions that provide behaviors applicable to multiple areas of functionality. SubAction steps are also the primary means by which iterative processing can be implemented within the client application's behavior. This step type is also the primary means of providing

cross-module functionality. Using a SubAction step an action in one module may be executed from an action in another module.

A primary part of a SubAction step's definition is the object the action it executes targets. This object should normally be within the context of the parent action's object. As an example, if the parent action is defined for Object A, which contains a collection of Object B, the SubAction step can target an instance of Object B within that collection. The exception to this is cross-module action execution.

To execute an action across modules, the target object for the SubAction step must be an object defined in the other module. When an object from a different module is defined as the target, the actions that may be selected for the SubAction step will be those defined in that module. Execution of the parent action on the Agentry Client will then result in the action in the second module being executed as defined. The parent action will then proceed as defined after the SubAction step has completed execution.

SubAction Step Attributes

General Attributes

- **Name:** Contains the unique internal name for the step definition. This value must be unique among all steps within the same parent action.
- **Execution Type:** Specifies how the sub-action should be executed. There are several options available for this attribute, many of which providing iterative behavior. When one of these selections is chosen, the SubAction step is referred to as a Looping SubAction step. Looping SubAction steps will have additional attributes that will differ depending on how the SubAction step loops. Following are the available items for this selection.
 - *Always - Execute until stopped:* This selection will define the SubAction step to execute repeatedly until the user explicitly ends the processing. This item should only be selected when the SubAction executes and action that allows the user to either cancel or finish the processing, normally within a transaction wizard screen set.
 - *Execute Once:* This selection will execute the defined action a single time when the SubAction step is executed.
 - *Execute once if rule is true:* This selection will execute the defined action a single time only when the rule referenced in the **Execution Rule** attribute returns true. If the rule returns false, the defined sub-action will not be executed and the parent action will continue execution as defined.
 - *Execute until rule is false:* This selection will execute the defined action until the rule referenced in the **Execution Rule** attribute returns false. This rule will be evaluated after each iteration of the sub-action. This behavior means the SubAction step will always execute the defined action at least once, as the rule will not be evaluated until after execution has completed.
 - *Execute while rule is true:* This selection will execute the defined action while the rule referenced in the **Execution Rule** attribute returns true. This rule will be evaluated before the first iteration of the sub-action and before each additional iteration. This

behavior means the SubAction step may or may not execute the defined action, as the rule will be evaluated to determine if the sub-action should be executed.

- **Loop over collection:** This selection will execute the sub-action once for each object instance referenced in the **Collection** attribute. This may be limited by referencing a rule in the **Execution Rule** attribute. In this case, the **Execution Rule** will be evaluated once for, and in the context of each object instance in the collection. The rule is expected to return a Boolean value. The sub-action will then only be executed for each object instance where the rule returns true.
- **Loop over list screen:** This selection will execute the sub-action once for each object listed in the current list screen. This may be limited by referencing a rule in the **Execution Rule** attribute. In this case, the **Execution Rule** will be evaluated once for, and in the context of each object currently displayed in the list screen. The rule is expected to return a Boolean value. The sub-action will then only be executed for each listed object instance where the rule returns true.
- **Loop over selected list screen objects:** This selection will execute the sub-action once for each selected object in the current list screen. This selection is provided in support of the multi-select behavior that may be enabled for list screens. If no items are selected in the list screen, the sub-action will not be executed by the SubAction step.
- **Collection:** This attribute is only enabled when the **Execution Type** is set to “Loop over collection.” The **Collection** attribute references the object collection property the SubAction step is to loop over.
- **Execution Rule:** This attribute is enabled when the **Execution Type** is set to “Loop over collection,” “Loop over list screen,” or to one of the execution types involving a rule. The **Execution Rule** references the rule definition to be evaluated to determine the execution behavior of the sub-action.
- **Act on Object:** This attribute references the object instance the sub-action is to target. This selection is normally a child object to the object for which the parent action is defined, or an instance of the object type for the parent object. It may also be an object defined in another module. When this last type of object is selected, the available items listed for the Actions attribute will be those actions within the same module as the selected object.
- **Action:** This attribute references the action the SubAction step will execute as a sub-action to the step’s parent action. The action selected here must be defined prior to the definition of the SubAction step. The selected action must be defined for the object type selected in the **Act on Object** attribute.
- **Begin Loop with Selection:** When the Execution Type is set to Loop over displayed list items, this attribute is enabled allowing for the specification of the first item to be executed on. When selected, the first item is the one currently selected in the list. When not specified, the first item is the one at the beginning of the list.

Looping Attributes - These attributes are available only when the Execution Type attribute is set to one of the iterative options. Depending on the type of iteration, different attributes listed here will be enabled or disabled.

- **Dialog:** This attribute is available only when the **Execution Type** is set to “Loop over collection” or “Loop over list screen.” The Dialog attribute specifies whether or not to

display a message when there are no items for the SubAction to loop over. This can occur if the selected collection contains no object instances, or if the list screen currently lists no items. When Dialog is set to true, the Dialog Message attribute will be enabled allowing for the definition of the message to display.

- **Dialog Message:** This attribute is available only when the Dialog attribute is set to true. The Dialog Message can contain the message text to display when the item to loop over is empty. The default message displayed is “No valid records found.”
- **Back Up:** This attribute specifies whether to complete the SubAction step when the user clicks the Back button in a screen set displaying a transaction or fetch.
- **Allow Done:** This attribute specifies whether to display a Done button in last screen of a wizard screen set. The Done button differs from the standard Finish button in that the Done button will break out of the SubAction’s loop and return execution control to the parent action. When a SubAction step’s **Execution Type** is set to “Always - Execute until stopped,” the **Allow Done** attribute should be set to true. In other looping SubAction steps, the Done button will allow the user to end the loop regardless of any other constraints related to the looping behavior.

Action Step Type: Transaction

The transaction step type instantiates a transaction on the Agentry Client. A transaction step also defines the screen set in which the transaction instance should be displayed, if any. A transaction step can also define a target for the transaction as well as a sub-action to execute after the transaction has been completed.

A transaction step can define a transaction to be instantiated but not displayed in a screen set. This is a common occurrence when the transaction is a Delete transaction type. Other transaction types may also be defined in this manner if it is not necessary to capture any data from the user for the transaction. To not display a transaction instance, the Screen Set attribute should be set to No Screen Set.

The target object and target property for the transaction may be set in the transaction step. By default, the target object is passed to the transaction step from the action. The transaction step can then change the target to a different object, provided it is a valid item within the context of the object passed in by the action. In many cases it is not necessary to change the target of the transaction within the transaction action step. This target is normally set when there are multiple transaction steps within the same action, and one or more of those transactions is defined for the object type for which the action is defined. In the situations where the target is specified, it is normally a child object to the object type for which the parent action is defined.

The option of defining a SubAction to execute when the transaction has been completed provides a means of executing a second action from the transaction step. This sub-action will be executed only when the transaction step completes the transaction processing; i.e., if the user clicks cancel in the wizard screen set displaying the transaction the defined SubAction will not be executed. The action executed as a SubAction to the transaction step is executed as the last task of the transaction step. This results in the condition that the sub-action is executed before any apply step within the parent action. The transaction will, therefore, not yet be

applied to the object it targets. This will impact the current data values that will be accessible within that target object and may, therefore, impact how the sub-action itself is defined, as well as how any other definitions it references will be defined, specifically as to which data values the sub-action will have access.

Transaction Step Attributes

- **Step Name:** This attribute contains the unique internal name for the step definition. This attribute must be unique among all step definitions within the same parent action.
- **Transaction:** This attribute references the transaction to be instantiated by the transaction step. The transaction selected here must exist prior to defining the transaction step.
- **Screen Set:** This attribute references the screen set in which the transaction will be displayed. Only screen sets defined to display transactions may be selected for this attribute. The screen set must exist prior to defining the transaction step.
- **Target Object:** This attribute can be set to change the target of the transaction from the object instance passed in by the action, to a different object instance. This attribute is optional and, if left set to its default, the target of the transaction will be the object instance passed to the transaction step by the action. The selection of a Target Object should be to an object instance that is easily related to the action's object instance wherever possible.
- **Target Property:** This attribute is obsolete in current versions of the Agentry Mobile Platform. It exists as a result of behaviors exhibited in early versions of the platform and in current implementations is no longer necessary. It is still provided for backwards compatibility and may be deprecated in a future release.
- **SubAction:** This attribute can specify an action to be executed as a sub-action to the parent action of the transaction step. The transaction selected for this attribute will only be executed when the transaction instantiated by the transaction step is completed successfully. Note that this sub-action is executed prior to the transaction being applied.

Action Step Type: Transmit

The transmit action step type initiates communications between the Agentry Client and Agentry Server. This includes displaying the Client's built-in transmit dialog where users can select a transmit configuration and begin the transmission. Alternately the transmit step can be defined to begin the transmission automatically and to hide the transmit dialog unless an error is encountered. The transmit step also defines the non-main fetches to be processed, if any.

Each module will contain at least one action with a defined transmit step. Additional actions may be defined as needed that include transmit steps for various purposes.

The transmit step can be defined to start transmission between the client and server automatically. The default behavior is to display the Client's Transmit Dialog, where the user can select a transmit configuration and then start the transmit. When the transmit step is defined to automatically start the transmission, the transmit will begin when the step is executed. The transmit configuration used will be the last one selected by the user; optionally the transmit step can define the transmit configuration to be used.

The transmit step can also be defined to automatically finish the transmit. By default when a transmit is complete, the user must close the Transmit Dialog by clicking the finish button. The transmit step can close this screen automatically when the transmit completes successfully.

If the transmit step is defined to automatically start and finish the transmit, it can also be defined to hide the Transmit Dialog. In this case, the dialog will not be displayed to the user unless an error occurs during the transmission.

The transmit step is where non-main fetches must be selected for processing. The main fetches of a module will always be run when a transmit occurs. A non-main fetch must be explicitly selected in a transmit step and will only be processed when that transmit step is executed.

A transmit step can be defined to skip fetch processing altogether. This can only be defined when the step is first defined to use a transmit configuration for which real-time communications have been defined. The transmit step can then be defined to simply connect the user to the Server, process any pending transactions, and the remain connected to receive push data and/or to allow for background sending. Note that this behavior can negatively impact the push functionality if the push is defined to use exchange data initially generated by a fetch. This exchange data will not exist as the fetch will not be processed.

Transmit Step Attributes

- **Step Name:** Contains the unique internal name of the step definition. This value must be unique among all step definitions within the same parent action.
- **Transmit Config:** This attribute can be set to a specific transmit configuration within the application. If a transmit configuration is selected here, the user will not be able to change the transmit configuration when the transmit step is executed.
- **On-line/Off-line:** This attribute specifies whether to change the on-line state of the client when the transmit step is executed. This will override any selection the user makes on the client for this state. It will also override the on-line state of the client if it is set to Off-line as the result of a disconnect.
- **Initiate Asynchronous Transmit Only:** This attribute is only available if the Transmit Config attribute is set to a transmit configuration defined to support real-time communications. If this attribute is set to true, no fetches will be processed during the transmit. Pending transactions will be sent to the Server to be processed and complex table and data table definitions will be synchronized.
- **Allow user to skip:** This attribute allows the user to skip the transmit. This is normally only set when the parent action contains multiple step definitions, including the transmit step. The user may skip the transmit behavior when this attribute is set to true and when the client is in an Off-line state.
- **Automatically start transmission:** When this attribute is true, the transmission between the client and server will begin automatically when the transmit step is executed. The default is to require the user to click the Start button in the Transmit Dialog to being the transmission.

- **Automatically finish transmission:** When this attribute is true, the Transmit Dialog will be closed automatically when the transmit has completed successfully. The default is to require the user to click the Finish button in this dialog when the transmission has completed.
- **Hide transmission screen:** This attribute is available only when the Automatically start transmission and Automatically finish transmission attribute are both true. The Hide transmission screen can then be set to true, which will result in the Transmit Dialog not being displayed when the transmit step is executed. The Transmit Dialog is always displayed if an error occurs during transmission, regardless of this attribute setting.
- **Hide Screen Timeout:** This attribute is available only if the Hide transmission screen attribute is set to true. This timeout value is set in minutes and seconds. If the transmission takes longer than the duration entered in Hide Screen Timeout, the Transmit Dialog will be displayed to the user indicating the progress of the transmission. This timeout value should be selected based on the typical duration of a transmit for the application.

Action Step Type: Windows Command

The Windows command action step type executes a command on the client device. This step type can be defined to wait for the command to complete execution, to capture the return code of the external process, and to display an error message based on a non-zero return code. The Windows command step type is also used to display external files on the client device by setting the full path and file name as the command. This will result in the file being opened by the default application for the file type.

The command executed by the Windows command step must include the full path and file name of the executable to be run or file to be opened. When waiting for the command to return, the step will block action execution until the command completes, or until the defined wait period expires. An expired wait periods is treated as a timeout error by the Windows command step.

Additional error conditions include a non-zero return value by the command to the operating system. If a non-zero value is returned, the Windows command step will treat this as an error condition.

The timeout and the error conditions each have associated messages that may be displayed as defined in the Windows command step. This step type allows for providing the user with the option to continue or cancel the parent action's execution. Alternately, the step can be defined to not allow action execution to continue, or to not allow the user to cancel the action regardless of the error.

Windows Command Step Attributes

- **Step Name:** Contains the unique internal name for the step definition. This value must be unique among all step definitions within the same action.
- **Command Line:** This attribute contains the command to execute or pass to the operating system. This may be a string value set within the attribute field, or it may be returned from a rule definition. The command may contain one or more format strings consisting of the

property names for the object passed to the command step form the action. These format strings take the form `%propertyName`. Note for properties of type External Data, the format string will return the full path and file name of the file referenced by the property. If a rule is referenced for the command, it may not return a string containing format strings. The rule is evaluated in the context of the object passed to the Windows command step by the action. The rule is expected to return a string value.

- **Wait:** This attribute specifies whether the Windows command step should wait for the command it executes to return. The default is to not wait, in which case the command line will be executed and the step will end execution. The timeout message will not be displayed. The only error captured by the step will be if the command line cannot be executed by the operating system, e.g. if the command referenced does not exist, or the file cannot be found. When the Wait attribute is set to true, the Wait Period Limit attribute is enabled.
- **Wait Period Limit:** This attribute is enabled only when the **Wait** attribute is set to true. In this case, the **Wait Period Limit** specifies the duration of time the Windows command step should wait for the command it executes to complete processing and return. If this duration is exceeded without a return from the command, the step's defined **Timeout Message** will be displayed.
- **Error Message:** This attribute contains the text to display when an error occurs. This may be displayed if the command fails to execute, or if the command returns a non-zero value after completing execution.
- **Timeout Message:** This attribute contains the text to display when the Wait Period Limit is exceeded without a return from the command executed by the step. This behavior also requires the **Wait** attribute to be set to true.
- **Continue Label:** This attribute contains the label for the Continue button that is a part of the dialog that displays the Error Message and Timeout Message. At run time, when this button is clicked the Windows command step will complete execution and the action will execute the next defined step. This button may be hidden by selecting the option Not Allowed, preventing the user from allowing the action to continue the action's execution when an error occurs executing the defined command.
- **Cancel Label:** This attribute contains the label for the Cancel button that is a part of the dialog that displays the Error Message and Timeout Message. At run time, when this button is clicked the Windows command step will complete execution and the parent action will be canceled. This button may be hidden by selecting the option Not Allowed, preventing the user from cancelling the action when an error occurs executing the defined command.

Report

A report defines a printed tabular format for the contents of an object collection on the Agentry Client. Reports can be generated for any object collection within the application data. A report can then be printed on the client device, provided it is equipped with a printer.

The report definition defines the object collection for the report and the property values for the collection's object type to include in the printed report. The report definition does not include

any behaviors related to when to print the report. To print a report on the Agentry Client the action step type Print Report must be defined within an action.

A report defines the point size for the values it contains. It can also define the header and footer text to display in the report. Three separate header and footer values may be defined to be displayed on the left, center, and right of the page across the top and bottom of the report. Separate point sizes may be defined for the header and footer text. Note that the header and footer within the report definition are not the same as the report column headers. They are intended for general information about the report as a whole, not to label individual values within the report.

The child definition Report Columns defines the which properties to display from the object type within the target collection, as well as the column header labels within the report table. The order of the columns within the report definition will specify the order in which the columns are printed in the report from left to right.

Report Child Definitions

Report Column: A report column defines which property values are listed in the corresponding printed column of a report.

Report Attributes

General Attributes

- **Name:** Contains the unique internal name for the report definition. This must be unique among all reports within the same module.
- **Display Name:** Contains the default name for the report definition displayed on the Client.
- **For Object:** This attribute references the parent object of the collection for which the report will be generated.
- **Collection:** This attribute references the object collection property whose contents will be printed in the report.
- **Point Size:** This attribute specifies the font point size for the data printed in the report. This excludes the report header and footer, which specify their own point sizes.
- **Gridlines:** This attribute specifies whether or not to print grid lines in the report to separate columns and rows in the table. When set to true these lines will be printed in the report table. When false they will be omitted.

Header/Footer Attributes

- **Left, Center, and Right Text:** These three text boxes within the definition contain the text to display at the left, center and right sides of the report page. These values are not column labels, but are intended for general information to display in the header and/or footer of the report.
- **Point Size:** This attribute specifies the font point size of the header or footer text.
- **Bold:** This attribute specifies whether or not to display the text in bold. When true, the text will be in bold. When false it will not be.

Report Column

A report column defines which property values are listed in the corresponding printed column of a report. The column definition includes attributes for formatting the values of the column and the order of the columns within the report.

Each column defines a property of the object type in the parent reports target collection to be printed in the report. Included in the column definition is the label for the column in the report table. Basic formatting can also be defined for the column, including whether or not the column label should be in bold text, whether or not to word wrap the text within the column, alignment of the values within the column, and the width of the column as a whole.

As an alternative to selecting a property whose value will be displayed in the column, format strings or format text may be specified. To make use of this behavior a column should not be selected, but rather the Format attribute should be set to specify the value to be displayed for each object. This attribute may contain format strings referencing the properties of the object, as well as plain text. This attribute may also be set via the return value of a rule, which will be expected to build the entire string to be displayed in the column for each object.

Report Column Attributes

- **Name:** Contains the unique internal name for the report column definition. This value must be unique among report columns within the same report.
- **Label:** Contains the column label for the header row of the report table.
- **Object Property:** This attribute references the object property to be printed in the column for each object instance in the target collection.
- **Bold Label:** This attribute specifies whether or not the label for the column should be printed in bold text. When true the label will be printed with bold text.
- **Wrap:** This attribute specifies whether or not the values of the column should be word wrapped. If true the values printed in the report will be word wrapped to fit in the space of the column. If false, the column width will be expanded to allow for the size of the text.
- **Alignment:** This attribute specifies the alignment of the text within the column. The options for this attribute are “Left Justified,” “Right Justified,” or “Centered.”
- **Column Width:** This attribute specifies the width of the column. The units for this attribute are the number of average sized characters. If left set to Auto, the width of the column will be set by evenly spacing all Auto Width columns within the report, after space is allocated for all columns with a defined width.
- **Format:** This attribute can contain a combination of format strings and standard text to specify the format of the values printed in the column. Alternately the value printed in the column can be the return from a rule definition. If a rule is used for this attribute, it will be evaluated once for, and the context of each object instance within the reports target collection property. It is expected to return a string value. If this attribute is set either format text or a rule definition, the **Object Property** attribute should be set to None.

Rule Function Terms Overview

Rule functions terms are the heart of most rule definitions within an application. While there are situations where a rule may be defined to contain a single rule term that returns the value of a global or other such data definition type, most rules are more complex than this and consist of multiple function calls.

Most rule functions take one or more arguments, each of which contains a data value for the function. When the function is evaluated, these data values are processed in some manner. The result of this processing is a single return value that is passed to the function's caller. A function will always provide the caller with a value in the data type the caller asks for. Not all function support all data types for their return values. If a data type is not one supported by the function, that function will return the null-equivalent of that data type.

There are over a hundred different functions available for a rule definition. These are organized into Function Categories. These categories denote the general types of behavior for the functions. The rule editor presents the functions to the developer organized into one of these categories.

- **Conversion Functions** - Conversion functions set the context of a given term to a specified data type. A conversion function supports all return types within the rule definition. The names of conversion functions dictate what data type they will set for the context of a function call.
- **Logical Functions** - Logical functions are those that provide the comparison and decision making functionality to a rule. This includes if-then-else and comparison operations and behaviors.
- **Mathematical Functions** - Mathematical functions provide math operations to rules. This includes addition, subtraction, multiplication, division, and modulus operations, as well other mathematical functions, such as rounding, and working with significant digits.
- **Property Functions** - Property functions are those that operate on properties, usually of a certain data type. Most property functions are provided for the intended purpose of working with a given type of property, such as an object collection or external data property.
- **String Functions** - String functions provide behaviors for manipulating string values, including concatenation and parsing operations, string search and replacement, and other string-related operations.
- **System Functions** - System functions are those that provide access to information about the Agentry Client's host system, or information that is general to the client. This can be information such as the system's time and date, or the user ID of the current user. This category also includes functions to access hardware components of the client device such as barcode scanners and GPS units.
- **Table Functions** - The table functions provide access to the records of complex table and data tables stored on the Agentry Client.

Note that the function categories do not directly impact where a function can be used, or which rules can use a given function. The categories are an organizational aid built into the rule editor to aid the developer in locating the rule function that is needed.

Conversion Functions for Rules

The Conversion functions category of rule function terms provide the means for changing the context in which a function or data term within a rule is called. Within this category of conversion functions there is one function for the integral number, string, and property data types. The decimal number data type has two conversion functions, one of which is for use with significant digit math.

The name for each conversion function represents the data type to which it will set the context of the term that is its argument. The function term name, then, does not represent the data type *to* which a value will be converted, but the data type *from* which a value will be converted. Each conversion function supports all return types.

Conversion functions are most commonly used when it is necessary to obtain the return value of a function that may exhibit different behaviors in different contexts, or when the desired return data type does not match the supported return type of a given function. The caveat to this is that the conversion desired is type safe.

An example of this is the string function @FIND. This function searches a source string for a given sub-string. The function supports three return types, string, integral number and Boolean. The context in which this function is called will then dictate what type of value it will return. In a string context the function returns the sub-string when found within the source string. When called in an integral number context, the function returns the position, as a number, of the first character within the source string of the found sub-string. In a Boolean context the function will return true if the sub-string is found and false when it is not. For this function call a conversion function may be used to change the data type of the context in which it is called in order to obtain the desired value.

@FROM_DECIMAL_NUMBER

The FROM_DECIMAL_NUMBER function sets the context of its single parameter to a data type of decimal number. It supports the decimal number, integral number, string, and property return types. The value of its single parameter will be converted from the decimal number data type to the data type of the context of the FROM_DECIMAL_NUMBER function call.

One of the main uses of this function is to set the context of another function call to a decimal number. Certain functions do not directly support non-numeric data types for return. The FROM_DECIMAL_NUMBER function allows for these other functions to be called in a decimal number context and to then return that value in a data type such as string. While this function supports the decimal number return type, it is unnecessary to call this function in this context.

Parameters

@FROM_DECIMAL_NUMBER (Convert Parameter, [Precision, [Rounding Method]])	
Convert Parameter	Required decimal number parameter, contains the value to be converted to the data type of the function's context.
Precision	Optional integral number parameter, contains the precision to which the returned decimal number should be rounded. Positive values specify the number of digits after the decimal place. Negative numbers specify number of digits before the decimal.
Rounding Method	Optional integral number parameter, specifies how the return value should be rounded. The default is to round to the nearest value. If this parameter is set to 1, the rules pertaining to NIST rounding will be used to round the value returned by the function.

Supported Return Types

- Integral Number
- Decimal Number
- String
- Property

@FROM_INTEGRAL_NUMBER

The FROM_INTEGRAL_NUMBER function sets the context of its single parameter to a data type of integral number. It supports the decimal number, integral number, string, and property return types. The value of its single parameter will be converted from the integral number data type to the data type of the context of the FROM_INTEGRAL_NUMBER function call.

One of the main uses of this function is to set the context of another function call to integral number. Certain functions do not directly support non-numeric data types for return. The FROM_INTEGRAL_NUMBER function allows for these other functions to be called in an integral number context and to then return that value in a data type such as string. While this function supports the integral number return type, it is unnecessary to call this function in this context.

Parameters

@FROM_INTEGRAL_NUMBER (Convert Parameter)	
Convert Parameter	Required integral number parameter, contains the value to be converted to the data type of the function's context.

Supported Return Types

- Integral Number
- Decimal Number
- String
- Property

@FROM_STRING

The FROM_STRING function sets the context of its single parameter to a data type of string. It supports the decimal number, integral number, string, and property return types. The value of its single parameter will be converted from the decimal number data type to the data type of the context of the FROM_STRING function call.

One of the main uses of this function is to set the context of another function call to string. The FROM_STRING function allows for other functions to be called in a string context and to then return that value in a data type such as integral or decimal number, depending on the context of the FROM_STRING function call. While this function supports the string return type, it is unnecessary to call this function in this context.

Note that converting from a string to a numeric data type is not considered type safe. Such operations should be limited in use and all reasonable precautions should be made to ensure the string value being converted to a numeric data type contains only numeric characters. The FROM_STRING function processes each character of a string one at a time and, in a numeric conversion, will stop processing with the first non-numeric character found in the source string. The value returned will then be the numeric value at the point the processing ended, which is not likely to be a useful value.

Parameters

@FROM_STRING (Convert Parameter)	
Convert Parameter	Required string parameter, contains the value to be converted to the data type of the function's context.

Supported Return Types

- Integral Number
- Decimal Number
- String
- Property

@FROM_SIG_DECIMAL_NUMBER

The FROM_SIG_DECIMAL_NUMBER function sets the context of its single parameter to a data type of decimal number. It supports the decimal number, integral number, string, and property return types. The value of its single parameter will be converted from the decimal number data type to the data type of the context of the FROM_SIG_DECIMAL_NUMBER function call. The decimal number of this parameter will respect the rules of significant digit math.

One of the main uses of this function is to set the context of a decimal value that is either not stored in a decimal property, or one that is stored in a decimal number property but that does not have the significant digits math attribute set.

An optional parameter to this function is Precision. This parameter will specify the number of digits after the decimal to keep, with the last digit being rounded. If the precision is greater than the number of digits after the decimal, the value will be padded with zeros up to the specified precision.

Parameters

@FROM_SIG_DECIMAL_NUMBER (Convert Parameter [, Precision])	
Convert Parameter	Required decimal number parameter, contains the value that will be treated as a decimal number with respect of significant digit math.
Precision	Optional integral number parameter, specifies the number of digits to keep after the decimal in the value provided by Convert Parameter. If this value is greater than the number of digits after the decimal the value will be padded with zeros up to the specified precision.

Supported Return Types

- Integral Number
- Decimal Number

- String
- Property

@FROM_PROPERTY

The FROM_PROPERTY function takes a variable number of arguments. Each argument is evaluated as a property and this evaluation is within the context dictated by the argument that precedes it in the arguments to the function. The overall purpose of this function is to provide a kind of drill-down access to the value of a property that may be a descendent of the current object.

One of the main uses of this function is to set the context of another function call to property. Certain property functions do not directly support other data types for return. The FROM_PROPERTY function allows for these other functions to be called in a property context and to then return that value in another data type.

When taking multiple parameters, the FROM_PROPERTY function is likely to be used in an overall search of a collection for a given object based on a property value, returning another property within the same object instance.

Parameters

@FROM_PROPERTY (Property 1 [, ..., Property N])	
Property 1	Required property parameter, contains the value to be evaluated as a property in the context of the function call. This parameter sets the context of the next parameter to the function, if present. If this is the only parameter, it will be returned in the context of the function call.
Property N	Optional property parameter(s), contains the value to be evaluated as a property in the context of the preceding parameter to the function. This parameter sets the context of the next parameter to the function, if present. If this is the last parameter, it will be returned in the context of the function call.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String
- Property

Logical Functions for Rules

The Logical category of rule function terms within the rule definition provide the decision making and comparison logic to a rule. Many, though not all, of these functions support the Boolean return type and will return true or false in this context based on some decision or comparison. The functions within this category provide behaviors including conjunctions, value comparisons, and if-then-else if and switch-case logic.

Unlike similar constructs in other development tools, many of the functions within the logical category support more than two arguments. As an example, the @AND function will take two or more arguments, returning true only when all of its arguments are true. In other development languages to provide similar logic, multiple operators may be needed, as in:

```
if (value1 && value2 && value3 && value4)
```

Here the presence of multiple and operators are required. In the rule functions, the same logic would use a single @AND function call, with each value passed as an argument to the function:

```
AND (value1, value2, value3, value4)
```

Many of the other functions provide similar support within the context of their behavior.

@AND

The AND function performs a logical conjunction between its parameters, returning true or false based on this conjunction. Each of its parameters is evaluated in the order provided as Boolean values. If any parameter is evaluated as false, the return value is false. Otherwise the function returns true. The function must have at least one parameter and may contain as many more as is needed.

Parameters

@AND (Expression 1 [, ..., Expression N])	
Expression 1	Required Boolean parameter, the first to be evaluated by the function. If false, evaluation stops and the function returns false. If true, the function will return true if no other parameters are provided or evaluate the next parameter.
Expression N	Optional Boolean parameters, each evaluated by the function in the order provided. Evaluation stops for the first false value found and the function returns false. Otherwise the function returns true.

Supported Return Types

Boolean

@CASE

This function has been deprecated and will not be supported in future releases. It should be replaced with one of the following: CASE_INT, CASE_STRING, CASE_DEC, or IF. The CASE function provides switch-case logic, allowing for the evaluation of a single test value for the purpose of returning one of a multiple number of possible values. The CASE function takes a variable number of parameters, but with a minimum of three. The first parameter is evaluated as an integral number. This value is then treated as a positional value for one of the other parameters to the function, with the second parameter at position 1. The parameter at the position specified by the position parameter is then returned. The data type of the other parameters varies depending on the function's context. For example, if the context of the function call is a string, the parameters Position1 through PositionN of the function will be treated as strings.

Parameters

@CASE (Position To Match, Position1 [, ..., PositionN])	
Position To Match	Required integral number parameter, indicating which of the case parameters the function should return.
Position 1	Required parameter, evaluated when the Position To Match parameter evaluates to 1. The data type is dictated by the context in which the function is called.
Position N	Optional parameter, evaluated when the Position to Match parameter evaluates to N, where N is the position of the parameter in the function's parameter list. The data type is dictated by the context in which the function is called.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String
- Property

@CASE_INT

The CASE_INT function is used to return a single value from a variable list of multiple possible returns, with a comparison made between a switch and 1 or more case values, both of type integral number. The first parameter to the function is evaluated as an integral number. The last parameter to the function is the default return value and is optional. Parameters between the first and last are provided in pairs. The first parameter in a pair is the value to which the switch value is compared. If these two values are equal, the second value of the pair is returned by the function. The comparison between the switch value and the first parameter of a pair is performed as an integer comparison. The data type for the second parameter in each pair is dependent on the context in which the function is called. If none of the case parameters match the switch parameter and a default parameter is provided, that parameter is evaluated and returned by the function. If no default parameter is provided, the default null equivalent of the context's data type is returned, e.g. 0, False, null string, etc.

Parameters

@CASE_INT (Integer To Match, Integer 1, Return 1 [, ..., Integer N, Return N] [, Otherwise])	
Integer To Match	Required integral number parameter, contains the value upon which the function will switch, i.e. compare against each of the Integer N parameters in turn until a match is found.
Integer 1	Required integral number parameter, contains the value to which Integer To Match is compared. This parameter must be followed by the Return 1 parameter, which is the value returned when Integer To Match matches the Integer 1 value.
Return 1	Required parameter with a context-dependent data type, contains the value returned if Integer To Match matches Integer 1.

@CASE_INT (Integer To Match, Integer 1, Return 1 [, ..., Integer N, Return N] [, Otherwise])	
Otherwise / Integer N	Optional parameter(s), the data type in which it is evaluated is dependent on whether it is the last parameter to the function, or if it is followed by another parameter. When this is the last parameter to the function, it will be evaluated in the data type corresponding to the context of the function call. In this situation the parameter is the default Otherwise parameter, evaluated by the function when Integer To Match does not match any of the Integer N parameters. If this parameter is followed by another function parameter, it is evaluated as an integral number. In this situation it is evaluated by the function to determine if the value matches the Integer To Match parameter value. If it matches, the subsequent parameter is evaluated by the function. If not, the function evaluates the next parameter. Multiple Integer N parameters can be provided with the requirement that they are paired with corresponding Return N parameters. Only one Otherwise parameter may be provided.
Return N	Optional parameter(s) with a context-dependent data type, contains the value returned if Integer To Match matches the corresponding Integer N parameter.

Supported Return Types:

- Boolean
- Integral Number
- Decimal Number
- String
- Property

@CASE_STRING

The CASE_STRING function is used to return a single value from a variable list of multiple possible returns, with a comparison made between a switch and one or more case values, each of type string. The first parameter to the function is evaluated as a string. The last parameter to the function is the default return value and is optional. Parameters between the first and last are provided in pairs. The first parameter in a pair is the value to which the switch value is compared. If these two values are equal, the second value of the pair is returned by the function. The comparison between the switch value and the first parameter of a pair is performed as a case-sensitive string comparison. The data type for the second parameter in each pair is dependent on the context in which the function is called. If none of the case

parameters match the switch parameter, and a default, non-paired parameter is provided, that parameter is evaluated and returned by the function. If no default parameter is provided, the default null equivalent of the context's data type is returned, e.g. 0, False, null string, etc.

Parameters

@CASE_STRING (String To Match, String 1, Return 1 [, ..., String N, Return N,] [, Otherwise])	
String To Match	Required string parameter, contains the value upon which the function will switch, i.e. compare against each of the String N parameters in turn until a match is found.
String 1	Required string parameter, contains the value to which String To Match is compared. This parameter must be followed by the Return 1 parameter, which is the value returned when String To Match matches the String 1 value.
Return 1	Required parameter with a context-dependent data type, contains the value returned if String To Match matches String 1.
Otherwise / String N	Optional parameter(s), the data type in which it is evaluated is dependent on whether it is the last parameter to the function, or if it is followed by another parameter. When this is the last parameter to the function, it will be evaluated in the data type corresponding to the context of the function call. In this situation, the parameter is the default Otherwise parameter, evaluated by the function when String To Match does not match any of the String N parameters. If this parameter is followed by another function parameter, it is evaluated as a string. In this situation, it is evaluated by the function to determine if the value matches the String To Match parameter value. If it matches, the subsequent parameter is evaluated by the function. If not, the function evaluates the next parameter. Multiple String N parameters can be provided with the requirement that they are paired with corresponding Return N parameters. Only one Otherwise parameter may be provided.
Return N	Optional parameter(s) with a context-dependent data type, contains the value returned if String To Match matches the corresponding String N parameter.

Supported Return Types:

- Boolean
- Integral Number
- Decimal Number
- String
- Property

@EQBOOL

The EQBOOL function takes two or more parameters, each of which is evaluated as a Boolean value and returning true if all parameters are either true or all are false. If all parameters have the same Boolean value, the function will return true. Otherwise, it will return false. The function will end evaluation and return false upon the first parameter found to be different than others passed to it.

Parameters:

@EQBOOL (Boolean 1 [, ..., Boolean N])	
Boolean 1	Boolean required parameter, evaluated for comparison to all other parameters to the function.
Boolean N	Optional additional Boolean parameter(s), evaluated for comparison to Boolean 1.

Supported Return Types

Boolean

@EQDEC

The EQDEC function takes two or more parameters, each evaluated as a decimal value, compares them for equality, returning true if all are equal or false if any are found to be different. The function will end evaluation of all parameters at the point where the first different value is found. If only a single parameter is provided, the function returns true.

Parameters

@EQDEC (Decimal 1 [, ..., Decimal N])	
Decimal 1	Required decimal number parameter, evaluated by the function for comparison to all other parameters.

@EQDEC (Decimal 1 [, ..., Decimal N])	
Decimal N	Optional decimal number parameter(s), each evaluated by the function for comparison to <code>Decimal 1</code> .

Supported Return Types

Boolean

@EQNUM

The EQNUM function takes two or more parameters, each evaluated as an integral number, compares them for equality, and returns true if all values are equal, or false if one or more are different. This function will end evaluation of all subsequent parameters after the first parameter is found to be different. If only a single parameter is provided, this function will return true.

Parameters

@EQNUM(Integer1 [, ..., IntegerN])	
Integer 1	Required integral number parameter, evaluated by the function for comparison to all other function parameters.
Integer N	Optional integral number parameter(s), evaluated by the function for comparison to <code>Integer 1</code> .

Supported Return Types

Boolean

@EQSTR

The EQSTR function takes one or more parameters, each evaluated as a case-sensitive string, and compares them for equality, returning true if all values are equal, or false if one or more values are different. The function will end evaluation of all subsequent parameters when the first different value is found. The function will return true if only a single parameter is provided.

Parameters

@EQSTR(String1 [, ..., StringN])	
String 1	Required string parameter, evaluated by the function for comparison to all other parameters to the function.

@EQSTR(String1 [, ..., StringN])	
String N	Optional string parameter(s), each evaluated by the function for comparison to <code>String 1</code> .

Supported Return Types

Boolean

@GT

The GT function takes two or more integral number parameters, comparing the second through the last parameters to the first, returning true if the first parameter is greater than all subsequent parameters. If any parameter is found to be greater than or equal to the first, the function will not evaluate any subsequent parameters and will return false.

Parameters

@GT (Integer 1, [, ..., Integer N])	
Integer 1	Required integral number parameter, contains the value to which all other parameters will be compared.
Integer N	Optional integral number parameter(s), each containing a value to be compared against <code>Integer1</code> . If <code>Integer1</code> is less than or equal to this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Integer 1</code> is greater than this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@GTDEC

The GTDEC function takes two or more decimal number parameters, comparing the second through the last parameters to the first, returning true if the first parameter is greater than subsequent parameters. If any parameter is found to be greater than or equal to the first, the function will return false. It will not evaluate any subsequent parameters.

Parameters

@GTDEC (Decimal 1 [, ..., Decimal N])	
Decimal 1	Required decimal number parameter, contains the value to which all other parameters will be compared.
Decimal N	Optional decimal number parameter(s), each containing a value to be compared against <code>Decimal 1</code> . If <code>Decimal 1</code> is less than or equal to this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Decimal 1</code> is greater than this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@GTEQ

The GTEQ function takes two or more integral number parameters, comparing the second through the last parameter to the first parameter, and returning true if the first parameter is greater than or equal to all subsequent parameters. If any other parameter is found to be greater than the first, the function will return false. It will not evaluate any subsequent parameters. If only a single parameter is provided the function will always return true.

Parameters

@GTEQ (Integer 1 [, ..., Integer N])	
Integer 1	Required integral number parameter, contains the value to which all other parameters will be compared.
Integer N	Optional integral number parameter(s), each containing a value to be compared against <code>Integer 1</code> . If <code>Integer 1</code> is less than this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Integer 1</code> is greater than or equal to this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@GTEQDEC

The GTEQDEC function takes two or more decimal number parameters, comparing the second through the last parameters to the first, returning true if the first parameter is greater than or equal to all subsequent parameters. If any other parameter is found to be greater than the first, the function will return false. It will not evaluate any subsequent parameters. If only a single parameter is provided, this function will always return true.

Parameters

@GTEQDEC (Decimal 1 [, ..., Decimal N])	
Decimal 1	Required decimal number parameter, contains the value to which all other parameters will be compared.
Decimal N	Optional decimal number parameter(s), each containing a value to be compared against <code>Decimal 1</code> . If <code>Decimal 1</code> is less than this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Decimal 1</code> is greater than or equal to this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@IF

The IF function provides the `if-then-else/else if` logic to rules. The function can take a variable number of parameters, and will behave differently based on the parameters. First, it can take a single parameter whose value is evaluated as a Boolean. If this parameter is true, the IF function will return true; otherwise it will return false. Note that this form of the function has limited use, as the condition being checked can be passed directly to what would otherwise be the caller of the IF function, without actually making the IF function call.

Second, the function can take a multiple number of parameters to provide the `if-then-else/else if` logic. The first parameter is a `Condition` parameter, and is evaluated as a Boolean. When true is returned, its corresponding `Then` parameter is evaluated and the resulting value is returned.

If a `Condition` parameter returns false, its corresponding `Then` parameter is not evaluated. The next `Condition` parameter is evaluated. A final optional parameter can be provided as the `Else` parameter. This parameter is evaluated when all `Condition` parameters have evaluated to false. The value resulting from evaluation of the `Else` parameter is then returned by the IF function.

All `Then` parameters and the `Else` parameter are evaluated in the context of the `IF` function call. The expected data type for these parameters is then the data type of that context.

Parameters

@IF (Condition 1 [,Then 1] [, ... Else If Condition N, Then N] [, Else])	
Condition 1	Required Boolean parameter, contains the value evaluated by the function to determine a return. When true, <code>Then 1</code> is returned if specified. When false, either the next <code>Else If Condition N</code> parameter is evaluated if specified. <code>Else</code> is evaluated if no <code>Else If Condition</code> exists. The context null-equivalent is returned if neither an <code>Else If Condition</code> or <code>Else</code> parameter is provided. If <code>Condition 1</code> is the only parameter, its Boolean value will be returned.
Then 1	Optional parameter, evaluated in the context of the function call. This parameter is evaluated when <code>Condition 1</code> is true. The value returned by the evaluation of <code>Then 1</code> is then returned by the <code>IF</code> function.
Else / Else If Condition N	Optional parameter(s), evaluated when <code>Condition 1</code> or the preceding <code>Else If Condition</code> parameter returns false. The data type for this parameter is dependent on whether it is the last parameter to the function or is followed by another parameter. If it is the last parameter, it is evaluated in the context of the function call. Its is the <code>Else</code> parameter to the function and will be evaluated when all preceding <code>Condition</code> parameters have returned false. If this parameter is not the last for the function, it is evaluated as a Boolean and is treated as an <code>Else IF Condition N</code> parameter. If it evaluates to true the corresponding <code>Then N</code> parameter will be evaluated by the function.
Then N	Optional parameter, evaluated in the context of the function call. This parameter is evaluated when its corresponding <code>Else If Condition N</code> parameter returns true. The value returned by the evaluation of <code>Then N</code> is then returned by the <code>IF</code> function.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String

- Property

@LT

The LT function takes two or more integral number parameters, comparing the second through last parameters with the first and returning true if the first parameter is less than all subsequent parameters. If any subsequent parameter is found to be equal to or less than the first parameter the function will return false. It will not evaluate any subsequent parameters. If only a single parameter is provided this function will return true.

Parameters

@LT(Integer 1 [, ..., Integer N])	
Integer 1	Required integral number parameter, contains the value to which all other parameters will be compared.
Integer N	Optional integral number parameter(s), each containing a value to be compared against Integer 1. If Integer 1 is greater than or equal to this parameter, the function returns false and will not evaluate any subsequent parameters. If Integer 1 is less than this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@LTDEC

The LTDEC function takes two or more decimal number parameters, comparing the second through last parameters with the first and returning true if the first parameter is less than all subsequent parameters. If any subsequent parameter is found to be equal to or less than the first parameter the function will return false. It will not evaluate any subsequent parameters. If only one parameter is provided to this function it will return true.

Parameters

@LTDEC (Decimal 1 [, ..., Decimal N])	
Decimal 1	Required decimal number parameter, contains the value to which all other parameters will be compared.

@LTDEC (Decimal 1 [, ..., Decimal N])	
Decimal N	Optional decimal number parameter(s), each containing a value to be compared against <code>Decimal 1</code> . If <code>Decimal 1</code> is greater than or equal to this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Decimal 1</code> is less than this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@LTEQ

The LTEQ function takes two or more integral number parameters, comparing the second through last parameter with the first parameter and returning true if the first parameter is less than or equal to all subsequent parameters. If any subsequent parameter is found to be less than the first parameter, the function will return false. It will not evaluate any subsequent parameters. If only a single parameter is provided, the function will return true.

Parameters

@LT(Integer 1 [, ..., Integer N])	
Integer 1	Required integral number parameter; contains the value to which all other parameters will be compared.
Integer N	Optional integral number parameter(s), each containing a value to be compared against <code>Integer 1</code> . If <code>Integer 1</code> is greater than this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Integer 1</code> is less than or equal to this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@LTEQDEC

The LTEQDEC function takes two or more decimal number parameters, comparing the second through last parameter with the first parameter and returning true if the first parameter is less than or equal to all subsequent parameters. If any subsequent parameter is found to be less than the first parameter, the function will return false and it will not evaluate any subsequent parameters. If only one parameter is provided, the function will return true.

Parameters

@LTEQDEC (Decimal 1 [, ..., Decimal N])	
Decimal 1	Required decimal number parameter; contains the value to which all other parameters will be compared.
Decimal N	Optional decimal number parameter(s), each containing a value to be compared against <code>Decimal 1</code> . If <code>Decimal 1</code> is greater than this parameter, the function returns false and will not evaluate any subsequent parameters. If <code>Decimal 1</code> is less than or equal to this parameter, the function will continue to evaluate any additional parameters, or return true if no other parameters are provided.

Supported Return Types

Boolean

@NAND

The NAND function takes one or more Boolean parameters and evaluates each until the first true value is found or until all parameters have been evaluated. If a true parameter value is found, the function returns false. It will return true if all parameters evaluate to false. The function will end evaluation when the first true value is found.

Parameters

@NAND (Expression 1 [, ..., Expression N])	
Expression 1	Required Boolean parameter, evaluated by the function for its Boolean value. If false, the function will return true. If true, the function will continue to evaluate the next parameter.
Expression N	Optional Boolean parameter, evaluated by the function for its Boolean value. If false, the function will return true. If true, the function will continue to evaluate the next parameter, or if no more parameters are provided, the function will return false.

Supported Return Types

Boolean

@NOR

The NOR function takes one or more parameters and returns true if all parameters are false, or false if any parameter is found to be true. This function supports the Boolean return type. Each parameter is evaluated as a Boolean.

Parameters

@NOR(Expression1 [, ..., ExpressionN])	
Expression 1	Required Boolean parameter; if true the function will return false. If false, the function will continue evaluating the next parameter, or return true if no additional parameters are provided.
Expression N	Optional Boolean parameter(s), each evaluated for their Boolean value. The function will return false and end evaluation with the first true parameter found. If all parameters are true, the function will return false.

Supported Return Types

Boolean

@NOT

The NOT function will evaluate all of the parameters as Booleans and return false if all parameters are true, or true if one or more parameters are false. This function supports the Boolean return type.

Most calls to this function provide only a single Boolean parameter, the value for which is inverted and returned.

Parameters

@NOT (Expression 1 [, ..., Expression N])	
Expression 1	Required Boolean parameter; if the value is true, the function will return false and not evaluate any subsequent parameters. If the value is false and if there are no subsequent parameters, the function will return true. If additional parameters are present, they will be evaluated only when Expression 1 is false.

@NOT (Expression 1 [, ..., Expression N])	
Expression N	Optional Boolean parameter(s), evaluated by the function in the order provided. The function will return false for the first parameter found to be true. If all parameters evaluate to true, the function will return false.

Supported Return Types

Boolean

@OR

The OR function takes two or more Boolean parameters and will return true if any one or more of its parameters evaluates to true; otherwise it returns false. This function can be called in a Boolean context.

Parameters

@OR(Expression 1 [, ..., Expression N])	
Expression 1	Required Boolean parameter; contains the first value evaluated by the function for true or false. If this parameter evaluates to true, the function will end evaluation and return true to the caller.
Expression N	Optional Boolean parameter(s); contains the next value evaluated by the function. Additional parameters are evaluated until a true parameter is found. The function will end evaluation and return true for the first true parameter found.

Supported Return Types

Boolean

@XOR

The XOR function provides the exclusive or logic and can take one or more parameters, each of which is evaluated as a Boolean, and returning true when one and only one of its parameters is true. If all of the parameters evaluate to false, or if two or more parameters evaluate to true, this function will return false. This function will end evaluation of all parameters and return false when the second true value is found.

Parameters

@XOR (Expression 1 [, ..., Expression N])	
Expression 1	Required Boolean parameter; evaluated by the function in relation to all other parameters looking for an exclusive true value among the parameter list.
Expression N	Optional Boolean parameter(s); evaluated by the function in relation to all other parameters looking for an exclusive true value among the parameter list. If both this and any preceding parameter are true, the function will end evaluation and return false. No additional parameters will be evaluated.

Supported Return Types

Boolean

Mathematical Functions for Rules

The Mathematical category of rule function terms available within the rule definition provide the mathematical operations. These include addition, subtraction, multiplication, division, and modulus, as well as several other math-related operations. Additional operations include returning the minimum and maximum values from a set of values, limiting a value to a given range, square root operations, rounding, and other similar functions.

@ABS

The ABS function returns the absolute value of the given numerical parameter. The function takes a single parameter, which is evaluated as either an integral or decimal value matching the context of the function call.

Parameters

@ABS (Number)	
Number	Required number parameter, evaluated as either an integral or decimal number depending on the context of the function call. The absolute value of this parameter is returned.

Supported Return Types

- Decimal Number
- Integral Number

@DIFF

The DIFF function takes two or more numeric parameters evaluated in the context of the function call. The second parameter through the last are subtracted from the first parameter. The function then returns the result. This function supports an integral or decimal number return type.

Parameters

@DIFF(Number 1 [, ..., Number N])	
Number 1	Required parameter, in a numeric context, contains the value from which all subsequent parameters will be subtracted. Evaluated as an integral or decimal number, matching the context of the function call.
Number N	Optional parameter(s), in a numeric context, contains the value(s) from which all subsequent parameters will be subtracted. Evaluated as integral or decimal number(s), matching the context of the function call.

Supported Return Types

- Integral Number
- Decimal Number

@DISTANCE

The DISTANCE function takes four decimal parameters assumed to be latitude and longitude values for two map positions, and returns the resulting distance in meters as a decimal value.

When working with GPS location values, this function should not be used. See the System functions GPS_LOCATION, LATITUDE, LONGITUDE, DISTANCE_MILES, DISTANCE_KILOMETERS, LOCATION, and IS_VALID_LOCATION.

Parameters

@DISTANCE (x1, y1, x2, y2)	
x1	Required decimal number parameter; contains the x coordinate of the first position.
y1	Required decimal number parameter; contains the y coordinate of the first position.
x2	Required decimal number parameter; contains the x coordinate of the second position.

@DISTANCE (x1, y1, x2, y2)	
y2	Required decimal number parameter; contains the y coordinate of the second position.

Supported Return Types

Decimal Number

@DIV

The DIV function takes two parameters, for which the data type is dependent on the context of the function. It divides the first parameter by the second and returns the quotient as either an integral number or decimal number, depending on the function's context.

Parameters

@DIV (Dividend, Divisor)	
Dividend	Required parameter containing the dividend value or the value to be divided. Evaluated as either an integral or decimal number, depending on the function's context.
Divisor	Required parameter containing the divisor value or the value to divide into Dividend. Evaluated as either an integral or decimal number, depending on the function's context.

Supported Return Types

- Integral Number
- Decimal Number

@FORMAT_DECIMAL

The FORMAT_DECIMAL function converts the given decimal number parameter into a string. It takes up to five additional optional parameters that are used in formatting the converted string value. The first parameter is the value to be converted and is required. This parameter is evaluated as a decimal, though the value itself may be either an integral or decimal number data type. This function should be used for any read-only detail screen field displaying a decimal value.

Parameters

@FORMAT_DECIMAL (Decimal [, Precision, Use Thousands Separator, Use Lead Zero, Decimal Point, Thousands Separator])	
Decimal	Required decimal number parameter; contains the value to be formatted to a string by the function. If this parameter is a decimal number property, the definition of that property's rounding attributes will affect the final value, specifically when rounding to a specified precision.
Precision	Optional integral number parameter; contains the number of digits after the decimal to keep when converting <code>Decimal</code> . The last kept digit will be rounded. If <code>Decimal</code> is a decimal number property, the property's rounding attributes will determine the behavior of rounding for the value returned. If <code>Decimal</code> is a decimal property, the precision defined for the property will take effect before the function applies any additional precision to the resulting string returned. If this value is not specified, the precision will be determined automatically by the function.
Use Thousands Separator	Optional Boolean parameter with a default value of false. When true, the final string value returned will contain a comma to denote thousands, millions, etc. When false, no comma will be present in the resulting string returned by the function.
Use Lead Zero	Optional Boolean parameter with a default value of false. When true, the final string value returned will contain a leading 0 in the ones position for decimals that contain only fractional values; e.g. when false or not specified <code>.23</code> ; when true <code>0.23</code> .
Decimal Point	Optional string parameter with a default value of a decimal point (<code>.</code>). This value may be set to any single character to be used in place of a decimal point. Many locales use a comma to denote the fractional portion of a decimal value.

@FORMAT_DECIMAL (Decimal [, Precision, Use Thousands Separator, Use Lead Zero, Decimal Point, Thousands Separator])	
Thousands Separator	Optional string parameter with a default value of comma (.). This value can be set to any single character to be used in place of a comma to separate thousands and hundreds, millions and hundred thousands, etc. This parameter is only evaluated by the function when Use Thousands Separator is true. Many locales use a period (.) as the separator character.

Supported Return Types

String

@MAX

The MAX function takes one or more parameters containing numerical values and compares each to the other, returning the value of the parameter with the greatest value. This function can be called in a decimal or integral number context and will evaluate its parameters according to that context.

Parameters

@MAX (Number 1 [, ..., Number N])	
Number 1	Required parameter evaluated as the data type of the context in which the function is called. Contains the first value to be compared against all other parameters by the function.
Number N	Optional parameter(s) evaluated as the data type of the context in which the function is called. Each contains the value(s) to be compared against all other parameters to the function.

Supported Return Types

- Integral Number
- Decimal Number

@MIN

The MIN function takes two or more parameters containing numerical values and compares each to the other, returning the value of the parameter with the least value. This function can be

called in a decimal or integral number context and will evaluate its parameters according to that context.

Parameters

@MIN(Number 1 [, ..., Number N])	
Number 1	Required parameter evaluated as the data type of the context in which the function is called. Contains the first value to be compared against all other parameters to the function.
Number N	Optional parameter(s) evaluated as the data type of the context in which the function is called. Each contains the value(s) to be compared against all other parameters to the function.

Supported Return Types

- Integral Number
- Decimal Number

@MOD

The MOD function performs a modulus operation on its two parameters, dividing the first by the second and returning the remainder. The parameters are evaluated as either decimal or integral numbers, matching the context of the function.

Parameters

@MOD (Dividend, Divisor)	
Dividend	Required parameter evaluated as either an integral or decimal number matching the context of the function call. Contains the dividend value, or the value to be divided by <i>Divisor</i> .
Divisor	Required parameter evaluated as either an integral or decimal number matching the context of the function call. Contains the divisor value, or the value to be divided into <i>Dividend</i> .

Supported Return Types

- Integral Number
- Decimal Number

@PARSE_FORMATTED_DECIMAL

The PARSE_FORMATTED_DECIMAL function converts the given string into a decimal. It takes up to two optional parameters that are used for deciphering the format of the number stored in the string. The first parameter is the value to be converted and is required. This parameter is evaluated as a string, and a decimal is created from it using the optional parameters or the client's locale. This string must be a valid representation of a decimal.

Parameters

@PARSE_FORMATTED_DECIMAL (String [, Decimal Point, Thousands Separator])	
String	Required string parameter, contains the value to be parsed interpreted as a string and parsed into a decimal by the function.
Decimal Point	Optional string parameter with a default value of the client's locale decimal separator. This value may be set to any single character to be considered as a decimal separator.
Thousands Separator	Optional string parameter with a default value of the client's locale thousands separator. This value may be set to any single character to be considered as separating thousand and hundreds, millions and hundred thousands, etc.

Supported Return Types

Decimal

@PERCENT

The PERCENT function takes one decimal number parameter, divides that parameter by 100, and returns the result. This function ignores any significant digit rules during the division process.

Parameters

@PERCENT (Dividend)	
Dividend	Required decimal number parameter; contains the value to be divided by 100.

Supported Return Types

Decimal Number

@PRECISION

The PRECISION function takes a single decimal number parameter and returns its precision. This is the number of digits after the decimal. The function supports string, integral number and decimal number return types for this precision value. The parameter to it is always evaluated as a decimal number.

Parameters

@PRECISION (Decimal)	
Decimal	Required decimal number parameter; contains the value whose precision is to be returned by the function, i.e. the number of digits after the decimal.

Supported Return Types

- Integral Number
- Decimal Number
- String

@PROD

The PROD function takes two or more numeric parameters, multiplies the values, and returns the product. The parameters to the function are evaluated as either decimal or integral numbers, matching the context of the function call.

Parameters

@PROD (Number 1 [, ..., Number N])	
Number 1	Required parameter; evaluated as either an integral or decimal number matching the context of the function call. Contains the value to be multiplied with all other function parameters.
Number N	Optional parameter(s); evaluated as either integral or decimal number(s) matching the context of the function call. Contains the value(s) to be multiplied with all other function parameters.

Supported Return Types

- Integral Number
- Decimal Number

@RANGE_LIMIT

The RANGE_LIMIT function constrains a given numeric parameter to within a range of values, returning a value that is no greater than or less than a given set of upper and lower limits. The function takes three parameters: the value to be constrained; a lower limit; and an upper limit. If the value to be constrained is greater than the lower limit and less than the upper limit, the function will return the value. If this value is less than the lower limit, the value of the lower limit will be returned. If the constrained value is greater than the upper limit, the upper limit value will be returned. Each of the three parameters are evaluated as either decimal or integral numbers, matching the context of the function call.

Parameters

@RANGE_LIMIT (Constrain, Limit 1, Limit 2)	
Constrain	Required parameter; evaluated as either an integral or decimal number matching the context of the function call. Contains the value to be constrained by the function. This value will be returned if it is between the values of the Limit 1 and Limit 2 parameters.
Limit 1	Required parameter; evaluated as either an integral or decimal number matching the context of the function call. Contains the first limiting value in the range to which Constrain will be limited. This may be either the minimum or maximum value to which the return value is to be constrained.
Limit 2	Required parameter; evaluated as either an integral or decimal number matching the context of the function call. Contains the second limiting value in the range to which Constrain will be limited. This may be either the minimum or maximum value to which the return value is to be constrained.

Supported Return Types

- Integral Number
- Decimal Number

@ROUND

The ROUND function will round a numeric value to the specified precision. The first parameter to the function is the numeric value to be rounded. The second parameter is always evaluated as an integral number, and specifies the number of digits before or after the decimal

place to which the number should be rounded. This function may be called in an integral or decimal context.

Parameters

@ROUND (Number To Round, Precision)	
Number To Round	Required parameter; evaluated as either an integral or decimal number matching the context of the function call. Contains the value to be rounded by the function.
Precision	Required integral number parameter; contains the value specifying the precision to which <code>Number To Round</code> should be rounded. Positive numbers indicate a precision after the decimal place. Negative numbers indicate position before the decimal. This parameter can be only negative numbers when the function is called in an integral number context.

Supported Return Types

- Integral Number
- Decimal Number

@SIGN

The SIGN function takes a single decimal number parameter and returns -1, 0, or 1 if the parameter is negative, zero, or positive in value, respectively. The function may be called in an integral or decimal number context. The parameter is always evaluated as a decimal number.

Parameters

@SIGN (Number)	
Number	Required number parameter; evaluated as either a decimal or integral number based on context. Contains the value whose sign is to be determined by the function.

Supported Return Types

- Integral Number
- Decimal Number

@SIGNIFICANT_DIGITS

The SIGNIFICANT_DIGITS function takes a single parameter evaluated as a decimal number, and returns the number of significant digits it contains.

Parameters

@SIGNIFICANT_DIGITS (Decimal)	
Decimal	Required decimal number parameter; contains the value whose number of significant digits is to be calculated by the function.

Supported Return Types

- Integral Number
- Decimal Number
- String

@SQRT

The SQRT function returns the square root of a decimal number to the specified precision. The number for which the square root is found is evaluated as a decimal number, though whole integral values can be provided. The precision of the result is specified by an integral number.

Parameters

@SQRT (Decimal, Precision)	
Decimal	Required decimal number parameter; contains the value for which the square root will be found.
Precision	Required integral number parameter; contains the value specifying the precision to which the square root value will be calculated. The function will round the result to this precision. If a property is specified for the <code>Decimal</code> argument, the definition of that property's rounding behavior will be applied. A <code>Precision</code> value of 0 indicates no digits after the decimal, and a negative value indicates digits before the decimal.

Supported Return Types

Decimal Number

@SUM

The SUM function takes one or more numeric parameters, adds the values of each, and returns the result. This function may be called in an integral number, decimal number, or string context. The data type of each parameter in the numeric contexts will match that context. In a

string context, the function will treat its parameters as decimal numbers, though integral numbers may be provided.

Parameters

@SUM (Number 1 [, ..., Number N])	
Number 1	Required parameter; evaluated as either an integral or decimal number, depending on the context of the function call. Contains the first value to be summed with all other parameters to the function.
Number N	Optional parameter(s); evaluated as either an integral or decimal number, depending on the context of the function call. Contains the additional value(s) to be summed with all other parameters to the function.

Supported Return Types

- Integral Number
- Decimal Number
- String

@TOTAL

The TOTAL function will add the values of a given object property together for all instances of the object in a specified collection property. Optional criteria may be specified to include only specific objects within the collection being processed.

This function takes three parameters. The first is the object collection property, and the second is the object property whose value is to be summed in each object instance. An optional third parameter can be provided to include only certain object instances within the collection in this operation. This function supports the integral number and decimal number return types.

The object property values are totalled by the function based on the function's context, not the data type of the object property definition. This is important to note when the context is in an integral number, and the properties to be totaled are decimals. In this situation, the fractional portion of the properties will be truncated from the values prior to being added together.

Parameters

@TOTAL (Object Collection, [Child Property [, Include Criteria]])	
Object Collection	Required object collection property parameter; contains the object instances whose property values will be totaled by the function.
Child Property	Optional property parameter; specifies the object property to be totaled in each object instance in <code>Object Collection</code> . These property values will be evaluated as the data type of the function's context, which may be either integral number or decimal number.
Include Criteria	Optional Boolean parameter; this term is evaluated once for, and in the context of each object instance in <code>Object Collection</code> . The function will process each object instance for which this term returns true, and exclude those for which it returns false. If this parameter is not provided, all object instances will be processed by the function.

Supported Return Types

- Integral Number
- Decimal Number

@TRUNC

The TRUNC function will truncate the given numeric value to the specified position either before or after the decimal. The first parameter to the function is the value to be truncated and will be evaluated as either a decimal or integral number, matching the context of the function call. The second parameter is evaluated as an integral number and specifies the precision, or number of digits to which the value should be truncated. A positive precision value counts the number of digits to the right of the decimal, while a negative precision counts them to the left of the decimal. Note that this function differs from the ROUND function in that no rounding occurs. The number is truncated to the specified precision with no rounding behavior.

Parameters

@TRUNC (Number [, Precision])	
Number	Required parameter; evaluated as either an integral or decimal number matching the context of the function call. Contains the value to be truncated by the function.
Precision	Optional integral number parameter; contains the value specifying the precision of the resulting truncation. Positive values indicate the number of places to the right of the decimal, while negative values indicate the number of places to the left.

Supported Return Types

- Integral Number
- Decimal Number

Property Functions for Rules

The Property category of rule function terms available within the rule definition provide functions that are specific to working with properties, and in most cases are intended for use with properties of a given type. While they may take any property as a parameter, the resulting behavior of the function may not be desired if the selected property is not of the type for which the function is intended.

Many of the functions within this category are intended for use with collection properties. Most of these are named to reflect this, beginning with the word COLLECTION, e.g. @COLLECTION_FIND. Exceptions to this are COUNT and SIZE, which both return the total number of objects in a collection property, optionally counting only those that meet some criteria.

Another sub-set of the functions within this category are intended for use with properties of type external data. External data properties are defined to reference files stored on the client device from within the mobile application. The functions within this category that are intended for this property type each begin with the value FILE in their names, e.g. @FILE_PATH.

Other functions within this category include those that work with the user interface, returning values or names from the screens and fields of the currently displayed screen or screen set on the client. Each of these functions begins with the value SCREEN or SCREENSET in their names, e.g. @SCREENFIELDVALUE.

@COLLECTION_FIND

The `COLLECTION_FIND` function searches a given collection for the first member for which the given second parameter returns true. This second parameter is evaluated once for each member of the collection in a Boolean context. When a member of the search collection is found, that member is returned to the caller of the function. If no member of the collection results in the search criteria returning true, an empty property of the type within the search collection is returned.

Parameters

@COLLECTION_FIND (Collection Property, Search Criteria)	
Collection Property	Required property parameter; the property referenced for this parameter is assumed to be a collection and contains the members to be searched by the function using the specified <code>Search Criteria</code> parameter.
Search Criteria	Required Boolean parameter; this term is evaluated once for, and in the context of each member of <code>Collection Property</code> . The member of <code>Collection Property</code> returned by the function will be the first one for which <code>Search Criteria</code> returns true.

Supported Return Types

Property

@COLLECTION_FIND_BY_DEC

The `COLLECTION_FIND_BY_DEC` function searches a collection property for the first member to match a specified decimal value. For object collection properties, the property within each object of the collection to compare to the search value is also specified. This function will return the first member within the specified collection property found to match the provided search decimal value. If no member of the collection matches the search value, an empty instance of the member type is returned to the function caller.

Parameters

@COLLECTION_FIND_BY_DEC (Collection Property, Search Decimal [, Search Property])	
Collection Property	Required property parameter; this parameter is assumed to be a collection property. References the collection the function will search.
Search Decimal	Required decimal number parameter; contains the value to search for within Collection Property.
Search Property	Optional property parameter; when Collection Property contains object instances SearchProperty, specifies the property within that object type to compare against Search Decimal. The value of the property specified for this parameter is converted from the data type of the property to a decimal number for comparison to Search Decimal.

Supported Return Types

Property

@COLLECTION_FIND_BY_NUM

The COLLECTION_FIND_BY_NUM function searches a collection property for the first member to match a specified integral value. For object collection properties, the property within each object of the collection to compare to the search value is also specified. This function will return the first member within the specified collection property found to match the provided search integral value. If no member of the collection matches the search value, an empty instance of the member type is returned to the function caller.

Parameters

@COLLECTION_FIND_BY_NUM (Collection Property, Search Integral [, Search Property])	
Collection Property	Required property parameter; this parameter is assumed to be a collection property. References the collection the function will search.

@COLLECTION_FIND_BY_NUM (Collection Property, Search Integral [, Search Property])	
Search Integral	Required integral number parameter; contains the value to search for within <code>Collection Property</code> .
Search Property	Optional property parameter; when <code>CollectionProperty</code> contains object instances <code>Search Property</code> , specifies the property within that object type to compare against <code>Search Integral</code> . The value of the property specified for this parameter is converted from the data type of the property to an integral number for comparison to <code>Search Integral</code> .

Supported Return Types
Property

@COLLECTION_FIND_BY_STR

The `COLLECTION_FIND_BY_STR` function searches a collection property for the first member to match a specified string value. For object collection properties, the property within each object of the collection to compare to the search value is also specified. This function will return the first member within the specified collection property found to match the provided search string value. If no member of the collection matches the search value, an empty instance of the member type is returned to the function caller.

Parameters

@COLLECTION_FIND_BY_STR (Collection Property, Search String [, Search Property])	
Collection Property	Required property parameter; this parameter is assumed to be a collection property. References the collection the function will search.
Search String	Required string parameter; contains the value to search for within <code>Collection Property</code> .

@COLLECTION_FIND_BY_STR (Collection Property, Search String [, Search Property])	
Search Property	Optional property parameter; when <code>Collection Property</code> contains object instances <code>Search Property</code> , specifies the property within that object type to compare against <code>Search String</code> . The value of the property specified for this parameter is converted from the data type of the property to a string for comparison to <code>Search String</code> .

Supported Return Types

Property

@COLLECTION_MAX

The `COLLECTION_MAX` function searches an object collection for the object instance whose designated property contains the largest value of all members of the collection and then returns that maximum value. The function takes parameters for the object collection to be searched, the property within the object type of the collection whose value is to be compared between object instances, and optionally a rule term containing the criteria specifying which objects to search and which to exclude.

The optional Boolean parameter to the function is evaluated once for each object instance contained in the specified collection. This term is evaluated in the context of each object instance. The function will then compare only those objects for which this rule term returns true, with those for which it returns false being excluded from the comparison.

The data type of the property to be compared should be one for which a value comparison makes sense. While a minimum or maximum value is readily apparent in a primitive data type such as an integer, such a comparison makes little sense for a signature or external data property type. For property data types like the latter, the return value is undefined. The data type of the property to be compared in each function should be considered in relation to the data type of the function's context. Though the function supports the integral number, decimal number, and string return types, the conversion from the property's data type to the return type should be "type safe." Specifically, if the designated property to compare in each object is a string, the function should not be called in an integral or decimal number context.

Parameters

@COLLECTION_MAX (Object Collection, Child Property [, Include Criteria])	
Object Collection	Required object collection property parameter; specifies the collection to be processed by the function.
Child Property	Required property parameter; specifies the property whose value will be compared in each object instance in <code>Object Collection</code> . The data type of this property specifies the type of comparison made between the values for each object instance.
Include Criteria	Optional Boolean parameter; this term is evaluated once for, and in the context of each object instance in <code>Object Collection</code> . The function will compare the <code>Child Property</code> value of each object instance for which <code>Include Criteria</code> returns true, and excludes from this processing each object for which false is returned. If this parameter is omitted, all object instances in the collection will be processed.

Supported Return Types

- Integral Number
- Decimal Number
- String

@COLLECTION_MIN

The `COLLECTION_MIN` function searches an object collection for the object instance whose designated property contains the smallest value of all members of the collection and then returns that minimum value. The function takes parameters for the object collection to be searched, the property within the object type of the collection whose value is to be compared between object instances, and optionally a rule term containing the criteria specifying which objects to search and which to exclude.

The optional Boolean parameter to the function is evaluated once for each object instance contained in the specified collection. This term is evaluated in the context of each object instance. The function will then compare only those objects for which this rule term returns true, with those for which it returns false being excluded from the comparison.

The data type of the property to be compared should be one for which a value comparison makes sense. While a minimum or maximum value is readily apparent in a primitive data type

such as an integer, such a comparison makes little sense for a signature or external data property type. For such property data types as the latter, the return value is undefined.

The data type of the property to be compared in each function should be considered in relation to the data type of the function's context. Though the function supports the integral number, decimal number, and string return types, the conversion from the property's data type to the return type should be "type safe." Specifically, if the designated property to compare in each object is a string, the function should not be called in an integral or decimal number context.

Parameters

@COLLECTION_MIN (Object Collection, Child Property [, Include Criteria])	
Object Collection	Required object collection property parameter; specifies the collection to be processed by the function.
Child Property	Required property parameter; specifies the property whose value will be compared in each object instance in <code>Object Collection</code> . The data type of this property specifies the type of comparison made between the values of each object.
Include Criteria	Optional Boolean parameter; this term is evaluated once for, and in the context of each object instance in <code>Object Collection</code> . The function will compare the <code>Child Property</code> value of each object instance for which <code>Include Criteria</code> returns true and exclude from this processing each object for which false is returned. If this parameter is omitted, all object instances in the collection will be processed.

Supported Return Types

- Integral Number
- Decimal Number
- String

@COUNT

The COUNT function returns the number of object instances in a given collection property and, optionally, can count only those where some condition is true, returning this count value or a Boolean indication of whether any objects within the collection were counted. The function takes two parameters. The first parameter is required and is the object collection property to be counted. The second parameter is an optional Boolean parameter that is evaluated once for each object in the collection. It is used to include only certain members of

the collection in those counted. Specifically, each object for which the second parameter evaluates to true will be counted, and those for which it evaluates to false will not be counted.

Parameters

@COUNT (Object Collection [, Include Criteria])	
Object Collection	Required property parameter; contains the object collection to be counted by the function.
Include Criteria	Optional Boolean parameter; this term is evaluated once for, and in the context of each object instance in <code>Object Collection</code> . The function will count each object instance for which <code>Include Criteria</code> returns true and exclude from the count each object for which false is returned.

Supported Return Types

- Boolean
- Integral Number
- String

@CURRENTVALUE

The `CURRENTVALUE` function takes a variable number of property parameters. Each is evaluated in the context of the parameter that precedes it until either an invalid property is found or the last parameter is evaluated. The value of this property, or of a field on the current detail screen that targets the property, is returned.

The function can take a single property parameter, evaluated in the context of the function call. Either this property value is returned by the function, or the value of a detail screen field targeting that property on the current screen is returned. When multiple parameters are specified, it is assumed these parameters are either objects or object collections. Each subsequent parameter should be a child member of the parameter that precedes it in the function call. Otherwise, the parameter will be treated as an invalid property.

Parameters

@CURRENTVALUE (Property 1 [, ..., Property N])	
Property 1	Required property parameter; references the first property evaluated in the context of the function call. This parameter should reference a property that is a child member of the definition setting the context of the function call.
Property N	Optional property parameter(s); references the next property evaluated by the function in the context of the property that immediately precedes it in the function's parameter list. This should be a child member of that parameter.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String
- Property

@FILE_CHANGED

The FILE_CHANGED function is provided to work with the external data property type. It evaluates the file referenced by such a property and returns true if the file has been modified since it was downloaded to the client device or attached locally. Though the data type of this function's single parameter is string, the function expects an external data property and will return false for any other value provided.

Parameters

@FILE_CHANGED (File Name)	
File Name	Required string parameter; specifies the file to be checked by the function for modifications. Though the data type of this parameter is string, the intended usage of this function is that this parameter is an external data property. Specifying any other string value for this parameter will result in the function always returning false.

Supported Return Types

Boolean

@FILE_EXTENSION

The FILE_EXTENSION function takes a single parameter, normally an external data property, and returns the file extension referenced by that property as a string. This function is intended for use with external data properties as a part of the attached document's functionality. If the external data property provided as a parameter does not currently reference a file, the function will return an empty string. The return value from this function is undefined when the parameter provided is not an external data property.

Parameters

@FILE_EXTENSION (File Name)	
File Name	Required string parameter; specifies the file whose file extension is to be returned. While the data type of this parameter is a string, the intended parameter is a property of data type external data. The file extension of the file referenced by such a property is returned by this function. If the property does not reference a file, the function returns an empty string.

Supported Return Types

String

@FILE_NAME

The FILE_NAME function takes a single parameter, normally an external data property, and returns the name and extension of the file referenced by that property. This function is intended for use with external data properties as a part of the attached document's functionality. If the external data property provided as a parameter does not currently reference a file, the function will return an empty string. The return value from this function is undefined when the parameter provided is not an external data property

Parameters

@FILE_NAME (File Name)	
File Name	Required string parameter; specifies the file whose name and extension is to be returned. While the data type of this parameter is a string, the intended parameter is a property of data type external data. The name and file extension of the file referenced by such a property is returned by this function. If the property does not reference a file, the function returns an empty string.

Supported Return Types

String

@FILE_PATH

The FILE_PATH function takes a single parameter, normally an external data property, and returns the full path of the location where the file referenced by that property is stored. This function is intended for use with external data properties as a part of the attached document's functionality. If the external data property provided as a parameter does not currently reference a file, the function will return an empty string. The return value from this function is undefined when the parameter provided is not an external data property.

Parameters

@FILE_PATH (File Name)	
File Name	Required string parameter; specifies the file whose full path is to be returned. While the data type of this parameter is a string, the intended parameter is a property of data type external data. The full path to the location of the file referenced by such a property is returned by this function. If the property does not reference a file, the function returns an empty string.

Supported Return Types

String

@FILE_PATH_AND_NAME

The FILE_PATH_AND_NAME function takes a single parameter, normally an external data property, and returns the full path and name of the file referenced by that property. This function is intended for use with external data properties as a part of the attached document's functionality. If the external data property provided as a parameter does not currently reference a file, the function will return an empty string. The return value from this function is undefined when the parameter provided is not an external data property.

Parameters

@FILE_PATH_AND_NAME (File Name)	
File Name	Required string parameter; specifies the file whose full path and name is to be returned. While the data type of this parameter is a string, the intended parameter is a property of data type external data. The full path and name of the file referenced by such a property is returned by this function. If the property does not reference a file, the function returns an empty string.

Supported Return Types

String

@FILE_SIZE

The FILE_SIZE function takes a single parameter, normally an external data property, and returns the size in bytes of the file referenced by that property. This function is intended for use with external data properties as a part of the attached document's functionality. If the external data property provided as a parameter does not currently reference a file, the function will return an empty string. The return value from this function is undefined when the parameter provided is not an external data property.

Parameters

@FILE_SIZE (File Name)	
File Name	Required string parameter; specifies the file whose size is to be returned. While the data type of this parameter is a string, the intended parameter is a property of data type external data. The size of the file referenced by such a property is returned by this function. If the property does not reference a file, the function returns 0.

Supported Return Types

Integral Number

@IS_SPECIAL_VALUE

The IS_SPECIAL_VALUE function returns a Boolean value indicating whether or not a specified property is currently set to its special value. The function returns true when the property value is equal to its defined special value. It returns false when it is not equal to the defined special value, if no special value is defined for the property, or if the property type does not include special value attributes.

The function can take one or more parameters to allow for the navigation through the object data structure of a module, beginning with a child of the object setting the context for the function call, and drilling down into this structure to the descendent object and, finally, a property of that object.

Parameters

@IS_SPECIAL_VALUE (Property 1 [, ..., Property N])	
Property 1	Required property parameter; if the only parameter to the function, this property's value will be compared to its special value attribute settings. If additional parameters are provided, the function will evaluate the next parameter in the context of this one, assuming it is a child member of this property.
Property N	Optional property parameter(s); with each specifying an object further down in the data hierarchy of the module. Each subsequent <i>Property N</i> provided must be a child to the parameter that immediately precedes it in the function call. The last parameter provided is compared to its special value attribute settings.

Supported Return Types

Boolean

@IS_VALID_DECIMAL_NUMBER

The IS_VALID_DECIMAL_NUMBER function takes a single parameter and returns true if the value is a valid decimal. If the value returned when evaluating this parameter as a decimal number is NaN (Not a Number), the function will return false.

Parameters

@IS_VALID_DECIMAL_NUMBER (Decimal)	
Decimal	Required decimal number parameter; contains the value to be evaluated as a valid decimal number.

Supported Return Types

Boolean

@LASTSCANVALUE

The LASTSCANVALUE returns the last value scanned by the client device and processed by the mobile application. If called prior to a value being scanned in or on a device without scanning capabilities, the function will return an empty string. The last scanned value will be returned regardless of where or when the function is called. Exiting and restarting the mobile application will remove the scanned value, and the function will return null until a new value is scanned.

Parameters

@LASTSCANVALUE()	
	This function takes no parameters.

Supported Return Types

String

@MEMBER

The MEMBER function is used to search an object collection and to return the value of a property within that collection that matches the given search value. The first match found within the collection will be returned. The function takes two parameters; the first is an object collection, and the second is a property with a value that is to be located. The object instance with the same property name, data type, and value within the object collection is then found and the value of the object's key property is returned. If no match is found within the collection, the function returns the null equivalent for the context's data type.

Parameters

@MEMBER (Collection Property, Search Property)	
Collection Property	Required property parameter; references the object collection to be searched by the function.
Search Property	Required property parameter; references the property definition for which an exact match within the Collection Property object instances is to be searched.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String
- Property

@NEEDS_XMIT

The NEEDS_XMIT function takes one or more property parameters, expects each to be an object, and returns true if the last parameter is an object and that object or one of its descendent objects in the module data structure has a pending transaction. If multiple parameters are provided to the function, the first parameter is evaluated in the context of the function call.

Each subsequent parameter is evaluated in the context of the parameter that precedes it. If any parameter evaluates to any definition instance other than an object, the evaluation ends and the function returns false.

In most current implementations of this function, a single parameter is provided that is a target path to an object instance selected using the target browser. If a pending transaction targets this object or any of its descendent objects, this function will return true. Otherwise it returns false.

Parameters

@NEEDS_XMIT (Property 1 [, ..., Property N])	
Property 1	Required property parameter; evaluated by the function to determine if it is an object instance first, and if a pending transaction exists that targets this object or any of its descendent object instances second. In current implementations and uses for this function, this is the only parameter provided in most cases.
Property N	Optional property parameters; each is evaluated in the context of the parameter before it in the function's parameter list. These parameters are expected to evaluate to an object instance. The last parameter in this list is evaluated by the function for pending transactions targeting it or any of its descendent object instances.

Supported Return Types

Boolean

@SCREENFIELDVALUE

The SCREENFIELDVALUE function returns the current value of a field on the current detail screen. The name of the field whose value is desired is the only parameter to this function. This name value is the internal name that uniquely identifies the field definition within the parent detail screen.

Parameters

@SCREENFIELDVALUE (Field Name)	
Field Name	Required string parameter; contains the name of the detail screen field definition whose current value is to be returned. This is the internal name of the field definition.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String

@SCREENFIELDNAME

The SCREENFIELDNAME function is supported only in update rules and returns the name of the current detail screen field being updated by the rule within which the function is called. If this function is called in a rule not used in an update rule, its return value is undefined. The function takes no parameters.

Parameters

@SCREENFIELDNAME()	
	This function takes no parameters.

Supported Return Types

String

@SCREENNAME

The SCREENNAME function returns the name of the screen definition with the focus at the time of its evaluation. This function takes no parameters.

Parameters

@SCREENNAME ()	
	This function takes no parameters.

Supported Return Types

String

@SCREENSETNAME

The SCREENSETNAME function returns the name of the current screen set displayed to the user. The value returned is the internal definition name of the screen set that uniquely identifies the screen set definition within the module. This function takes no parameters and supports the string return type. The current screen set is the parent definition to the screen with the current focus.

Parameters

@SCREENSETNAME ()	
	This function takes no parameters.

Supported Return Types

String

@SIZE

The SIZE function returns the number of object instances in a given collection property and, optionally, can count only those where some condition is true, returning this count value or a Boolean indication of whether any objects within the collection were counted. The function takes two parameters. The first parameter is required and is the object collection property to be counted. The second parameter is an optional Boolean parameter that is evaluated once for each object in the collection. It is used to include only certain members of the collection in those counted. Specifically, each object for which the second parameter evaluates to true will be counted, and those for which it evaluates to false will not be.

Parameters

@SIZE (Object Collection [, Include Criteria])	
Object Collection	Required property parameter; contains the object collection to be counted by the function.
Include Criteria	Optional Boolean parameter; this term is evaluated once for and in the context of each object instance in <code>Object Collection</code> . The function will count each object instance for which <code>Include Criteria</code> returns true and exclude from the count each object for which false is returned.

Supported Return Types

- Boolean
- Integral Number
- String

@TRANSACTIONPROPERTYNAME

The TRANSACTIONPROPERTYNAME function returns the name of the transaction property for which the rule is being evaluated. This function supports the string return type. The value returned is the internal definition name of the transaction property. This function is

only supported when part of a rule being evaluated is an initial value rule for a transaction property. The return from this function in any other context is undefined.

Parameters

@TRANSACTIONPROPERTYNAME ()	
	This function takes no parameters.

Supported Return Types

String

@TYPE

The TYPE function returns the definition type of the last parameter it evaluates. This function takes one or more parameters, each evaluated as a property and each evaluated in the context of the parameter before it in the function's parameter list. The function supports the integral number, Boolean, and string return types.

In a Boolean context, the function will return true if the last parameter evaluated exists in the current context, or false if it does not exist. In a string context, this function returns the name of the definition type, e.g. object, transaction, etc. In an integral number context, this function returns the internal identifier for that definition type.

In current implementations this function has limited usage and may be deprecated in a future release.

Parameters

@TYPE (Property 1 [, ..., Property N])	
Property 1	Required property parameter; this is the first parameter evaluated by the function in the context of the function call. This parameter should be a child member of the definition setting the function's context. If it is not, the function will return false or null depending on the data type of the context. If this is the only parameter, and it exists, the return will be either true or the identifier for the definition type, depending on context.

@TYPE (Property 1 [, ..., Property N])	
Property N	Optional property parameter(s); each evaluated in the context of the parameter before it and assumed to be a child member of that previous property. Each additional <i>Property N</i> parameter will set the context for the next in the list. The last <i>Property N</i> parameter will be evaluated for its definition type, and the function will then return true or the identifier for this type if it exists, or else false or null if it does not exist.

Supported Return Types

- Boolean
- Integral Number
- String

@UI

This function has been deprecated and will not be supported in future releases. It should be replaced in all existing rule definitions with SCREENFIELDVALUE. The UI function takes a single parameter that is the field index for the currently displayed screen and returns the value of that field. The field index is a 0-based index. The parameter must be between 0 and the total number of field definitions on the current screen minus 1.

The value of the specified field is evaluated in the context of the function call. The UI function supports the Boolean, integral number, decimal number, string, and property return types.

Parameters

@UI (Field Index)	
Field Index	Required integral number parameter; specifies the field on the current detail screen by its index whose current value is to be returned. This is a 0-based index, with the first field at index 0 and the last field at the index position equal to the total number of fields on the screen minus 1. Index numbering occurs from left to right and top to bottom.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String

- Property

String Functions for Rules

The String category of rule function terms available within the rule definition provide the string parsing, concatenation, conversion, and other related behaviors for manipulating string values. These functions include those that return or remove sub-strings from source strings, convert strings to all upper or lowercase, trim whitespace, or return formatting characters, including new lines and tabs.

@CONCATENATE

The CONCATENATE function's behavior has changed with the release of Agentry 5.1. The function now takes two or more string parameters and concatenates them together, returning the resulting string. Previously this function took two strings and an integral number, with this third (and now deprecated) parameter limiting the length of the string returned by the function. This same behavior can now be provided by returning the result from CONCATENATE to the LEFT string function.

Parameters

@CONCATENATE(String 1, [, ..., String N])	
String 1	Required string parameter; contains the string value to which the remaining parameter(s) will be appended.
String N	Optional string parameter(s); each containing a string value to be appended to the resulting string returned by the function.

Supported Return Types

String

@FIND

The FIND function searches a source string for a provided sub-string and, depending on the context of the function call, returns one of the sub-string, the position of its first character within the source string, or an indicator as to whether or not it was located. This search can optionally be case-insensitive and may begin at the beginning of the source string or at some character position within the source string.

Note that this function no longer works with object collection properties. This functionality is now provided by the new rule function COLLECTION_FIND. Upgrades of projects from previous platform versions will be modified with the replacement of FIND with COLLECTION_FIND in any rule definition. This will happen as a part of the standard

upgrade process built into the Agentry Editor and should require no additional actions on the part of the developer.

When FIND is called in a string context, the function will search a source string for a sub-string, returning that sub-string when found or an empty string if not found.

When FIND is called in an integral number context, the function will search a source string for a sub-string, returning the position of the first character of the sub-string within the source string when found, or -1 if not found. The first character of the string is at position 0.

When FIND is called in a Boolean context, the function will search a source string for a sub-string, returning true when found and false when not found.

Parameters

@FIND (Source String, Search String [, Case Sensitive [, Start Position]])	
Source String	Required string parameter; contains the string value to be searched by the function.
Search String	Required string parameter; contains the string value to search for within Source String. If Search String contains more characters than Source String, no sub-string will be found.
Case Sensitive	Optional Boolean parameter; when specified, determines whether or not the search should consider or ignore case. When this parameter is true or not present, the search will be case-sensitive. When this parameter is false, case will be ignored.
Start Position	Optional integral number parameter; when specified, contains the character position within Source String to begin the search for Search String. The first character in Source String is at position 0. If this parameter is not provided, the search will always begin with the first character of Source String.

Supported Return Types

- Boolean
- Integral Number
- String

@LEFT

The LEFT function returns a sub-string from a specified source string, with the length of the sub-string specified and beginning with the left-most character in the string. This function

takes two parameters: a source string and the number of characters to extract from the source. If this length is equal to or greater than the length of the source string, the entire string is returned. If the length is 0 or less, an empty string is returned.

Parameters

@LEFT (Source String, Length)	
Source String	Required string parameter; contains the source string from which a sub-string will be extracted.
Length	Optional integral number parameter; contains the maximum number of characters to extract from <code>Source String</code> . If this value is equal to or longer than <code>Source String</code> , the entire string is returned. If <code>Length</code> is 0 or less, an empty string will be returned.

Supported Return Types

String

@LENGTH

The `LENGTH` function determines the length of the given string and based on context, returns either the number of characters in the string, or an indicator of whether or not the string is empty. In an integral context, the number of characters is returned as an integer. In a string context, the number of characters is returned as a string. In a Boolean context, true is returned if the source contains at least one character, and false is returned when the string is empty.

Parameters

@LENGTH (Source String)	
Source String	Required string parameter; contains the source string whose length will be determined.

Supported Return Types

- Boolean
- Integral Number
- String

@LOWERCASE

The LOWERCASE function converts all alphabetical characters in the source string to lower case and returns the result. Any non-alphabetical characters are unchanged.

Parameters

@LOWERCASE (Source String)	
Source String	Required string parameter; contains the string value to be converted to lowercase.

Supported Return Types

String

@MID

The MID function parses a source string to return a sub-string that begins at a specified position and contains at most the specified number of characters. The first parameter to the function is the source string from which the sub-string is extracted. The second and third parameters are optional and specify the start and end position within the source string from which the sub-string is to be extracted. The first character in the source string is at position 0. The default starting position is 0 if not provided. If the number of characters to return is not provided, the default is the remaining length of the source string after the start position.

Parameters

@MID (Source String [, Start Position [, Max Length]])	
Source String	Required string parameter; contains the source string from which the sub-string will be extracted.
Start Position	Optional integral number parameter; contains the zero-based position of the first character within <i>Source String</i> for the sub-string to be extracted. If this parameter is not provided, the default start position is 0 and the entire <i>Source String</i> value will be returned.
Max Length	Optional integral number parameter; contains the maximum number of characters to return from <i>Source String</i> after <i>Start Position</i> . If this parameter is not provided, all characters after <i>Start Position</i> will be returned as the sub-string.

Supported Return Types

String

@NEWLINE

The NEWLINE function returns the command characters <CR> <LF>, (0x0D 0x0A), which result in a Windows new line. The return value of this function can be concatenated with other strings for formatting purposes.

Parameters

@NEWLINE()	
	This function takes no parameters.

Supported Return Types

String

@REMOVE

The REMOVE function searches a given source string, removes all instances of a provided search string, and returns the result. As optional behaviors, parameters can be provided to specify whether or not the search is case-sensitive, and to specify the starting position within the source string to begin the search.

Parameters

@REMOVE (Source String, Search String [, Case Sensitive [, Start Position]])	
Source String	Required string parameter; contains the source string to be searched by the function.
Search String	Required string parameter; contains the sub-string to be removed from Source String.
Case Sensitive	Optional Boolean parameter; when provided, indicates whether or not the search of Source String for Search String should be case-sensitive. If this value is true or not specified, the search is case-sensitive. If set to false, case is ignored.

@REMOVE (Source String, Search String [, Case Sensitive [, Start Position]])	
Start Position	Optional integral number parameter; when provided, specifies the zero-based position within <code>SourceString</code> to begin the search. The default is to begin at position 0.

Supported Return Types

String

@REPLACE

The REPLACE function searches a given source string for a provided search string and replaces each instance of the search string with a replacement string. By default, this search is case-sensitive and includes the entire source string. Both of these behaviors can be overridden based on optional parameters to the function.

Parameters

@REPLACE(Source String, Search String, Replace String [, Case Sensitive [, Start Position]])	
Source String	Required string parameter; contains the source string to be searched by the function.
Search String	Required string parameter; contains the string value to be searched for within <code>Source String</code> .
Replace String	Required string parameter; contains the string value to replace <code>Search String</code> within <code>Source String</code> .
Case Sensitive	Optional Boolean parameter; when specified, indicates whether the search should be case-sensitive. When true or if not provided, the search is case-sensitive. When false, case is ignored.

@REPLACE(Source String, Search String, Replace String [, Case Sensitive [, Start Position]])	
Start Position	Optional integral number parameter; when specified, indicates the zero-based position within <code>Source String</code> to begin the search. The default is to search the entire <code>Source String</code> . If <code>Start Position</code> is less than 0 or is greater than the number of characters in <code>Source String</code> , an empty string will be returned.

Supported Return Types

String

@RFIND

The RFIND function searches a string for a sub-string of characters beginning with the right-most character in a string and based on the function's context, returns the sub-string when found, the position of the first character of the sub-string within the source string, or an indicator of whether the sub-string was found. This search can optionally be case-insensitive and may begin at the right-most character of the source string, or somewhere within the source string by specifying the first position, counting from the left, to begin searching.

When RFIND is called in a string context, the function will search a source string for a sub-string, returning that sub-string when found, or an empty string if not found.

When RFIND is called in an integral number context, the function will search a source string for a sub-string, returning the position of the first character of the sub-string within the source string when found, or -1 if not found. The left-most character of the source string is at position 0.

When RFIND is called in a Boolean context, the function will search a source string for a sub-string, returning true when found, or false if not found.

Parameters

@RFIND (Source String, Search String [, Case Sensitive [, Start Position]])	
Source String	Required string parameter; contains the string value to be searched by the function.

@RFIND (Source String, Search String [, Case Sensitive [, Start Position]])	
Search String	Required string parameter; contains the string value to search for within <code>Source String</code> .
Case Sensitive	Optional Boolean parameter; when provided, specifies whether or not the search of <code>Source String</code> for the <code>Search String</code> value should be case-sensitive. When this value is true or not provided, the search is case-sensitive. If this value is false, case is ignored.
Start Position	Optional integral number parameter; when provided, specifies the zero-based character position, counting from the left, within <code>Source String</code> to begin the search. If this value is not provided, the search begins at the right-most character within <code>Source String</code> . If this value is 0 or negative, an empty string is returned. If this value is equal to or greater than the total number of characters within <code>Source String</code> , the entire string is searched.

Supported Return Types

- Boolean
- Integral Number
- String

@RIGHT

The `RIGHT` function returns a sub-string of specified length from a given source string, beginning at the right-most character of the source string and counting back towards the beginning. The function takes two parameters. The first is the source string from which the sub-string is extracted. The second is the number of characters in the sub-string. If the specified number of characters is greater than the length of the source string, the entire source string is returned. If the specified number of characters specified is 0 or less, an empty string is returned.

Parameters

@RIGHT (Source, MaxLength)	
Source String	Required string parameter; contains the string value from which the sub-string will be extracted.

@RIGHT (Source, MaxLength)	
Max Length	Required integral number parameter; contains the maximum number of characters to return from <code>Source String</code> . If this value is greater than the number of characters in <code>Source String</code> , <code>Source String</code> is returned in its entirety. If <code>Max Length</code> is 0 or negative, an empty string will be returned.

Supported Return Types

String

@TAB

The TAB function takes no parameters and returns a tab <HT> character (0x09). This function is most often used to insert a tab into a text value for the purposes of formatting.

Parameters

@TAB()	
	This function takes no parameters.

Supported Return Types

String

@TRIM

The TRIM function removes any leading or trailing whitespace characters from a given source string. The following are considered whitespace characters and will be removed from the beginning and end of the given source string:

- Horizontal tab
- Vertical tabs
- Space
- Newline
- Carriage return
- Formfeed

Parameters

@TRIM (Source String)	
Source String	Required string parameter; contains the value from which all leading and trailing whitespace will be removed.

Supported Return Types

String

@UPPERCASE

The UPPERCASE function converts all alphabetical characters in a given source string to upper case and returns the result. Any non-alphabetical characters are returned unchanged.

Parameters

@UPPERCASE (Source String)	
Source String	Required string parameter; contains the string value to be converted to upper case by the function.

Supported Return Types

String

System Functions for Rules

The System function category of rule functions available in the rule definition provide functionality related mostly to accessing values from the mobile application as a whole, those retrieved by interacting with the client device or specific hardware components of the device, or by interacting directly with the client device's operating system.

These functions include those that return date and time values, interact with a client device's GPS system, return on-line state information about the mobile application, and other similar items.

@DATE

The DATE function returns either the current system date of the client device, or the date value specified as a parameter to the function. The function can also take an optional format parameter when called in a string context. The function supports the integral number, decimal number, and string return types.

When called in an integral or decimal number context, the DATE function will return the number of days before (negative number) or after (positive number) the date January 1, 2001.

This is the Agentry epoch date. When called in a string context the function will return the date value in the default format of the client device's locale.

The function's first parameter, if provided, is a string and contains the date value to be returned by the function. The second parameter is also optional and is evaluated as a string. It can contain one or more of the following date format tokens, which will be used to then format the date value returned by the function. Note that this format parameter is ignored when the function is called in any context other than string:

Table 1. Rule Date Format Tokens - *All date format tokens are case sensitive*

Date Token	Description
d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month with leading zero for single-digit days.
ddd	Day of week as three letter abbreviation. The function uses the LOCALE_SAB-BREVDAYNAME value associated with the device's specified locale.
dddd	Day of week as its full name. The function uses the LOCALE_SDAYNAME value associated with the device's specified locale
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as three letter abbreviation. The function uses the LOCALE_SAB-BREVMONTHNAME value associated with the device's specified locale.
MMMM	Month as its full name. The function uses the LOCAL_SMONTHNAME value associated with device's specified locale.
y	Year as last two digits with no leading zero for years less than 10.
yy	Year as last two digits with leading zero for years less than 10.
yyyy	Full four digit year value.
non-token characters	Any non-token character within the format string is passed through as is; e.g in the following string: d={MM/dd/yyyy} the resulting string will contain the slash characters separating each date element: 11/17/1967

Parameters

@DATE ([Date String [, Format Tokens]])	
Date String	Optional string parameter; contains the date value to be returned by the function. To format the current system date, this parameter may be set to a second call to the DATE function. A data definition such as a property or global of type Date or Date and Time may be used for this parameter. The time portion of the value will be truncated. A string may be used, provided the date is in the format MM/dd/yyyy. A numeric value may be passed for this parameter, in which case it will be treated as the number of days before or after the Agentry epoch date.
Format Tokens	Optional string parameter; contains the date format tokens that will format the function's return value when called in a string context. When the function is called in any other context, this parameter is ignored. A DateString must be specified before FormatTokens can be provided.

Supported Return Types

- Integral Number
- Decimal Number
- String

@DATE_AND_TIME

The DATE_AND_TIME function returns either the current system date and time of the client device, or the date and time value specified as an optional parameter to the function. The function can also take an optional format parameter when called in a string context. The function supports the integral number, decimal number, and string return types.

When called in an integral or decimal number context, the function will return the number of seconds before (negative value) or after (positive value) the date and time of January 1, 2001 - 12:00:01 AM. This is the Agentry epoch date and time. When called in a string context, the function will return a date and time value in the default format for the client device's locale.

The function's first parameter, if provided, is a string and contains the date and time value to be returned by the function.

The second parameter is also optional and is evaluated as a string. It can contain one or more of the date and time format tokens. The syntax for this parameter is as follows:

```
d={date format tokens} t={time format tokens}
```

The tokens within the curly braces will be used to format the date and time value returned by the function:

Table 2. Rule Date Format Tokens - *All date format tokens are case sensitive*

Date Token	Description
d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month with leading zero for single-digit days.
ddd	Day of week as three letter abbreviation. The function uses the LOCALE_SAB-BREVDAYNAME value associated with the device's specified locale.
dddd	Day of week as its full name. The function uses the LOCALE_SDAYNAME value associated with the device's specified locale
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as three letter abbreviation. The function uses the LOCALE_SAB-BREVMONTHNAME value associated with the device's specified locale.
MMMM	Month as its full name. The function uses the LOCAL_SMONTHNAME value associated with device's specified locale.
y	Year as last two digits with no leading zero for years less than 10.
yy	Year as last two digits with leading zero for years less than 10.
yyyy	Full four digit year value.
non-token characters	Any non-token character within the format string is passed through as is; e.g in the following string: d={MM/dd/yyyy} the resulting string will contain the slash characters separating each date element: 11/17/1967

Table 3. Rule Time Format Tokens - *All time format tokens are case sensitive*

Time Token	Description
h	Hour of day in 12 hour clock format with no leading zero for single digit hours.
hh	Hour of day in 12 hour clock format with leading zero for single digit hours.
H	Hour of day in 24 hour clock format with no leading zero for single digit hours.
HH	Hour of day in 24 hour clock format with leading zero for single digit hours.
m	Minute of the hour with no leading zero for single digit minutes.
mm	Minute of the hour with leading zero for single digit minutes.

Time Token	Description
s	Seconds of the minute with no leading zero for single digit minutes.
ss	Seconds of the minute with leading zero for single digit minutes.
t	One character time marker string, such as A or P.
tt	Two character time marker string, such as AM or PM.
non-token characters	Any non-token character within the format string is passed through as is; e.g in the following string: <code>t={ hh:mm:ss }</code> the resulting string will contain the colon characters separating each time element: <code>10:12:32</code>

Parameters

@DATE_AND_TIME ([Date Time String [, Format Tokens]])	
Date Time String	Optional string parameter; contains the date and time value to be returned by the function. To format the current system date and time, this parameter may be set to a second call to the DATE_AND_TIME function. A data definition such as a property or global of type Date, Date and Time, or Time may be used for this parameter. A string may be used, provided the date and time is in the format <code>hh:mm:ss MM/dd/yyyy</code> . A numeric value may be passed for this parameter, in which case it will be treated as the number of seconds before or after the Agentry epoch date and time.
Format Tokens	Optional string parameter; contains the date and time format tokens that will format the function's return value when called in a string context. When the function is called in any other context, this parameter is ignored. A Date Time String must be specified before Format Tokens can be provided.

Supported Return Types

- Integral Number
- Decimal Number
- String

@DISTANCE_MILES

The DISTANCE_MILES function takes two GPS location parameters and returns the total distance between them in miles as a decimal number. The distance returned is always 0 or a

positive number. The function may return an invalid decimal value (NaN) if either of the two GPS location parameters to the function are invalid location values.

This function is intended for use on devices equipped with a GPS unit, though it will return a distance in miles for any two valid GPS location values.

Parameters

@DISTANCE_MILES (GPS Location 1, GPS Location 2)	
GPS Location 1	Required location parameter; contains one of the two GPS location values between which the distance will be calculated by the function.
GPS Location 2	Required location parameter; contains one of the two GPS location values between which the distance will be calculated by the function.

Supported Return Types

Decimal Number

@DISTANCE_KILOMETERS

The DISTANCE_KILOMETERS function takes two GPS location parameters and returns the total distance between them in kilometers as a decimal number. The distance returned is always 0 or a positive number. The function may return an invalid decimal value (NaN) if either of the two GPS location parameters to the function are invalid.

This function is intended for use on devices equipped with a GPS unit, though it will return a distance in kilometers for any two valid GPS location values.

Parameters

@DISTANCE_KILOMETERS (GPS Location1, GPS Location2)	
GPS Location 1	Required location parameter; contains one of the two GPS location values between which the distance will be calculated by the function.
GPS Location 2	Required location parameter; contains one of the two GPS location values between which the distance will be calculated by the function.

Supported Return Types

Decimal Number

@GPS_LOCATION

The GPS_LOCATION function returns the GPS location of the device's current position, optionally based on a maximum age for the GPS data. This function can take a single optional parameter of type integral number treated as the maximum number of seconds for the GPS data. This function will return an invalid location value if the client device is not equipped with a GPS unit or if the GPS unit is not accessible to the client application.

If the maximum age of the GPS data available to the function exceeds the maximum age parameter provided to the function, the function will query the GPS unit for a current location. The function will set all components of the location data type, including the location value itself, as well as the number of satellites and precision as reported by the GPS unit.

Parameters

@GPS_LOCATION ([Max Age])	
Max Age	Optional integral number parameter; specifies the maximum age in seconds of the GPS data for which a location should be returned. If this parameter is not provided, the function will return the most recent location.

Supported Return Types

Location

@IS_VALID_LOCATION

The IS_VALID_LOCATION function takes a single GPS location parameter and returns true or false based on whether or not the value of the parameter is a valid location value. If this parameter is a GPS location property type, the definition of that property's precision attributes will be used as a part of determining the parameters validity. Empty location values are always invalid.

Parameters

@IS_VALID_LOCATION (Location)	
Location	Required location parameter; contains the GPS location value to be checked for validity. If the value of this parameter is valid, the function will return true. If this parameter is a location property type, the property's precision attributes will be used to determine if the value is valid.

Supported Return Types

Boolean

@JAVASCRIPT

The JAVASCRIPT function is provided to allow JavaScript logic to be embedded within a rule definition. Included in the JAVASCRIPT function's behavior is the ability to pass values to the script being processed from the rule definition. The value returned by the JavaScript logic will be the value returned by the function. This function takes one required parameter, and as many additional parameters as needed to pass in additional values. The supported return types of the JAVASCRIPT function are Boolean, integral number, decimal number, or string.

The first parameter to the JAVASCRIPT function is the JavaScript logic to be processed. This parameter may be any string value from any source within the application. In most cases, it is recommended that the rule term JavaScript Text is used, which is an available item within the rule editor. The main purpose for this term is that it provides a large text box control to display multiple lines of text, making it easier to write and edit JavaScript logic.

Additional parameters to the JAVASCRIPT parameter are referenced within the JavaScript logic through the zero-based array `argv[]`. This array is available in all JavaScript logic processed by the JAVASCRIPT rule function. The second parameter to JAVASCRIPT is stored in the `argv[0]` element, the third is in `argv[1]`, and so on.

The data types of the additional parameters are strings. The parameters will be converted to this data type and will be passed to the JavaScript as such. These values can then be converted to a different data type where necessary within the JavaScript logic.

The JavaScript engine used to process the script logic is SpiderMonkey. Note that the Data Object Model (DOM) and the `XMLHttpRequest` object are not implemented as a part of this JavaScript support.

The usage of JavaScript within rule definitions is intended to be supplemental functionality. It is not recommended that all rules be written exclusively with JavaScript, as the processing of such script files is less efficient than the processing of rule definitions. The main intent is to allow a developer to implement certain pieces of logic using JavaScript wherever it is deemed appropriate to do so.

Parameters

@JAVASCRIPT (JavaScript [, ..., ArgV String N])	
JavaScript	Required string parameter; contains the JavaScript logic to be processed by the JavaScript engine. In most cases, this logic will be contained in the special rule term JavaScript text, though any data term that may be safely converted to a string and that contains valid JavaScript may be used.
ArgV String N	Optional string parameter(s); contains value(s) passed to the JavaScript and available in the <code>argv[]</code> array. The value of the second JAVASCRIPT parameter, i.e. <code>ArgV String 1</code> , is available within the array element <code>argv[0]</code> , the next optional JAVASCRIPT parameter's value is stored in <code>argv[1]</code> , and so on. The members of this array are strings and should be converted within the JavaScript logic where necessary.

Supported Return Types

- Boolean
- Integral Number
- Decimal Number
- String

@LATITUDE

The LATITUDE function returns the latitude of a provided GPS location in degrees. The function takes a single location parameter from which the latitude portion of the coordinates is returned. This function will return an invalid decimal value (NaN) if the GPS location parameter is invalid.

This function is primarily intended for use on client devices equipped with GPS units, though it will return a latitude for any valid GPS location value provided. It does not interact with the GPS unit directly.

Parameters

@LATITUDE (GPS Location)	
GPS Location	Required location parameter; specifies the GPS location from which the latitude will be calculated. If this is not a valid GPS location value, the function will return an invalid decimal value (NaN).

Supported Return Types

Decimal Number

@LOCATION

The LOCATION function takes two decimal number parameters, treated as degrees of latitude and longitude, and returns the GPS location for those two values. The function may return an invalid GPS location if either the latitude or longitude parameters are invalid values.

Longitude values must be in the range -179 and 180, inclusive. Latitude values must be in the range of -90 and 90, inclusive. The returned location value includes the GPS location, a satellite count of 50, and a precision of 1.0.

This function is intended for use on devices equipped with a GPS unit, though it will return a GPS location for a given valid set of longitude and latitude values.

Parameters

@LOCATION (Latitude, Longitude)	
Latitude	Required decimal number parameter; provides the latitude in degrees of the location to be returned as a GPS location by the function. Valid latitude values are in degrees with a range of -90 to 90, inclusive.
Longitude	Required decimal number parameter; provides the longitude in degrees of the location to be returned as a GPS location by the function. Valid longitude values are in degrees in the range -179 to 180, inclusive.

Supported Return Types

Location

@LONGITUDE

The LONGITUDE function returns the longitude for a given GPS location. The function takes a single location parameter from which the longitude portion of the coordinates is returned. This function will return an invalid decimal value (NaN) if the GPS location parameter is invalid.

This function is primarily intended for use with devices equipped with a GPS unit, though it will return a longitude value for any valid GPS location value provided. It does not interact with the GPS unit directly.

Parameters

@LONGITUDE (GPS Location)	
GPS Location	Required location parameter; contains the GPS location from which the longitude in degrees is calculated by the function.

Supported Return Types

Location

@MODULE_ENABLED

The MODULE_ENABLED function returns a Boolean value indicating whether or not the module specified by name is enabled or disabled. The function will return true if the module is enabled. It will return false if the module is disabled or is not present. The function takes a single string parameter containing the name of the module definition to be checked.

Parameters

@MODULE_ENABLED (Module Name)	
Module Name	Required string parameter; contains the definition name of the module to be checked by the function.

Supported Return Types

Boolean

@OFFLINE

The OFFLINE function returns a value indicating whether or not the client application is in an off-line state. The function supports the Boolean and integral number return types. In a Boolean context, this function will return true if the client application is in an off-line state, and false if it is in an on-line state. In an integral number context, the function will return a non-zero value if the client application is in an off-line state, and zero if in an on-line state.

Parameters

@OFFLINE ()	
	This function takes no parameters.

Supported Return Types

- Boolean
- Integral Number

@TIME

The TIME function returns either the current system time of the client device, or the time value specified as an optional parameter to the function. The function can also take an optional format parameter when called in a string context. The function supports the integral number, decimal number, and string return types.

When called in an integral or decimal number context, the function will return the number of seconds before (negative value) or after (positive value) the time 12:00:00 AM. This is the Agentry epoch time. When called in a string context, the function will return a time value in the default format for the client device's locale.

The function's first parameter, if provided, is a string and contains the time value to be returned by the function.

The second parameter is also optional and is evaluated as a string. It can contain one or more of the time format tokens. The syntax for this parameter is as follows:

```
t={time format tokens}
```

The tokens within the curly braces will be used to format the time value returned by the function:

Table 4. Rule Time Format Tokens - *All time format tokens are case sensitive*

Time Token	Description
h	Hour of day in 12 hour clock format with no leading zero for single digit hours.
hh	Hour of day in 12 hour clock format with leading zero for single digit hours.
H	Hour of day in 24 hour clock format with no leading zero for single digit hours.
HH	Hour of day in 24 hour clock format with leading zero for single digit hours.
m	Minute of the hour with no leading zero for single digit minutes.
mm	Minute of the hour with leading zero for single digit minutes.
s	Seconds of the minute with no leading zero for single digit minutes.
ss	Seconds of the minute with leading zero for single digit minutes.
t	One character time marker string, such as A or P.

Time Token	Description
tt	Two character time marker string, such as AM or PM.
non-token characters	Any non-token character within the format string is passed through as is; e.g in the following string: <code>t={ hh:mm:ss }</code> the resulting string will contain the colon characters separating each time element: <code>10:12:32</code>

Parameters

@TIME ([Time String [, Format Tokens]])	
Time String	Optional string parameter; contains the time value to be returned by the function. To format the current system time, this parameter may be set to a second call to the TIME function. A data definition such as a property or global of type Time or Date and Time may be used for this parameter. The date portion of a Date and Time value will be truncated. A string may be used provided the time is in the format <code>hh:mm:ss</code> . A numeric value may be passed for this parameter, in which case it will be treated as the number of seconds before or after the Agentry epoch time.
Format Tokens	Optional string parameter; contains the time format tokens that will format the function's return value when called in a string context. When the function is called in any other context, this parameter is ignored. The <code>Time String</code> parameter must be specified before <code>Format Tokens</code> can be provided.

Supported Return Types

- Integral Number
- Decimal Number
- String

@TIME_TICKS

The `TIME_TICKS` function returns the number of milliseconds since the client device booted, excluding any time the device was in sleep or hibernation modes, or any similar modes of operation. The function supports the integral number, decimal number, and string return types.

The difference between the return values of two separate calls to this function can be used to calculate duration values for various purposes.

Parameters

@TIME_TICKS ()	
	This function takes no parameters.

Supported Return Types

- Integral Number
- Decimal Number
- String

@USERID

The USERID function returns the user ID value entered to log into the client application. This value is returned as a string.

Parameters

@USERID ()	
	This function takes no parameters.

Supported Return Types

String

Table Functions for Rules

The Table functions category of rule functions available within the rule definition provide access to the complex tables and data tables of an application. This category consists of three functions. The first two return a record from a complex table or data table based on some search criteria. The third returns the total number of records within a complex table.

@COMPLEXTABLE

The COMPLEXTABLE function searches the specified complex table for a single record and returns a single field from that record. The parameters to this function include, at a minimum, the name of the complex table to be searched and the value to search for within the records. If only these two values are provided, the function will search the complex table using the table's primary index, returning the field upon which the primary index has been defined from the record found.

As optional parameters to the function, the index to search upon and the field to return from the matching record can be specified by passing in the definition names of each. While optional, these parameters are provided in most use cases.

An additional variation on the parameters passed to the function is in the situation when a search index is specified, and that index is a child to another index within the table. In this

scenario, additional search values must also be provided to the function. The requirement is for each index, starting with the one specified up to the top-level index (one that has no parent index) in the structure, there must also be a corresponding search value provided to the function.

As an example, assume a complex table with three indexes defined: A, B, and C. Index C is a child index to B, and B in turn is a child index to A, which is a top-level index in the table. When searching this table with the COMPLEXTABLE function, if index C is specified as the search index, search values for indexes A, B, and C must be provided. During the search, the function will begin by finding records that match on index A, then within that set those records that match index B, and finally within that sub-set the first record that matches index C.

Parameters

@COMPLEXTABLE (Table Name, Search Value, [Parent Search Value N,] [Search Index, Return Field])	
Table Name	Required string parameter; contains the name of the complex table to be searched by the function.
Search Value	Required string parameter; contains the value used to search the complex table records.
Parent Search Value N	String parameter(s); required and specified only when Search Index is a child index. One Parent Search Value must then be specified for each index above the Search Index in the index hierarchy.
Search Index	Optional string parameter; provides the name of the index within the complex table used by the function to locate the desired record based on the Search Value and possibly Parent Search Values. If this parameter is not provided, the primary index of the complex table is used by the function.
Return Field	Optional string parameter; provides the name of the complex table field whose value is returned from the record found by the function. If this parameter is not specified, the field for which the primary index of the complex table is defined is the default field value returned.

Supported Return Types

- Boolean

- Integral Number
- Decimal Number
- String

@TABLE

The TABLE function searches a data table for a record with the specified key value and returns the value field for the matching record. The search performed by this function is a record by record search, attempting to match the provided search value with the key field of each record. Data tables contain no indexes or guaranteed record order, and therefore the search is performed in a first to last manner. Data tables with large numbers of records will take longer to search, both by the TABLE function as well as any other methods. The value field returned is converted to the data type of the function's context, which may be integral number, Boolean, or string.

Parameters

@TABLE (Table Name, Search String)	
Table Name	Required string parameter; contains the name of the data table the function will search.
Search String	Required string parameter; contains the key value the function will use to search the data table records.

Supported Return Types

- Boolean
- Integral Number
- String

@TABLE_COUNT

The TABLE_COUNT function takes a single string parameter that is assumed to be the name of a data table within the application. The function counts the number of records in the named table and returns the result. This function supports the integral number return type.

Parameters

@TABLE_COUNT (Table Name)	
Table Name	Required string parameter; contains the name of the data table whose total number of records is to be returned. If this parameter contains a name that does not match any defined data tables, the function will return zero.

Supported Return Types

Integral Number

Syclo Data Markup Language

When synchronizing data between the mobile application and the back end system, it is necessary to have access to the mobile application's data values. This access is provided in Agentry using the Syclo Data Markup Language, or SDML. The SDML is a markup language consisting of tags that provide access to the data values of the mobile application. Additionally, the SDML includes a full set of functions, or function tags, that can be used to perform logical operations in relation to this values or to drive the overall logic the Agentry Server will execute against the back end system.

The SDML tags used during synchronization are a part of the text within the scripts for step definitions defined for SQL Database, HTTP-XML, and File system connection types. Also, the synchronization components of data tables and complex tables for each of these system connection types can contain SDML. In addition to SQL Step definitions, other .sql script files run by the Server may also contain SDML tags. Steps defined for Java Virtual Machine system connections also include the ability to access SDML tags, but these tags may not be contained directly in the source code of the Java Steplet files used by these steps.

The Agentry Server will pre-process the script files of steps containing SDML markup. This processing is referred to as tag expansion. Each tag within the script is expanded, with the value it represents replacing the tag at the exact position of that tag within the file. Function tags are expanded with the results of their expansion being placed in the exact position of the function call within the file. Once the tag expansion has completed, the resulting text is submitted to the back end system for processing.

The two categories of tags within the SDML are data tags and function tags. Data tags represent data values available to the script file based on when it is executed. This information must be known when writing the script in which the SDML will be contained. For step definitions the values in scope are dictated by the step usage definition running them. For .sql scripts run by the server, but not a part of the step definition, the values in scope will vary depending on how that script is used. Certain values are globally available, such as the user ID as entered by the user to log into the Agentry Client.

Function tags are globally available, with certain exceptions. Function tags provide the logical, mathematical, string manipulation, and other similar functionality to the SDML. Function tags can take values passed in as arguments, parameters, or expressions. These values are processed by the function during tag expansion, with the resulting value of the function call being placed within the script.

Following is a basic example of a simple SQL statement containing SDML data tags:

```
SELECT
    A.FIELD1,
    B.FIELD2,
    C.FIELD3,
FROM
    TABLEA A,
    TABLEB B,
    TABLEC C
WHERE
    A.NAME = '<<user.agentryID>>' AND
    A.ACCTNUM = '<<object.acctnum>>' AND
    B.ACCTNUM = A.ACCTNUM AND
    C.ACCTNUM = B.ACCTNUM
```

In this example, the value `<<user.agentryID>>` is replaced with the user ID as entered when the user logged into the Agentry Client. The data tag `<<object.acctnum>>` will be replaced with the value of the `acctnum` property of the object currently being processed.

SDML Data Tags Overview

Data tags provide the access to the production data of the application within the synchronization components of the mobile application. This includes access to property values, global definition values, query constants, and client and server information system information. Each of these items just listed are referred to as the data tag's data source.

In addition to the tag's data source, all data tags also have a certain data type. The data source and the data type of a tag combine to give the data tag its overall behavior. This behavior includes how the value is expanded during data tag expansion, as well as the parameters that the tag will support. By and large, the data tags created for properties are the tags that have the most complex behavior.

Data tags that provide access to data other than from properties or globals are strings. Those data tags that are based on a property definition are one of several data types, based on the property data type.

Data Tag Data Types

Within the SDML, all data tags have a data type. This data type affects how the data tag is processed during tag expansion. Do not confuse the data tag's type with the data type for that same value in the back end system. When the Agentry Server has completed tag expansion the resulting values within the script are plain text. At this point, the methodology for denoting the value's data type will depend on the type of back end system in use.

As an example, for a database system connection, data tags with a data type of date and time will be expanded with the date and time conversion function for that database as a part of the text, as in:

```
<<object.statusDate>>
```

If this data tag is a date and time and used in a script for an Oracle database, it would expand to:

```
to_date('01/12/2004 14:23:45', 'mm/dd/yyyy hh24:mi:ss')
```

As you can see, the date and time value has been wrapped in the `to_date` function call for Oracle, which converts string values into dates and times.

The data types for data tags are as follows. Note that all data types other than string are applicable only to tags for global and property values:

- String
- Integral Number
- Decimal Number
- Boolean
- Date
- Time
- Date and Time
- Signature

The Scope of Data Tags

The data tags within the SDML may or may not be valid in one area verses another. The scope of a data tag will vary from one to the next. Certain tags are only valid in steps used by a fetch. Others are only available in steps used by transactions. Still others are available globally. It is important to note that for scripts within step definitions, the scope for a data tag is determined by the type of step definition and also the step usage definition referencing the step to be processed at run time. For example, the data tags that are in scope for a step used by an object read step will be different from the tags that are in scope for a step used by a transaction's server update step. The terms used to describe a data tag's scope are:

- **Global:** A data tag with Global scope is valid in all scripts processed by the Agentry Server. NOTE: Do not confuse the term Global used here to denote a tag's scope with the definition type global. Values for a global definition do have a global scope. However, there are other data tags that also are globally available to the application's synchronization components.
- **Definition-Type:** A data tag with the Definition-*Type* scope is one that is in scope only for a certain type of definition, such as a Transaction or Object. The Type portion of this scope specifies the definition type for which the data tag is applicable.
- **Definition:** A data tag with the Definition scope is one that is only in scope for instances of a specific definition. For example, the data tags in scope for an Object named Customer

will be different than those for an Object named Order. Data tags that have a Definition scope are those that provide access to the properties of an object.

<<user>> Data Tag Container

The user data tag is a container tag for several user-related values. Each of these values is represented by a member tag within the user container. Of these members, two contain members of their own. All members of the user data tag container are available in all scripts processed by the Agentry Server.

Table 5. <<user>> Data Tag Members

Tag Name	Description
<<user.name>>	Returns the name of the client user. By default this will be the ID entered by the user to log into the client. This value may be overridden.
<<user.deviceID>>	Returns the device ID for the client device upon which the Agentry Client is running. This value is set by the original equipment manufacturer.
<<user.agentryID>>	Returns the user ID entered to log into the Agentry Client. This value cannot be overridden during synchronization.
<<user.client>>	A data tag container within <<user>>. Member data tags provide information about the client application and client device as provided by the Agentry Client during synchronization.
<<user.info>>	A data tag container within <<user>>. Member data tags are variable and set during synchronization based on the logic of the mobile application.

<<user.client>> Data Tag Container

The <<user.client>> data tag container is a member of the <<user>> container. Members of <<user.client>> provide information about the client device's hardware and software, and information about the Agentry Client application.

Many of the member data tags of <<user.client>> are valid only on client devices running a Windows operating system. Such members names begin with the text `Win_` and will return empty strings for any other client device type. This members providing information about the Agentry Client software or the host system are invalid for web browser clients.

Table 6. <<user.client>> Member Data Tags

Tag Name	Description
<<user.client.time>>	Returns the date and time of the client device when the transmission between the Agentry Client and Server began.

Tag Name	Description
<<user.client.Language>>	Returns the two character abbreviation for the client device's configured locale.
<<user.client.Win_MajorVersion>>	Returns the major version number of the Windows operating system installed on the client device.
<<user.client.LocaleID>>	Returns the local ID for the configured locale of the client device.
<<user.client.Win_ServicePack>>	Returns the service pack installed on the Windows operating system on the client device.
<<user.client.timeDifference>>	Returns the difference in time's between the client device and the Agentry Server's host system. This value is represented in number of seconds where negative values indicate the client is behind the server.
<<user.client.Win_MinorVersion>>	Returns the minor version number of the Windows operating system installed on the client device.
<<user.client.FirstLogin>>	Returns the text value true or false based on whether the current transmit is the result of the user's initial login to the Agentry Client. This value is representative of the first time a user transmits from a given client device.
<<user.client.Platform>>	Returns the platform type of the client device.
<<user.client.timeZone>>	Returns the time zone to which the client device has been set.
<<user.client.timeZoneBias>>	Returns the difference in seconds between the client's time zone and Greenwich Mean Time (GMT).
<<user.client.screenHeight>>	Returns the height of the client device's screen in pixels.
<<user.client.screenWidth>>	Returns the width of the client device's screen in pixels.

Tag Name	Description
<<user.client.PreviousUser>>	Returns the text true or false based on whether the current synchronization processing is a part of a previous user transmit resulting from a user change on the Agentry Client.
<<user.client.Win_ComputerName>>	Returns the network name of the client device.
<<user.client.Win_PlatformID>>	Returns the platform ID of the client device. The specific value is dependent on the system and OEM settings.
<<user.client.Country>>	Returns the abbreviated country name for the client device, as indicated by the device's locale settings.
<<user.client.Win_OS>>	Returns the type of Windows operating system (e.g. Mobile, XP, etc.) of the client device.
<<user.client.clientTime>>	Returns the current time of the client device when the transmission began.
<<user.client.ClientVersion>>	Returns the Agentry Client executable's Agentry version number, such as 6.0.0.0.
<<user.client.Win_UserName>>	Returns the Windows account login under which the client device is currently running. This tag is only valid on Windows devices where an account name is entered. It will return an empty string for all other device types.
<<user.client.Win_BuildNumber>>	Returns the build number of the Windows operating system installed on the client device.
<<user.client.clientTimeZone>>	Returns the time zone configured on the client device.
<<user.client.WinCE_Platform>>	Returns the platform type of the client device. This value is valid only for client devices running a mobile version of the Windows operating system.

Tag Name	Description
<<user.client.TestEnvironmentVersion>>	Returns the version number of the Agentry Test Environment. This value is valid only when the client is the ATE. Returns an empty string for all other clients.
<<user.client.Win_ProcessorLevelCode>>	Returns the processor level code of the client device's processor. This value is dependent on the original equipment manufacturer.
<<user.client.Win_ProcessorRevision>>	Returns the processor revision of the client device's processor. This value is dependent on the original equipment manufacturer.
<<user.client.clientTimeZoneDifference>>	Returns the difference in seconds between the client's time zone and Greenwich Mean Time (GMT).
<<user.client.Win_ProcessorArchitecture>>	Returns the processor architecture of the client device's processor. This value is dependent on the original equipment manufacturer.
<<user.client.Win_ProcessorArchitectureID>>	Returns the processor architecture ID of the client device's processor. This value is dependent on the original equipment manufacturer.
<<user.client.Win_ProcessorCount>>	Returns the number of processors on the client device.
<<user.client.isDaylightSavings>>	Returns the text true or false based on whether or not the client device is currently in daylight savings time.
<<user.client.xmitConfigGroup>>	Returns the defined group of the transmit configuration definition selected by the user for the current transmission.
<<user.client.xmitConfigName>>	Returns the name of the transmit configuration definition selected by the user for the current transmission. This is the internal definition name.

<<user.info>> Data Tag Container

The `<<user.info>>` data tag container is a special data tag provided to allow for user-specific data to be captured at the beginning of the synchronization process and made available globally to all other synchronization processing. The members of this container tag are determined based on values returned from the back end system. The methodology for this depends on the type of system connection for the back end system.

Members within the `<<user.info>>` container are named when retrieved. Those values are then referenced using the syntax:

```
<<user.info.tagName>>
```

The members within this container tag are retrieved using either SQL queries run from the `SqlBE.ini` section `[UserInfo]`; or they are set via an HTTP-XML system connection's response mappings. Specifically, the response mappings within the validate user requests for this system connection type.

When creating these tags via a SQL Database system connection, the column in the return set of the query retrieving these values will be the name for the data tag. When creating these tags using the HTTP-XML system connection, a part of the response mapping definition is the attribute containing the tag's name.

For both system connection types, the tags available in the `<<user.info>>` container are set immediately following user validation and are available to all synchronization processing that takes place after this point.

When a tag is added to this container in one system connection it will be available to synchronization components for all other system connections.

<<server>> Data Tag Container

The `<<server>>` data tag container includes members that return values and information about the Agentry Server instance for the current transmission. Each of these tags will return values specific to the current Server instance for the current transmission.

Table 7. <<server>. Member Data Tags

Tag Name	Description
<code><<server.admin.name>></code>	Returns the value configured in the <code>agentry.ini</code> server configuration file section <code>[Server]</code> . The configuration option <code>administratorName</code> contains the value returned by this tag.
<code><<server.admin.phone>></code>	Returns the value configured in the <code>agentry.ini</code> server configuration file section <code>[Server]</code> . The configuration option <code>administratorPhone</code> contains the value returned by this tag.

Tag Name	Description
<<server.admin.email>>	Returns the value configured in the <code>agentry.ini</code> server configuration file section <code>[Server]</code> . The configuration option <code>administratorEmail</code> contains the value returned by this tag.
<<server.system-Name>>	Returns the value configured in the <code>agentry.ini</code> server configuration file section <code>[Server]</code> . The configuration option <code>systemName</code> contains the value returned by this tag. If this configuration option is not set or is not present, the default return from this tag is <code>Agentry Server</code> .
<<server.serial-Number>>	Returns the serial number entered when the Agentry Server was installed. This value will be unique for all Server instances in a multi-server production implementation.

Data Tags for Application Globals

The values of any application global definition can be returned using SDML data tags. The syntax for a global data tag is as follows:

```
<<groupName.globalName [length=n]>>
```

The `groupName` is the defined group for the global definition. The global name is internal definition name of the global. These tags will return the current value of the global definition. If the global value has been overridden the override value will be returned. References to global data tags in synchronization components processed by the Agentry Server prior to the global override processing will return the global's defined value.

All global data tags are strings, regardless of the data type of the global definition. Global data tags accept a single named parameter specifying the length of the string to return from the global. When a length is specified the data tag will return no more characters than specified in the length parameter, counting from the left. Any characters beyond the specified length will be truncated from the returned value.

Query Constants Files and Data Tags

Installed with the Agentry Server are two query constants files provided for use with SQL Database system connections: `Oracle_sd.ini` and `SqlServer_sd.ini`. Each is intended for use with the database type for which they are named. The contents of these files include a single configuration section, `[Database]`, within which are a set of configuration options listed as key and value pairs. Within each file exist the same keys. The values for these items are different in each file.

The purpose of these values is to provide support for applications which may synchronize with the same back end system, but which may be driven by different database types. These files support query reuse between these systems by providing expressions matching a given

vendors variation in support of the ANSI SQL and database-type specific functions. The contents of these files are listed next, with the value for each key listed for both files:

Table 8. Query Constant Files Keys and Values

key	Oracle_sd.ini Value	SqlServer_sd.ini Value
name	Oracle	SqlServer
getSystemTime	sysdate	getdate()
timeStampFormat	to_date('%m/%d/%Y %H:%M:%S', 'mm/dd/yyyy HH24:MI:SS')	'%m/%d/%Y %H:%M:%S'
dateFormat	to_date('%m/%d/%Y', 'mm/dd/yyyy')	%m/%d/%Y'
timeFormat	to_date('%H:%M:%S', 'HH24:MI:SS')	'%H:%M:%S'
tempdate	to_date('01/02/1901 12:00:00', 'mm/dd/yyyy HH24:MI:SS')	convert(DATETIME, '01/02/1901 12:00:00')
substring	substr	substring
stringcat		+
charFunction	chr	char
nullFunction	nvl	isNull
singleRow	from dual	<i>null</i>
unicodePrefix	N	N
terminalErrorCodes	00028;01001;01012;03113;03114; 12203;12500;12505;12535;12571	0;1;2;4;5;11;53
retryWithChangeErrorCodes	<i>null</i>	<i>null</i>
retryWithoutChangeErrorCodes	00060	00060

key	Oracle_sd.ini Value	SqlServer_sd.ini Value
fatalWithMessageErrorCodes	<i>null</i>	<i>null</i>
fatalWithoutMessageErrorCodes	<i>null</i>	<i>null</i>

Each of these values is specific to a database type. The key for each value is available using SDML data tags using the syntax:

```
<<database.keyName>>
```

These data tags will return the value as specified in the query constant file in use for the SQL Database system connection. The file used by a system connection is set in the [SQL-n] section of the `agentry.ini` file by setting the configuration option `queryConstantFiles`.

This file is processed by the Server at startup and data tags are created and made available to all SQL scripts processed by the Agentry Server.

Other values may be added to this file within other sections. The syntax for referencing these values is:

```
<<sectionName.keyName>>
```

Following is a description of each of these data tags intended purpose:

Tag Name	Description
<<database.name>>	Returns the name of the database type for which the file was created. This value should not be altered in the source file
<<database.getSystemTime>>	Returns the database-specific system date and time function.
<<database.timeStampFormat>>	Returns the database-specific date and time tokens used to format date and time values. This setting is used by the Agentry Server when expanding date and time data tags and should not be altered in the source file. This may be passed to the format parameter for date and time values within data tags.

Tag Name	Description
<<database.dateFormat>>	Returns the database-specific date tokens used to format date and time values. This setting is used by the Agentry Server when expanding date data tags and should not be altered in the source file. This may be passed to the format parameter for date and time values within data tags.
<<database.timeFormat>>	Returns the database-specific time tokens used to format date and time values. This setting is used by the Agentry Server when expanding time data tags and should not be altered in the source file. This may be passed to the format parameter for date and time values within data tags.
<<database.temptime>>	Returns the date and time of January 2, 1901, 12:00:01 am in the database-specific format for dates and times. This value can be used in synchronization when last update or other date and time values for data definitions contain invalid date and times. If a different date and time is necessary it can be altered in the query constants file.
<<database.substring>>	Returns the database-specific function for extracting a substring from a source string.
<<database.stringcat>>	Returns the database-specific character for concatenating string values.
<<database.charFunction>>	Returns the database-specific function for converting values to the character or VARCHAR data type.
<<database.nullFunction>>	Returns the database-specific function used to test values for null and optionally replace those values with a default.
<<database.singleRow>>	Returns the required FROM portion of a query to select values from nothing.
<<database.unicodePrefix>>	Returns the prefix to append to values to indicate they are encoded in unicode.

Password Data Tags

The following two data tags are available only in SQL Scripts run from the [ChangePassword] section of the `SqlBE.ini` configuration file of the Agentry Server. They contain the old and new passwords for a user when that user is performing a password change from the Agentry Client.

Tag Name	Description
<code>newPassword</code>	Returns the new password entered by the user when changing the password on the Agentry Client.
<code>oldPassword</code>	Returns the previous password being change by the user when changing the password on the Agentry Client.

Complex Table Data Tags

The synchronization components for complex tables have specific data tags available providing information about the definition and its current data. These include whether or not the table is in a rebuild state, the last update date and time indicating when the table was last synchronized on the client, and the name of the table. These tags are in addition to those that are globally available.

Tag	Description
<code><<name>></code>	Returns the internal name of the complex table definition as entered in the application project.
<code><<rebuild>></code>	Returns true or false, indicating whether or not the table is in a rebuild state. This is intended for use in synchronization to determine if the synchronization should retrieve only modifications to the table's data, or if all records for the table should be retrieved. Returns true when a change to the complex table definition has been published to the Agentry Server; if the synchronization logic includes the data tag <code><<user.agentryID>></code> and the value of that tag changes from the previous transmit; or if the tables force reload logic indicates the table should be in a rebuild state.
<code><<lastUp-date>></code>	Returns the date and time provided by the client when the complex table was last synchronized. By default this value is provided by the Agentry Server based on a query of the back end system. However, synchronization of the complex table should include retrieving this value with the table's data. The latest date and time retrieved during that process will be used as the complex table's last update value. This tag supports the use of the named parameter <code>format</code> to format the time and date using date and time tokens.

Data Table Data Tags

The synchronization components for data tables have specific data tags available providing information about the definition and its current data. These include the last update date and time indicating when the table was last synchronized on the client, and the name of the table. These tags are in addition to those that are globally available.

Tag	Description
<<name>>	Returns the internal name of the data table definition as entered in the application project.
<<lastUp-date>>	Returns the date and time provided by the client when the data table was last synchronized. By default this value is provided by the Agentry Server based on a query of the back end system. However, synchronization of the data table should include retrieving this value with the table's data. The latest date and time retrieved during that process will be used as the data table's last update value. This tag supports the use of the named parameter <code>format</code> to format the time and date using date and time tokens.

Property Data Tags Overview

The data tags for property values within the application have unique, additional behaviors to the other data tags within Agentry. There are common items to all property data tags, including access to raw values and indicator of the property value in relation to its defined special value. There are also behaviors for data tags specific to the data type of the property the tag represents. Another aspect unique to property data tags is their scope. The property values available within the SDML will vary depending on which synchronization definition is referencing the tag.

The first item to be aware of with property data tags is that they are only available in step definitions. The specific properties in scope for a step will depend on which step usage definition is running the step during synchronization. No property data tags are available to any other synchronization component beyond the step definitions of a module.

Property Data Tag Parameters

Data tags for properties, regardless of data type, support two parameters specific to properties. These are the `.isSpecial` and `.raw` parameters. The syntax for these parameters is as follows:

```
<<object.propertyName.isSpecial>>
<<object.propertyName.raw>>
```

The `.isSpecial` parameter returns a true or false value indicating whether or not the current property value is equal to the property's defined special value. True indicates it is equal

to the special value. False will be returned when the property value is anything other than the special value or if the property does not have a special value.

The `.raw` parameter is available to all property data tags regardless of data type, though its exact behavior will be data type-specific. The purpose of the `.raw` parameter is to return the value of the property without any formatting of the data. By default many of the property data tags will return the value in a formatted manner befitting the data type of the property. As an example, string properties are automatically dequoted during expansion. If the `.raw` parameter is used, the value will not be dequoted during expansion. Other data types have different behaviors related to the formatting and therefore the value returned by `.raw` will be different for each data type.

Data Tags for Fetch Client Exchange and Server Exchange Steps

Steps run by fetch definitions will have access to all properties defined for the fetch. Object key properties for the object instances in the collection targeted by the fetch may also be available depending on how the step usage definition's **Run** attribute setting.

The following lists describe the data tags in scope for each of the fetch step usage definition **Run** attribute settings.

Table 9. Run Attribute: Run One Time

Tag	Description
<code><<collectionName>></code>	This tag returns the collection targeted by the fetch. This tag may be passed to the <code><<foreach...>></code> function tag to iterate over the object instances within the collection. For each object instance, the tags available include the key property of the object type, and the last update (<code><<lastUpdate>></code>) value of each object.
Either: <ul style="list-style-type: none"> <code><<fetch.propertyName>></code> <code><<fetchName.propertyName>></code> 	Any properties defined within the fetch definition are available using the syntax shown.
<code><<fetch.messageNumber>></code>	This data tag returns the fetch's message number as recorded in the <code>messages.log</code> file generated by the Agentry Server. This is typically used for debugging and similar purposes.

Table 10. Run Attribute: Run Once per Object

Tag	Description
<<object.keyPropertyName>>	This tag returns the key property of the object instance currently being processed by the step.
<<lastUpdate>>	This tag returns the last update value of the object instance currently being processed by the step.
<<fetch.messageNumber>>	This data tag returns the fetch's message number as recorded in the messages.log file generated by the Agentry Server. This is typically used for debugging and similar purposes.

Data Tags for Transaction Step Usage Definitions

All step usage definitions within transactions include the same data tags within their scope. Following is a list of these data tags:

Tag	Description
<<timestamp>>	Returns the date and time when the transaction was applied on the Agentry Client. This value is obtained from the client device.
<<transaction.propertyName>>	Returns the value of the transaction property, <i>propertyName</i> . All properties within a transaction are available via data tags.
<<objectName.keyPropertyName>>	Returns the key property of the object targeted by the transaction. The object name must be used in this syntax, as the generic <i>object</i> designation is not valid in this context.
<<transaction.messageNumber>>	Returns the value of the transaction's message number as recorded in the messages.log file generated by the Agentry Server. Typically used for debugging and similar purposes.

Property Data Tags for Push Step Usage Definitions

Steps run as push retrieval and push removal steps are either once per poll period or once per user per poll period. For either run setting, these steps do not have access to object properties and therefore have no available property data tags.

For a given poll, push read steps can be run once, once per user, once per object, or once per collection object. When run once per object, the steps will be able to use property data tags to access the key property of any object for the target collection created by the push retrieval steps or previous push read steps. When run once per collection object, the data tag for that child

object type's key property will be available to the step. The child objects in the collection must have been defined before the step that needs to reference the key property.

Push response and error steps are both always run once per object. Therefore these steps have access to the key property of the object for which they are run.

Property Data Tags for Service Event Step Usage Definitions

The step usage definitions for service events include read steps, data state steps, update steps, and error handling steps.

Service event read steps can be defined to run once or to run once per object. When defined to run once, the read step will have access to the collection created by the service event's synchronization components.

When defined to run once per object, the read steps the property data tags for the object type will be available. These will expand to the property value of the object instance currently be processed by the error handling step. For all of these values the syntax of the SDML property data tag is `<<object.propertyName>>`.

Data state steps and update steps within service events are always run once per object and will have access to all property values of the object instance being processed.

Error handling steps can be defined to run one time or run once per object. When defined to run one time, the error handling steps will have access to the object collection being synchronized by the service event. When defined to run once per object, the property data tags for the object type will be available. These will expand to the property value of the object instance currently be processed by the error handling step. For all of these values, the syntax of the SDML property data tag is `<<object.propertyName>>`.

Property Data Tags for Object Read Steps

Object read steps are run as a part of downstream synchronization that may occur for various synchronization processes, including fetch, push, service event, and transaction processing. Also, the read step itself may be defined to run one time or run once per object. Both of these aspects of an object read step can impact the property data tags available to the step.

When the object read step is run after a fetch, any fetch properties will be in scope for the object read step. These must be referenced as `<<fetch.propertyName>>`.

For all four situations, the read step will be run either in the context of a target collection, for a specific object instance, or for an instance of an object in a collection property of the object definition that contains the read step, based on the run attribute.

When defined to run one time the object read step will have access to the collection targeted by the fetch, push, or service event run immediately before the object read steps. If run one time the object read steps will have access to the key property and last update values for the object currently being processed. If defined to run once per collection object, the properties of the object collection property available to the read step include the key property of the child object, as well as the key property of the step's parent object. In this case, both objects must be

referenced by their definition names, as in: `<<customer.customerID>>`, `<<order.orderID>>`.

Data Tags and Property Data Types

The data tags to access property values are different from other data tags. The basics of there use are the same as all data tags. However, data tags for properties include additional parameters to access the property values and those parameters depend upon the data type of the property they are referencing.

Boolean

A Boolean property will result in a Boolean data tag. Like their property counterparts, Boolean data tags are either true or false. When passed as an argument to a function tag, there are special syntactical rules that apply to Boolean data tags.

When data tag expansion occurs, a Boolean data tag will result in either the text “true” or “false.” Because of this fact, when a Boolean data tag is used as a parameter to a function tag, it should not be enclosed in markers (`<<` and `>>`). When a Boolean is not enclosed in these markers, the value of either true or false is passed to the function, rather than the text values of “true” or “false”. This is important since, if the text values are passed to a function that is expecting a Boolean value, it will always consider the value passed in to be true. Remember that true is a value and false is the absence of a value. The text “false” is a value and, thus, will be treated as true in the context of a Boolean parameter.

When a Boolean data tag is passed as an argument to a function, it is likely that the Boolean value of true or false is desired, not the text. In this case, you omit the markers around the tag. This will result in the Boolean value of the tag being passed as an argument to the function. So, in the following examples:

```
<<if <<object.BooleanProp>> ... >>
```

```
<<if object.BooleanProp ... >>
```

The first will result in the Boolean data tag being expanded to result in:

```
<<if “false” ... >>
```

This will result in the text value of “false” being passed to the function.

The second line will result in the Boolean value of true or false being passed to the `<<if ...>>`, rather than the text “true” or “false.” Note that referencing a Boolean without the markers is only valid when the Boolean tag is passed as an argument to a function.

Strings

There are four property data types that will result in a string data tag. These property data types include:

- String
- Complex Table Selection
- Data Table Selection
- External Data (provides access to the file name and location, not the file data)

The value of the item the data tag provides access to will be placed in the script at expansion time. There is, however, a minor modification to the value that will occur for scripts used in SQL system connections. Any single quotes within the string will be escaped for the database, that is, a second single quote will be placed before the existing quote. This is the standard escape character in most database systems and is necessary as the single quote is used to denote the beginning and end of a string in a SQL statement. So, when expanded, if a string data tag contains the value:

```
The customer's car has front end damage.
```

the value in the script when data tag expansion occurs will be:

```
The customer''s car has front end damage.
```

Note the two single quotes in place of the previously single quote (used as an apostrophe here) within the word “customer’s”. As explained in the chapter on function tags, the `<<dequote...>>` function also provides this ability. However, for string data tags with a property as its data source, this behavior is automatic.

Another optional behavior in string property data tags is the ability to truncate the value, if needed. This is accomplished through the optional named parameter, `length=`. The syntax for this is as follows:

```
<<parent.stringDataTag length=n>>
```

Denoting this data tag in this manner, the value of the string will be truncated to the length of *n*. This truncation occurs before any quotes are escaped, so that the extra quotes added in that process are not affected by the truncation. As stated, `length` is an optional parameter and, if not provided, the entire value of the string will be placed in the script file during data tag expansion.

Another optional parameter to a property string data tag is `raw`. This parameter will return the value of the string without escaping the quotes it may contain. The syntax for this is:

```
<<object.stringDataTag.raw>>
```

Integral and Decimal Numbers

Integral data tags result from properties of the types Integral Number and Identifier. Decimal data tags result from properties of type Decimal Number. During data tag expansion, the value

of these tags are placed in the script without modification. In this respect these two data types are treated the same. It is when these values are passed as arguments to functions where the difference between the two types becomes important.

Integral Number data tags will contain whole number values. These data tags can be used with the math function tags that accept integral numbers. Many of the function tags that can accept the use of numerical values have a type parameter. When using this type of data tag, the value to the type parameter of the function is `Int`.

Decimal Number data tags will contain numerical values that have a fractional portion, such as 2.4 or 3.00. These data tags can be used with math function tags that accept decimal numbers. Many of the function tags that accept the use of numerical data, math tags as well as others, can accept a type parameter. When using this type of data tag, the value for the type parameter is `Float`.

Date

Date data tags contain a calendar date value. During data tag expansion of a SQL script, date tags are expanded in such a way that the resulting text is enclosed in the conversion function of the target database system that converts string values to date values. So, if a date data tag, `StatusDate`, contains the value 01/25/2006, then the data tag

```
<<transaction.StatusDate>>
```

in an Oracle database will be expanded to the value

```
to_date('01/25/2006', 'MM/DD/YYYY')
```

It is possible to get just the date value as a string by using the raw parameter, which is available to all date data tags. Continuing with the previous example, the data tag

```
<<transaction.StatusDate.raw>>
```

will be expanded to the value

```
01/25/2006
```

This can be useful if the date value is to be within some sort of string value within the database, such as a description. In this case, you do not want to convert the value to a database date format, but rather use it as a string.

Time

Time data tags contain a time of day value, in a 24 hour format. When data tag expansion occurs in a SQL script, the resulting value is enclosed in the conversion function for the target database system that is used to convert string values into times. So, if a data tag named `EndTime` contains the value 13:10:43, then the data tag

```
<<transaction.EndTime>>
```

in an Oracle database will be expanded to

```
to_date('13:10:43', 'HH24:MI:SS')
```

It is possible to access the value as a string, without the conversion function, by using the parameter raw, which is available to all Time data tags. Using the previous example data tag

```
<<transaction.EndTime.raw>>
```

will expand to the value

```
13:10:43
```

This is used whenever you wish to access just the time string, without converting it to the database time format.

Date And Time

Date and Time data tags are, in essence, a combination of the Time data type and the Date data type. This data tag type contains the calendar date and time of day in a single value. When data tag expansion occurs in a SQL script, the resulting value is enclosed in the conversion function for the target database system that is used to convert string values into dates and times. If a data tag named InspectionDateTime contains the value 02/13/2005 13:20:35, then the data tag:

```
<<transaction.InspectionDateTime>>
```

in an Oracle database will be expanded to

```
to_date('02/13/2005 13:20:35', 'MM/DD/YYYY HH24:MI:SS')
```

As with other data types, it is possible to access the string value without wrapping it in a conversion function, by using the parameter raw, which is available in all Date and Time data tags. Using the previous example data tag

```
<<transaction.InspectionDateTime.raw>>
```

will expand to

```
02/13/2005 13:20:35
```

This is used whenever just the date and time value is desired, without wishing to convert it before being processed by the enterprise system.

Formatting Dates and Times

These three data types that deal with dates and times support the use of the named parameter format=. This parameter accepts one or more of several date and time tokens. These tokens are combined to provide a picture of how the data should be placed in the file. When a date, time,

or date and time data tag contains the format parameter, the default format is overridden, including the conversion function within which the values are normally contained.

Following is a list of the tokens supported by these data tag types. In each of the examples the date and time is 02/07/2001 10:09:03 AM. The **Example** column contains the value for the token listed. The **Short Form** contains the example of the value that results by preceding the token with a hyphen, as in: %-m

Token	Description	Example	Short Form
%a	The three letter abbreviation of the day.	Wed	We
%A	The name of the day.	Wednesday	Wed
%b	The three-letter abbreviation of the month.	Feb	n/a
%B	The name of the month.	February	Feb
%d	The date of the month.	07	7
%j	The Julian date, with a leading 0.	038	38
%m	The two digit month (01-12)	02	2
%w	The numerical day of the week (0-6 Sunday = 0)	3	n/a
%y	The two-digit year	01	1
%Y	The four-digit year.	2001	n/a
%R or %r	The raw format of the value	36543,37	n/a
%H	The hour of the day, 24 hour format.	10	n/a
%h or %I	The hour of the day, 12 hour format	10	n/a
%M	The minutes of the hour.	09	9
%p	AM or PM indicator.	A	a
%S	Seconds of the minute.	03	3
%Z	The time zone when the time was recorded.	Central Standard Time	n/a
non-to-ken characters	Any non-format token character, or any character not preceded by the % sign passed to the named parameter format will be returned unchanged at the position at which it was placed in the parameter value.	n/a	n/a

Signature

Signature data tags result from Signature property types. This data tag type is used with the signature capture functionality available in Agentry. This functionality allows for an application to capture and store a signature the user enters on the screen. The image is stored as a bitmap, and is also available in the raw pixels.

This data tag type supports the following parameters. The value returned for all of these parameters is a string, with the exception of `bmp`, `row.n`, and `raw`.

- `type` - Will return either “image” or “none” during expansion. Image indicates that there is an image and the transaction was performed on a client device that supports this functionality. None indicates that the device does not support the signature capture functionality.
- `bmp` - This parameter returns the signature, if it exists, in a bitmap format. This returns a string of hexadecimal values that may be used as an argument to another utility program that processes the data, e.g. stores it in a database.
- `height` - This parameter returns the height, in pixels, of the signature image.
- `width` - This parameter returns the width, in pixels, of the signature image.
- `row.n` - This parameter returns the row of pixels, specified by `n`
- `signed` - Returns either true or false. True is returned when a signature has been captured on the client, or, for devices that do not support this functionality, if the check box control that replaces it has been checked. These values are returned as text values of “true” or “false.”
- `raw` - Returns the pixels that make up the image.

The syntax for these parameters is:

```
<<transaction.signatureProp.parameter>>
```

Of these parameters, only `type` and `signed` are always available. The others will return a data tag not found error if a signature was not captured on the client, i.e. `signed` returns false. Therefore, the return value of `signed` should be checked before attempting to access the other parameters.

Image

Image data tags result from Image property types. This data tag type is used with the image capture functionality available in Agentry. This functionality allows the application to interact with a device’s built in still camera, when present. A captured image is stored on the device as a file and referenced by the image property.

During synchronization this file data is sent to the server for processing as a part of the transaction data. To access this image data there are two options. The first is to use a file document management step. In this case, it is likely not necessary to reference the data tags for the image property, though they can be when necessary. For other step types access to the

image data requires the use of SDML data tags. Data tags for the image property include two parameters in the format `<<transaction.imageProperty.parameter>>`.

The following list describes these parameters and their purpose:

- `data` - This parameter returns the image data in ASCII-encoded hexadecimal values.
- `type` - This parameter returns the image type as stored on the client device. The possible return values of this parameter are `jpeg`, `bitmap`, and `unknown` when the file type is not determined.

<<agent>> Data Tag Container

The `<<agent>>` data tag container includes only a single member, `.version`. The data tag `<<agent.version>>` returns the full version of the Agentry Server.

This is the only member of this data tag member in the `<<agentry>>` data tag container. Additional members may be added in a future release.

SDML Function Tags Overview

SDML Function tags provide value processing and evaluation to the SDML. Function tags are represented in print with the syntax `<<funcName...>>`. Within the SDML there are numerous functions that provide processing for logic operations, string operations, and mathematical operations.

Function tags within the SDML will often return a value. That return value is placed at the point where the function tag exists within the script file in which it is contained. The use of function tags can provide a significant source of operational power, allowing for different sets of logic to be processed by a script at runtime, depending on conditions.

The following sections list each of the function tags available, including descriptions of their purpose and behavior, as well as usage syntax and similar information.

<<if>>

The `<<if...>>` function provides the if-then-else logic to the SDML. This is the most common decision making mechanism within any language. The `<<if>>` function receives a single argument, which it evaluates as being either true or false. If the argument is true, its first expression, `trueExpression`, is returned. Otherwise, the `falseExpression` value is returned. This expression must be preceded by the keyword `else`. `falseExpression` is optional and if it is not present, the `else` keyword cannot appear either.

This function can be used to return something as basic as a single word, or as complex as an entire SQL statement. The contents of either expression can contain SDML text as well. In the case where there is no `else` portion, and the `boolArg` is false, the return value of the `<<if...>>` function is an empty string.

Arguments

<pre><<if boolArg "trueExpression" [else "falseExpression"]>></pre>	
boolArg	The value to be evaluated as either true or false. May be a Boolean data tag or the Boolean return value of function call. If this argument is a Boolean data tag, the tag should be entered by name, excluding the tag markers (<< and >>) to return the Boolean value of that property, rather than the text value.

Expressions

- **trueExpression** – Required expression containing the value to be returned by the function when `boolArg` is true. This expression must be enclosed in double quotes.
- **falseExpression** – Optional expression containing the value to be returned by the function when `boolArg` is false. This expression must be preceded by the keyword `else` and the expression itself must be enclosed in double quotes.

Parameters

- N/A

<<case>>

The `<<case...>>` function provides the `switch-case` logic to the SDML. It takes a required switch argument and at least one case-expression argument pair. It may take as many additional case-expression pairs as are needed, plus an optional default argument.

During expansion, this function tag compares the value of the switch argument to each provided case argument in turn. It will return the expression argument for the first case argument to which the switch argument matches.

As an optional argument, a default value may be provided that will be returned by the function when the switch argument does not match any provided case argument. The syntax for the default return value is `default=returnValue`, where `default` is a keyword. For this reason, neither the value of the switch argument, nor any of the case arguments may be the value `default`.

Arguments

<pre><<case switch case1=exprssion1 [caseN=expressionN] [default=defaultExpression]>></pre>	
switch	The value the function will switch on, comparing to the value of each <code>case</code> argument until a match is found.
case1	The first case argument to the function. This is a required argument and must be immediately followed by an equal sign (=) with no whitespace between the case argument and the sign. The <code>expression1</code> argument immediately follows the equal sign, also with no whitespace allowed.
expression1	The first <code>expression</code> argument to the function. This is a required argument and contains the value the function will return when <code>switch</code> matches <code>case1</code> . <code>case1</code> and <code>expression1</code> are separated by an equal sign with no whitespace allowed between them, as in: <code>case1=expression1</code> . Any SDML text in <code>expression1</code> will be expanded after it has been returned by the function.
caseN	Additional optional <code>case</code> arguments to the function. If <code>case1</code> does not match <code>switch</code> , the function will compare each subsequent case argument in order until a match is found. Each <code>caseN</code> argument must be followed by and equal sign and then a corresponding <code>expressionN</code> value. No whitespace can exist between each case-expression pair.
expressionN	Additional optional <code>expression</code> arguments to the function. Each case argument must be followed by a corresponding <code>expressionN</code> value. Each case-expression argument pair must be separated by an equal sign with no white space allowed between them, as in: <code>caseN=expressionN</code> . Any SDML text in <code>expressionN</code> will be expanded after it has been returned by the function.

<pre><<case switch case1=exprssion1 [caseN=expressionN] [default=defaultExpression]>></pre>	
<pre>default=defaultExpression</pre>	<p>This optional argument specifies the expression returned by the function should the <code>switch</code> not match any of the <code>case</code> values. The syntax for this argument requires the text <code>default=</code> followed by the default expression the function should return.</p>

Parameters

None

<<skip>>

The `<<skip . . .>>` function will force the Agentry Server to skip the step definition in which the function call is contained. This function takes an optional comment argument, the contents of which will be the log message generated by the Server for the log file of the step type's system connection. This function is only valid when called within the script component of a module-level step definition.

This function can be used during testing to skip over a script that you do not wish to run, or in certain production situations where you may not wish a script to run under certain conditions. The primary intent of this function is the result of the requirements of the contents of a SQL step's script. This script cannot be empty, nor can it contain only SDML logic with no valid SQL statement to be processed. Depending on conditional processing, such as queries returned by the `<<if . . .>>` function, it is possible for a script to return a valid SQL statement in one condition, but not in another. In this situation, the `<<skip>>` function should be the expression returned when no SQL statement should be run. Note that this function is not limited to SQL step definitions, though this is its primary intended use.

Arguments

<pre><<skip ["comment"]>></pre>	
<pre>comment</pre>	<p>This is an optional argument. It contains any text value that will be used as a log message generated by the Server for the log file of the step type's system connection.</p>

Parameters

None

<<stop>>

The <<stop . . .>> function will stop the Agentry Server's interactions with the the back end system. Any subsequent steps within the same group of the same parent definition will not be processed. The function takes an optional comment argument, the contents of which will be written as a log message by the server to the log file for the parent step's back end log category.

As an example of the function's behavior, if the second of four client exchange steps within a fetch contains a <<stop>> function, that step and those that come after it will not be run. This function will result in a commit being performed, committing any changes made previously by the processing of the previous steps within the same parent.

Arguments

<<stop ["comment"]>>	
comment	This optional argument contains text which will be written as a log message by the server to the log file for the parent step's back end log category.

Parameters

None

<<abort>>

The <<abort . . .>> function within a step will result in that step's processing being halted. Any subsequent steps within the same parent definition will also not be processed. Any changes made by the previous steps in the group will be rolled back. This function takes an optional comment argument, the contents of which will be written as a log message by the Agentry Server to the log category of the step's system connection.

As an example of this function's behavior, if the third of five steps in a fetch's server exchange steps is aborted, steps four and five will not be run either. Changes made by the first two steps will be rolled back.

Note that the <<abort>> and <<rollback>> functions perform the exact same behavior.

Arguments

<<abort ["comment"]>>	
comment	This optional argument contains text which will be written as a log message by the Agentry Server to the log category of the step's system connection.

Parameters

None.

<<rollback>>

The <<rollback . . . >> function within a step will result in that step's processing being halted. Any subsequent steps within the same parent definition will also not be processed. Any changes made by the previous steps in the group will be rolled back. This function takes an optional comment argument, the contents of which will be written as a log message by the Agentry Server to the log category of the step's system connection.

As an example of this function's behavior, if the third of five steps in a fetch's server exchange steps is rolled back, steps four and five will not be run either. Changes made by the first two steps will be rolled back.

Note that the <<abort>> and <<rollback>> functions perform the exact same behavior.

Arguments

<<rollback ["comment"]>>	
comment	This optional argument contains text which will be written as a log message by the Agentry Server to the log category of the step's system connection.

Parameters

None.

<<and>>

Description

The <<and . . . >> function performs a logical conjunction of two or more Boolean values. If all arguments to the function are true, the function will return true. Otherwise, the <<and . . . >> function will return false.

This function is almost always used as an argument to another function, normally the <<if . . . >> function. The reason for this is that the value returned is a Boolean value within the SDML and will simply return the text values of "true" or "false," if not an argument to another function.

Arguments

<<and boolArg1 boolArg2 [boolArg3...boolArgN]	
boolArg1-N	The boolean values checked for true or false by the function. May be either a Boolean data tag, or a function that returns a Boolean value. The function must have at least two arguments, and up to as many as needed. Each is checked in the order listed, until a false value is found.

Parameters

None

<<or>>

Description

The <<or . . . >> function performs the logical disjunction of two or more Boolean values. Each argument is compared in the order listed until a true value is found, at which point the function returns true. If all arguments contain a false value, then the function will return false.

This function is almost always used as an argument to another function, normally the <<if . . . >> function. The reason for this is that the value returned is a Boolean value within the SDML and will simply return the text value of either “true” or “false” if not passed as an argument to a function.

Arguments

<<or boolArg1 boolArg2 [boolArg3...boolArgN]>>	
boolArg1-N	The boolean values checked for true or false by the function. May be either a Boolean data tag, or a function that returns a Boolean value. The function must have at least two arguments, and up to as many as needed. Each is checked in the order listed, until a true value is found.

Parameters

None

<<not>>

The <<not . . . >> function inverts the Boolean value of boolArg and returns this inverted value. If boolArg is true, the function will return false, and vice versa.

This function is almost always used as an argument to another function, normally the `<<if . . >>` function. The reason for this is that the value returned is a Boolean value within the SDML and will simply return the text value of either “true” or “false” if not passed to another function.

Syntax

`<<not boolArg>>`

Return Value

Boolean

Arguments

<code><<not boolArg>></code>	
boolArg	A boolean value that is inverted by the function. May be a Boolean data tag or a function that returns a Boolean value.

Parameters

None

`<<eq>>`

The `<<eq . . >>` function compares `arg1` and `arg2`, based on the value of the “type” parameter if present, and returns true if the two values are found to be equal. Otherwise, this function returns false. If the type parameter is not specified, the default comparison is String.

When a type is specified any values not of that type are converted prior to the comparison. So, if an integral data tag contains a value of 10 and a decimal number contains a value of 10.1, and the type is `Int`, these two values would be considered equal, as the decimal value would be converted to an integer, and the decimal portion is truncated.

Arguments

<code><<eq arg1 arg2 [type=Int Float String]>></code>	
arg1	The first value of the two compared by the function. May be a hard coded value, data tag, or a function.
arg2	The second value of the two compared by the function. May be a hard coded value, data tag, or a function.

Parameters

- **type** - This optional named parameter specifies how the function will compare the values of the two arguments. The acceptable values for this parameter are
 - **Int** - The arguments are Integral Numbers.
 - **Float** - The arguments are Decimal Numbers (short for “floating point number”)
 - **String** - The arguments are String values.

<<ne>>

Description

The <<ne...>> function compares arg1 and arg2, based on the value of the “type” parameter if present, and returns false if the two values are found to be equal. Otherwise, this function returns true. If the type parameter is not specified, the default comparison is as Strings.

When a type is specified any values not of that type are converted prior to the comparison. So, if an integral data tag contains a value of 10 and a decimal number contains a value of 10.1, and the type is Int, these two values would be considered equal, as the decimal value would be converted to an integer, and the decimal portion is truncated.

Syntax

<<ne arg1 arg2 [type=Int|Float|String]>>

Arguments

<<ne arg1 arg2 [type=Int Float String]>>	
arg1	The first value of the two compared by the function. May be a hard coded value, data tag, or a function.
arg2	The second value of the two compared by the function. May be a hard coded value, data tag, or a function.

Parameters

- **type** - This optional named parameter specifies how the function will compare the values of the two arguments. The acceptable values for this parameter are
 - **Int** - The arguments are to be compared as Integral Numbers.
 - **Float** - The arguments are to be compared as Decimal Numbers (short for “floating point number”)
 - **String** - The arguments are to be compared as Strings

<<gt;>

The `<<gt; . . . >>` function compares `arg1` and `arg2`, based on the value of the “type” parameter if present. It returns true if `arg1` is greater than `arg2`. If `arg1` is less than or equal to `arg2`, it returns false. If the `type` parameter is not specified, the default comparison is as Strings. When comparing values of different data types, it is strongly recommended that you do specify the type.

When a type is specified any values not of that type are converted prior to the comparison. So, if an integral data tag contains a value of 10 and a decimal number contains a value of 10.1, and the type is `Int`, these two values would be considered equal, as the decimal value would be converted to an integer, and the decimal portion is truncated. String comparisons are made based on alphabetical order.

Arguments

<<gt; arg1 arg2 [type=Int Float String]>>	
arg1	The first value of the two compared by the function. May be a hard coded value, data tag, or a function.
arg2	The second value of the two compared by the function. May be a hard coded value, data tag, or a function.

Parameters

- `type` - This optional named parameter specifies how the function will compare the values of the two arguments. The acceptable values for this parameter are
 - **Int** - The arguments are to be compared as Integral Numbers.
 - **Float** - The arguments are to be compared as Decimal Numbers (short for “floating point number”)
 - **String** - The arguments are to be compared as Strings

<<lt>>

Description

The `<<lt; . . . >>` function compares `arg1` and `arg2`, based on the value of the “type” parameter if present. It returns true if `arg1` is less than `arg2`. If `arg1` is greater than or equal to `arg2`, it returns false. If the `type` parameter is not specified, the default comparison is as Strings.

When a `type` is specified any values not of that type are converted prior to the comparison. So, if an integral data tag contains a value of 10 and a decimal number contains a value of 10.1, and the type is `Int`, these two values would be considered equal, as the decimal value would be

converted to an integer, and the decimal portion is truncated. String comparisons are made based on alphabetical order.

Arguments

<<lt arg1 arg2 [type=Int Float String]>>	
arg1	The first value of the two compared by the function. May be a hard coded value, data tag, or a function.
arg2	The second value of the two compared by the function. May be a hard coded value, data tag, or a function.

Parameters

- **type** - This optional named parameter specifies how the function will compare the values of the two arguments. The acceptable values for this parameter are
 - **Int** - The arguments are to be compared as Integral Numbers.
 - **Float** - The arguments are to be compared as Decimal Numbers (short for “floating point number”)
 - **String** - The arguments are to be compared as Strings

Expressions

None

<<ge>>

The <<ge . . .>> function compares **arg1** and **arg2**, based on the value of the **type** parameter if present. It returns true if **arg1** is greater than or equal to **arg2**. If **arg1** is less than **arg2**, it returns false. If the **type** parameter is not specified, the default comparison is as Strings.

When a **type** is specified any values not of that type are converted prior to the comparison. So, if an integral data tag contains a value of 10 and a decimal number contains a value of 10.1, and the **type** is **Int**, these two values would be considered equal, as the decimal value would be converted to an integer, and the decimal portion is truncated. String comparisons are made based on alphabetical order.

Arguments

<<ge arg1 arg2 [type=Int Float String]>>	
arg1	- The first value of the two compared by the function. May be a hard coded value, data tag, or a function.
arg2	- The second value of the two compared by the function. May be a hard coded value, data tag, or a function.

Parameters

- **type** - This optional named parameter specifies how the function will compare the values of the two arguments. The acceptable values for this parameter are
 - **Int** - The arguments are to be compared as Integral Numbers.
 - **Float** - The arguments are to be compared as Decimal Numbers (short for “floating point number”)
 - **String** - The arguments are to be compared as Strings

<<le>>

Description

The <<le...>> function compares **arg1** and **arg2**, based on the value of the **type** parameter if present. It returns true if **arg1** is less than or equal to **arg2**. If **arg1** is greater than **arg2**, it returns false. If the **type** parameter is not specified, the default comparison is as Strings.

When a **type** is specified any values not of that type are converted prior to the comparison. So, if an integral data tag contains a value of 10 and a decimal number contains a value of 10.1, and the type is **Int**, these two values would be considered equal, as the decimal value would be converted to an integer, and the decimal portion is truncated. String comparisons are made based on alphabetical order.

Arguments

<<le arg1 arg2 [type=Int Float String]>>	
arg1	The first value of the two compared by the function. May be a hard coded value, data tag, or a function. The data type of the value must be Integral Number, Decimal Number, or String.
arg2	The second value of the two compared by the function. May be a hard coded value, data tag, or a function. The data type of the value must be Integral Number, Decimal Number, or String.

Parameters

- **type** - This optional named parameter specifies how the function will compare the values of the two arguments. The acceptable values for this parameter are
 - **Int** - The arguments are to be compared as Integral Numbers.
 - **Float** - The arguments are to be compared as Decimal Numbers (short for “floating point number”)
 - **String** - The arguments are to be compared as Strings

<<empty>>

The <<empty . . .>> function allows you to check whether or not an object collection has any objects or if it is empty. If the collection contains no objects, this function will return true. If the collection contains at least one object this function will return false. This function is only valid when an object collection property is in scope for the step definition in which it is used.

Arguments

<<empty objectCollectionArg>>	
objectCollectionArg	This argument is a data tag representing an object collection. This collection must be defined. It is evaluated by the function for members.

Parameters

None

<<notEmpty>>

The <<notEmpty . . .>> function allows you to check whether or not an object collection has any objects or if it is empty. If the collection contains no objects, this function will return

false. If the collection contains at least one object this function will return true. This function is only valid when an object collection property is in scope for the step definition in which it is used.

Arguments

<<notEmpty objectCollectionArg>>	
objectCollectionArg	This argument is a data tag representing an object collection. This collection must be defined. It is evaluated by the function for members.

Parameters

None

<<size>>

The <<size...>> function returns the number of object instances in an object collection. This value is always 0 or higher.

Arguments

<<size objectCollectionArg>>	
objectCollectionArg	This is the object collection whose members are to be counted.

Parameters

None

<<exists>>

The <<exists...>> function determines whether or not the specified object collection exists. If the collection specified by the `objectCollectionArg` argument exists, this function will return true. Otherwise it returns false.

A common use for this function outside of development testing is to verify the data tag returned by a call to the function <<sql...>> exists and contains data.

This function may also useful during testing or debugging of an application. It can be useful in applications that are deployed with various configurations from one installation to the next. The usage of this function can be helpful in determining if a specific configuration contains all of the definitions necessary.

Arguments

<<exists objectCollectionArg>>	
objectCollectionArg	This argument is the data tag representing the object collection that whose existence is to be checked. This argument must be a data tag that contains the name of an object collection whose existence is to be confirmed.

Parameters

None

<<foreach>>

The `<<foreach...>>` function allows for iteration over an object collection property. This function takes as its single argument the name of an object collection. The `expression` can contain text and SDML that will be returned once for each member of the named collection. If the collection is empty, nothing is returned.

This function allows you to iterate over a collection of objects. The expression specified will be returned once for each member and normally contains SQL and SDML that is to be processed for a single object. The `<<foreach...>>` function is commonly used for INSERT statements, which can only insert a single record at a time. It is often seen in SQL steps used to update a client exchange table during fetch processing.

The `expression` to the function can also contain an additional data tag, `<<my>>`. This tag represents the current member of the collection being processed by the function. To use the `<<my>>` data tag, the same syntax is used as in other areas where property data tags are used.

There is also the optional `<<key>>` tag available within a `<<foreach>>` expression, which provides the name of the current key being iterated over. This is normally only used in conjunction with SQL Flunkies, explained later in this chapter.

Arguments

<<foreach objectCollectionArg expression>>	
objectCollectionArg	The name of a collection that the function is to iterate over.
expression	The text to be returned once for each member of the collection. The expression can contain plain text and SDML. It will be submitted for expansion once for each collection member.

Parameters

None

<<upper>>

The <<upper . . . >> function converts a given string to all uppercase characters. The value returned is the string passed in with all characters converted to upper case. Any non-alphabetical characters, such as numbers, symbols, or punctuation (% , \$, etc.) are returned unchanged.

Arguments

<<upper stringArg>>	
stringArg	This argument is a string value to be converted by the function. It can be either a hard coded value, a data tag, or a string return value from a function.

Parameters

None

<<lower>>

The <<lower . . . >> function converts a given string to all lower case characters. The value returned is the string passed in with all characters converted to lower case. Any non-alphabetical characters, such as numbers, symbols, or punctuation ("% , \$" , etc.) are returned unchanged.

Arguments

<<lower stringArg>>	
stringArg	This argument is a string value to be converted by the function. May be a hard coded value, a data tag, or a string return value from a function.

Parameters

None

<<length>>

This function returns the length of a string value. All printable characters within the string are counted. This includes white space characters, where tabs are counted as a single character, and symbols, such as \$ or % . Non-printable characters are also counted, such as newline and carriage returns. Remember in Windows systems that the end of a line in a multi-line string

value contains two command characters, \n and \r, which will be counted by the <<length...>> function as one character each.

Arguments

<<length stringArg>>	
stringArg	The string value evaluated by the function. May be a hard coded value, a string data tag, or the return value of another function.

Parameters

None

<<join>>

Description

The <<join...>> function concatenates two or more string values together, with each separated by the value of the optional named parameter, join. If the join parameter is not specified, the values are concatenated together without any character separating them.

Arguments

<<join stringArg1 stringArg2 [stringArg3...stringArgN] join=joinString>>	
stringArg1/stringArg2	The required arguments to the function, which are the strings that will be joined. May be a hard coded value, an string data tag, or the return value of a function.
stringArg3-N	The optional additional strings to be joined. May be a hard coded value, string data tag, or the return value of another function.

Parameters

- **joinString** – The value of this optional named parameter contains the character or string used to join the arguments together.

<<dequote>>

The <<dequote...>> function, by default, removes any double-quote characters from a given string. It contains three optional named parameters, however, that significantly alter and enhance this functionality. First, by including the quote parameter, you can specify a different character to be removed form the given string or strings.

Second, the `replace` parameter can specify the character you wish to replace the quote character with. Finally, the `join` parameter allows you to specify the character used to join multiple string arguments to the function together.

Arguments

<<dequote stringArg1 [stringArgN] [quote=quoteChar] [replace=repChar] [join=joinString]>>	
stringArg1	This argument contains the string to be “dequoted.” May be a hard coded value, a data tag, or the return value of another function call.
stringArgN	Additional, optional argument(s) to be dequoted, and joined together with the previous arguments.

Parameters

- **quote** – The value to this named parameter is the character to be removed from the given string or strings. If not provided, the default value is double quotes. If the value to this parameter contains more than one character, the first will be used and the rest ignored.
- **replace** – The value to this named parameter is the single character to replace the “quote” character with in the string. If the value to this parameter contains more than one character, the first will be used and the rest ignored.
- **join** – The value to the named parameter is the character or string used to join together the arguments to the function. If not provided, each string is separated by a single white space.

<<trunc>>

The <<trunc . . . >> function will truncate the given stringArg to the number of characters of the value given to the required length parameter. By default, the characters are counted from the left most position up to and including the character at the position specified by the length parameter. This includes any white space characters, and also includes the end of line characters of line feed and carriage return, which each count as one.

If the `from` parameter is given and its value is “left”, the counting begins at the right-most character of the string, truncating the left characters beyond the given length.

If the stringArg is shorter or equal to the length specified, the entire string is returned.

Arguments

<<trunc stringArg length=lengthParam [from=left right]>>	
stringArg	This argument contains the string to be truncated by the function. May be a hard coded value, a data tag, or another function.

Parameters

- **length** – This required named parameter specifies the length to which the stringArg value should be truncated.
- **from** – This optional named parameter specifies the portion of the stringArg value to be truncated. The values are left or right; any other value will be ignored and the default value of right will be used.

<<wordTrunc>>

The <<wordTrunc . . . >> function is similar to the <<trunc . . . >> function in that it will truncate a string to a given length. The length parameter specifies the maximum length of the string and must be provided to the function. The difference between this function and <<trunc . . . >> is that <<wordTrunc . . . >> will always end its truncation on a white space. That is, the truncation of the string will be at the end of a whole word.

The string returned by the function will be, at most, the size of the length specified. However, if the specified length ends in the middle of the word, the last white space character before this point will be where the string returned ends.

The start parameter specifies the starting point of the function. If this parameter is given, the function will count from the beginning of the string up to the start character. Then, the function will count from this point up to the length value of characters.

If the string contains no white space, then length number of characters will be returned. If the string is shorter or equal to the length, the entire string will be returned.

Syntax

<<wordTrunc stringArg length=lengthParam [start=startParam]>>

Arguments

- **stringArg** – The string value to be truncated by the function. May be a hard coded value, a string data tag, or the return value of another function, provided it returns a string.

Parameters

- **length** – This required named parameter specifies the maximum length of the string returned by the function.
- **start** – This optional named parameter specifies the starting position from which the function will begin counting. Any characters before this position will be truncated, as will any characters beyond the value of the length parameter. If this parameter is not present, the starting position will always be the first character of the string, as specified by the from parameter.

<<cgi>>

The <<cgi . . .>> function can operate with either a single unnamed argument, or with many arguments in name-and-value pairs. If a single stringArg is given, it expands to a string that has all characters converted to CGI scoped values. If one or more name and value pair arguments are given, they are formatted to a string that is the named pairs, joined by ampersands (&), with the values CGI escaped.

The CGI function escapes strings following the CGI conventions certain characters are replaced with a % followed by two hexadecimal digits that are the ASCII value for the character.

The stringArg will have any characters it contains escaped according to CGI conventions. The name-and-value pairs will be formatted into named parameters and values, with the values also escaped according to CGI standards. The order of the named parameters is not preserved when this function is expanded. Also, the CGI function escapes spaces with %20's rather than with +'s. Both are allowed by the CGI convention.

Arguments

<<cgi stringArg>> --OR-- <<cgi named1=value [named2=value...namedN=value]>>	
stringArg	A text string that will be escaped according to CGI conventions. This may be a hard coded value, a data tag, or the return value of a function.
named1-n	A named parameter to be returned with a value that will be formatted according to CGI conventions. May be a hard coded value, a data tag, or the return value of a function.

<<cgi stringArg>> --OR-- <<cgi named1=value [named2=value...namedN=value]>>	
value	A value to the corresponding named parameters that will be formatted according to CGI conventions. May be a hard coded value, a data tag, or the return value of a function.

Parameters

None

<<sum>>

The <<sum...>> function provides the operation of the plus sign (+) operator in other languages. This function will sum the arguments and return the result. There must be at least two arguments provided to the function, and there can be as many more arguments as needed.

The data type of these values can be Strings, provided the string contains only numbers, sign, and a single decimal. The values of string data types will be converted before being passed to the function.

Arguments

<<sum numArg1 numArg2 [numArg3...numArgN]>>	
numArg1-2	The required arguments, numerical, that are summed together. Maybe a hard coded value, data tag, or the return value of another function. If the value is hard coded and contains a negative or postivie sign, the entire value must be enclosed in double quotes, as in "-12.34".
numArg3-N	The optional numeric aguments to be summed together with all other arguments. May be a hard coded value, data tag, or the return value of antoher function. If the value is hard coded and contains a negative or postiive sign, the entire value must be enclosed in double quotes, as in "-10.23".

Parameters

None

<<diff>>

The <<diff...>> function provides the operation of the minus sign (-) operator in other languages. This function subtracts the second argument from the first and returns the difference. This function takes two and only two arguments.

String data tags or string return values may be passed to the function, provided those values contain only numeric, sign, and a single decimal character.

Arguments

<<diff numArg1 numArg2>>	
numArg1	The first numerical value from which numArg2 is subtracted. May be a hard coded value, data tag, or the return value of another function. If the value is hard coded and contains a negative or positive sign, the entire value must be enclosed in double quotes, as in "-12.34".
numArg2	The second numerical value that will be subtracted from numArg1. May be a hard coded value, data tag, or the return value of another function. If the value is hard coded and contains a negative or positive sign, the entire value must be enclosed in double quotes, as in "-10.23".

Parameters

None

<<prod>>

The <<prod...>> function provides the operation provided by the multiplication operator, either x, or more commonly *, in other languages. This function multiplies the first argument by the second and returns the product.

A String data tag may be passed as an argument to the function, provided it contains only numerical characters, sign, and a single decimal character.

Arguments

<<prod numArg1 numArg2>>	
numArg1	This required argument contains the value that will be multiplied by <code>numArg2</code> . May be a hard coded value, data tag, or the return value of another function. If the value is hard coded and contains a negative or positive sign, the entire value must be enclosed in double quotes, as in “-12.34”.
numArg2	This required argument contains the value to be multiplied by <code>numArg1</code> . May be a hard coded value, data tag, or the return value of another function. If the value is hard coded and contains a negative or positive sign, the entire value must be enclosed in double quotes, as in “-10.23”.

Parameters

None

<<div>>

The `<<div . . . >>` function provides the same operation as is provided by the division sign (/) operator in other languages. This function divides the second argument into the first and returns the results.

A string data tag may be passed as an argument to the function, provided it contains only numerical characters, sign, and a single decimal character.

Arguments

<<div dividendArg divisorArg>>	
dividendArg	This required argument contains the number to be divided by the <code>divisorArg</code> . May be a hard coded value, data tag, or the return value of another function. If the value is hard coded and contains a negative or positive sign, the entire value must be enclosed in double quotes, as in “-12.34”.
divisorArg	This required argument contains the number to be divided into the <code>dividendArg</code> . May be a hard coded value, data tag, or the return value of another function. This value must not be 0. If the value is hard coded and contains a negative or positive sign, the entire value must be enclosed in double quotes, as in “-10.23”.

Parameters

None

<<remainder>>

The <<remainder . . . >> function provides the modulus operation of the modulus sign operator, usually %, in other languages. This function divides the first argument by the second and returns the remainder of the division.

String data tags can be passed as arguments to the function, provided the value contains only numeric characters, sign, and a single decimal character.

Arguments

<<remainder dividendArg divisorArg>>	
dividendArg	The value to be divided by the divisorArg. May be a hard coded value, data tag, or the return value of another function. If the value is hard coded, and it contains a positive or negative sign, the entire value must be enclosed in quotes, as in "-12.34".
divisorArg	The value to be divided into the dividendArg. May be a hard coded value, data tag, or the return value of another function. If the value is hard coded, and it contains a positive or negative sign, the entire value must be enclosed in double quotes, as in "-10.23". The value of this argument must not be 0.

Parameters

None

<<local>>

The <<local . . . >> function allows you to create data tags within the script. The data tags created are always string values. Their scope is limited to the step within which they are created, and the other steps within the same parent definition that follow it. So, if an object contains 4 read steps, and the second contains a <<local . . . >> function call, the data tag or tags created will be available in the second step as well as the third and fourth. It will not be available in the first.

If the value for a tagName argument is a hard coded value, or contains a mixture of text and SDML, the value must be enclosed in quotes.

To reference a local data tag, the syntax is <<local . tagName>>, where tagName is the name given in the function call. Local data tags support the named parameter length=,

which will truncate the string to the given value. String values are not dequoted; time and date values are not wrapped in any type of conversion function.

Arguments

<<local tagName1=value [tagName2=value...tagNameN=value]>>	
tagName1	This required argument is the name that will be given to the data tag created. Its corresponding value will be the value the tag contains.
tagName2-N	These optional arguments are the same as tagName1 and allow for the creation of multiple data tags with the same function call.

Parameters

- **length** – This optional named parameter takes a non-negative whole number and specifies the maximum number of characters to assign to the local data tag created by the function.

<<sql>>

The <<sql . . . >> function allows you to create data tags based on the data returned by a SQL statement, specifically a SELECT statement. All records returned by the statement are stored in the newly created data tag, commonly referred to as a SQL flunky. Each field of each record returned by the SELECT statement can be accessed in the flunky.

The flunky created is named the same as the argument you provide. This SQL flunky has a scope limited to the script within which it is contained.

The <<sql . . . >> function is normally used to retrieve a small number of records, usually consisting of one or two selected fields, to retrieve data using a simple SELECT statement, where otherwise it may be necessary to create a more complex statement within the script. The statement used in the function call should never be used to perform the main processing of the script, nor to return large numbers of records. Rather, it should be used to aid in this main processing. Furthermore, the statement should never contain UPDATE or INSERT statements. Additionally, if the SELECT statement is returning more than 10 records at a time, the design of your script should be reevaluated and adjusted so that this is not the case. The main reason for this is performance.

While the <<sql . . . >> function will not cause any delays or hitches in processing if used correctly, using it to return large amounts of data will slow down the processing of the Agentry Server considerably. Each record returned by the function must be processed by the Agentry Server and stored in memory until the script has completed processing. This can tie up a significant amount of the system resources in the event of a large number of records being returned.

As stated, the SQL flunky created by the `<<sql . . .>>` function call is only in scope within the script in which it is called. If the value is needed in other Steps within the same parent definition, the desired values can be assigned to a local flunky, via use of the `<<local...>>` function described previously.

The syntax to reference the SQL flunky created by this function is as follows

```
<<sql.nameArg[.recordIndex][.fieldName]>>
```

All SQL flunkies are referenced beginning with `sql`. The `nameArg` is the name of the argument as you provided when calling the `<<sql . . .>>` function. The `recordIndex` is a numerical value indicating which record within the data set you wish to access. The records are referenced in the order in which they were returned by the database system, and are indexed starting with 0. The field name is the name of the column, or its alias, that contains the data you wish to retrieve. So, to access a field named `COST` in the first record of a SQL flunky named `prodCost`, the tag would be `<<sql.prodCost.0.COST>>`

Arguments

<<sql nameArg="SQLStatement">>	
nameArg	The name of the SQL flunky to be created as a result of processing the argument value, <code>SQLStatement</code> . The <code>SQLStatement</code> must always be enclosed in double quotes and should contain a <code>SELECT</code> statement.

Parameters

None

<<include>>

The `<<include . . .>>` function allows you to include the contents of another file within the file calling the function. This content will be included at the point where the function call is placed. The included file should always be a plain text file.

This function is only used in specific cases and there are certain caveats that accompany its usage. These caveats are related to the fact that the file referenced does not need to be associated with any definition within the Agentry Editor. Because of this fact, the included file may not be controlled or monitored by the Editor. This means that, during a publish, this file will not be copied or transferred in any way to the Agentry Server. Therefore, changes made to this file will not be updated to the Server during a publish, meaning the file must be moved separately if changes are made to it.

Related to this, if the included file does not exist in a location that is accessible to both the Editor and Server, it must be copied to two separate locations, one for each of these components.

Arguments

<<include fileName>>	
fileName	The name of the file whose contents are to be included in the file calling the function.

Parameters

None

Agentry Test Script Overview

The Agentry Test Script is an XML schema supported by the Agentry Test Environment that can be used to automate testing the Client behavior of a mobile application built on Agentry. The Agentry Test Environment includes a script recorder that allows for the recording of test scripts, and can then play back those test scripts.

The test script language includes the ability to interact with all controls present on the client application's interface, including field selection, data entry, button clicks, and navigation. Additionally, this language also supports the ability to check the current values of labels, fields, and other items displayed on the client application's interface for expected values.

In addition to direct client interaction, the test script also includes the ability to query database systems for expected values. This can be used after transmit to verify the proper functioning of transactions related to the back end processing that is defined within those transactions.

Elements within the test script XML schema are logically grouped into the following categories:

- **Script Elements:** Elements for the script itself, including the top-level <script> element and elements related to logging and script execution.
- **Button Elements:** Elements that allow for interaction with button definitions, including selection (or "clicking"), checking the state of the button, and label values.
- **Field Elements:** Elements that allow for interaction with detail screen fields. Note that certain field edit types are supported by elements in other groups.
- **List Elements:** Elements for working with list controls of various types. This includes list controls on list screens, as well as the various list types that can be defined for detail screen fields.
- **Tree Elements:** Elements for working with tree controls. This includes tree controls presented by detail screen fields.
- **Scanner Elements:** Elements for simulating scanner behaviors, including passing values in as barcode scanner values.
- **SQL Elements:** Elements for creating connections to and running queries against database back end systems. Values can be returned by these queries and checked against expected values.

- **Tab Elements:** Elements for working with the tab controls presented by screen sets for each child screen definition.
- **Window Elements:** Elements for closing windows on the client. Rarely used, as navigational actions defined to close screen sets should be used wherever present.
- **Client Elements:** Elements for affecting the client process, including restarting and other behaviors.
- **Client Host Elements:** Elements to interact directly with the client device, which may in turn affect the test client, such as entering key strokes or executing commands on the client device.

Common Test Script Element Attributes

The following attributes are common to the bulk of the elements within the Agentry Test Script XML schema. They relate primarily to time outs for the execution of a given element, and the amount of time to pause between the execution of one element and the next. Setting these attributes in the <script> element of the test script will set defaults for the entire script execution that can then be overridden by individual child elements if needed.

Name	Description	Data Type	Default Value	Required
timeout	The amount of time to wait for the element to finish processing before returning an error. This value can be set in the <script> element for the entire script and/or at each processing element within the test script. Child elements with this attribute will override the value set in parent elements. The value is specified in milliseconds.	Positive Integer	N/A	No
sleep	The amount of time to pause after the element is executed. This value can be set in the <script> element for the entire test script and/or at each processing element within the test script. Child elements with this attribute will override the value set in parent elements. The value is specified in milliseconds.	Positive Integer	N/A	No

Agentry Test Script: Script Elements Overview

The script elements within the Agentry Test Script language include the top-level <script> element that is the root to all test scripts, as well as elements for logging messages and pausing the execution of the script.

When a new test script is created by the Script Recorder in the ATE, it automatically creates the `<script>` element and required attributes. The other elements `<script-log>` and `<script-pause>` are manually added when needed.

Included in the `<script>` element is the attribute specifying the name space for the Agency Test Script language, which is `xmlns:ags="urn:script.Agency.Syclo"`. If a script is created manually this should be an attribute included in the `<script>` element.

`<script>`

The `<script>` element is the root element for any Agency Test Script. All elements are contained within the `<script>` element, either directly or as descendents. Two of its attributes, `timeout` and `sleep`, will affect how each element it contains is processed. The `timeout` attribute sets the duration of time to wait for an element to be processed. Setting the `timeout` in the `<script>` element will set a timeout for all elements. Other elements may individually override this duration with their own `timeout` attributes. The `sleep` attribute will set the amount of time to wait for before processing an element within the test script. Setting this attribute for the `<script>` element will affect all elements, waiting to process each for the configured time. Other elements may individually override this duration with their own `sleep` attributes.

Structure

Contained By:

- None - Root element for Agency Test Script files.

Table 11. Attributes

Name	Description	Data Type	Default Value	Re-quired
xmlns	This attribute defines the base namespace.	String	urn:script:Agency:Syclo	Yes
xmlns:ags	This attribute defines the ags namespace, making it the default namespace.	String	urn:script:Agency:Syclo	Yes
xmlns:meta	This attribute defines the meta namespace used for comments.	String	urn:meta:Editor.Agency:Syclo	Yes

Name	Description	Data Type	Default Value	Required
show-execute	This attribute enables or disables displaying log messages from the test script on standard output. When set to true log messages are written to standard output.	Boolean	False	No
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<script-log>

The <script-log> element will write a log statement to the log file `AgentryScriptOutput.log`. The contents of this element are the message written to the log file. The log message level must also be specified in the level attribute to the element, which indicates the severity of the log message.

Structure

Contains:

- Text - The log message to be written to the `AgentryScriptOutput.log` log file.

Contained By:

- <script>

Table 12. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
level	<p>This attribute can contain either a numeric or string value indicating the severity of the log message. The following list includes both the numeric and string values, only one or the other of which should be used for this attribute:</p> <ul style="list-style-type: none"> • 1 - critical • 2 - high • 3 - mediumHigh • 4 - medium • 5 - mediumLow • 6 - low • 7 - veryLow 	String	N/A	Yes
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<script-pause>

The <script-pause> element will pause playback of the test script, displaying a popup dialog within the Agentry Test Environment. The playback will not resume until this popup message is acknowledged.

Structure

Contains:

- None

Contained By:

- <script>

Attributes: None

Agentry Test Script: Button Elements Overview

The button-related elements available in the Agentry Test Script structure can be used to simulate a user pushing a button on a screen as a part of a sequence of interactions. Additionally, these elements can be used to test the state of the button, such as enabled or

disabled, its label value, and so forth, and to wait for the button to be enabled for attempting to push it.

Common Button Element Attributes

The following attributes are available to all button elements within the Agentry Test Script language:

Name	Description	Data Type	Default Value	Required
id	The identifier of the button, typically set by the script recorder in the ATE and not modified manually. If specified, the <code>name</code> and <code>label</code> cannot be present.	String	N/A	No
name	The resource name of the button the element affects or monitors. If specified, the <code>id</code> and <code>label</code> attributes cannot be present.	String	N/A	No
label	The label text of the button definition the element affects or monitors. If specified, the <code>id</code> and <code>name</code> attributes cannot be present.	String	N/A	No

<button-expect>

The `<button-expect>` element is used to verify the state of a button definition within a screen. This includes the button's label text, enabled state, checked state, whether or not it is a popup button, and whether or not it is visible. The type of button definition checked by this element will dictate the supported states and other expected values for the button definition.

If any of the configured expected state information is not matched by the button, a script error is thrown.

Structure

Contains:

- Text - The expected label for the button definition as displayed on the screen.

Contained By:

- `<script>`

Table 13. Attributes

Name	Description	Data Type	De- fault Value	Re- quired
enabled	This attribute specifies the expected enabled state of the button. The value <code>t</code> is true, meaning the button is expected to be enabled. The value <code>f</code> is false, meaning the button is expected to be disabled.	string	<code>t</code>	No
checked	This attribute specifies the expected checked state of the button, which is either checked or unchecked. The value <code>t</code> is true, meaning the button is expected to be checked. The value <code>f</code> is false, meaning the button is expected to not be checked.	string	<code>t</code>	No
popup	This attribute specifies whether the button is expected to be an Action Button with a defined action of popup menu. The value <code>t</code> is true, meaning the button is expected to be a popup menu. The value <code>f</code> is false, meaning the button is not expected to be a popup menu.	string	<code>f</code>	No
visible	This attribute specifies whether the button is expected to be visible or not. The value <code>t</code> is true, meaning the button is expected to be visible. The value <code>f</code> is false, meaning the button is not expected to be visible.	string	<code>t</code>	No
common button attrib- utes	For <code><button-expect></code> these attributes specify the expected related items for each attribute.	N/A	N/A	N/A
common script attrib- utes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

`<button-push>`

The `<button-push>` element is used to push, or click, a button on the current screen. The button can be pushed in combination with the `Shift`, `Ctrl`, and/or `Alt` keys when in a `<script>` context as optional attributes to the element. If the button is disabled when this element is processed a script error is thrown.

In a `<field-popup>` context the element will push a button on the popup dialog displayed. The contents of the `<button-push>` element contain the label of the button to be pushed. In this context none of the attributes are supported.

Structure

Contains:

- In a `<script>` context any text contents ignored.
- In a `<field-popup>` context this element contains text that specifies the button to push in the popup screen. Valid contents include:
 - 0-9
 - +/-
 - . (decimal)
 - “Back,” “Clear,” or “Close”

Contained By:

- `<script>`
- `<field-popup>`

Table 14. Attributes - `<script>` context only

Name	Description	Data Type	De-fault Value	Re-quired
shift	This attribute specifies whether the <code>Shift</code> key should be held down in combination with the button push. The value <code>t</code> is true and the <code>shift</code> key will be held down. The value <code>f</code> is false.	string	<code>f</code>	No
ctrl	This attribute specifies whether the <code>Ctrl</code> key should be held down in combination with the button push. The value <code>t</code> is true and the <code>Ctrl</code> key will be held down. The value <code>f</code> is false.	string	<code>f</code>	No
alt	This attribute specifies whether the <code>Alt</code> key should be held down in combination with the button push. The value <code>t</code> is true and the <code>Alt</code> key will be held down. The value <code>f</code> is false.	string	<code>f</code>	No
check	This attribute specifies whether the control should be checked or unchecked. The value <code>t</code> is true and will result in the control being checked. The value <code>f</code> is false. This attribute is valid only for check box controls on built-in client screens.	string	<code>t</code>	No

Name	Description	Data Type	De-fault Value	Re-quired
common button attributes	For <code><button-push></code> these attributes specify the button to be pushed.	N/A	N/A	N/A
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

`<button-wait>`

The `<button-wait>` element will force the test script to wait for the specified button's state to change either from enabled to disabled, or from disabled to enabled, depending on the element's configuration. The script will wait until either the specified state change occurs or until the timeout value for the `<script>` or the `<button-wait>` element is reached. If the button's state does not change before the timeout has elapsed a script error will be thrown.

Structure

Contains:

- None

Contained By:

- `<script>`

Table 15. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
enabled	This attribute specifies whether to wait for the button to be enabled or disabled. The value <code>Ⓣ</code> is true and the element will wait until the button is enabled. The value <code>ⓕ</code> is false and the element will wait until the button is disabled.	string	<code>Ⓣ</code>	No
common button attributes	These attributes when set for the <code><button-wait></code> element specify the button to be monitored for a state changed.			

Name	Description	Data Type	Default Value	Required
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

Agentry Test Script: Field Elements Overview

The elements related to detail screen fields within the Agentry Test Script are provided to allow for interaction with the detail screen fields of the application. This includes field selection, entering values, testing for expected values, pushing or selecting fields with an edit type of button, and interacting with date, time, and duration related field types.

Common Field Element Attributes

The following list includes the common attributes for most field-related XML elements within the Agentry Test Script. These attributes are used by the elements in different ways depending on the purpose and behavior of the element.

Name	Description	Data Type	Default Value	Required
id	The identifier of the field definition, typically set by the script recorder in the ATE and not modified manually. If the <code>id</code> attribute is specified, the <code>name</code> and <code>label</code> attributes cannot be present.	String	N/A	No
name	The resource name of the field the element affects or monitors. If the <code>name</code> attribute is specified, the <code>id</code> and <code>label</code> attributes cannot be present.	String	N/A	No
label	The label text of the field definition the element affects or monitors. If the <code>label</code> attribute is specified, the <code>name</code> and <code>id</code> attributes cannot be present.	String	N/A	No

<field-button-push>

The `<field-button-push>` element will push the button control for a detail screen, provided that field's edit type includes a button control. Following is a list of the field edit types for which this element will push a button:

- Barcode Scan (when defined to include a scan button)

- Button
- List Tile View - Add, Edit, and Filter Buttons
- Complex Table Search
- Complex Table Drop Down
- Complex Table List
- Complex Table Tree
- Data Table Selection (displays drop down list or popup list view based on field's definition)
- External Data

Structure

Contains:

- None

Contained By:

- `<script>`

Table 16. Attributes

Name	Description	Data Type	De- fault Value	Re- quired
buttonLabel	This attribute contains the label text of the field's button that is to be pushed. Many field's button controls do not contain text, or the text changes based on theme. It is recommended that the <code>label</code> attribute be used to specify the label of the field containing the button control to be pushed.	string	none	No
buttonControl	This attribute specifies the control ID of the button control that is to be pushed. This is normally set by the Script Recorder within the ATE under specific circumstances and is normally not set when manually editing a test script.	string	none	No
common button attributes	The following common button attributes are a part of this element: <ul style="list-style-type: none"> • <code>name</code> • <code>label</code> • <code>id</code> 	N/A	N/A	N/A

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

<field-expect>

The <field-expect> element allows for the validation of a detail screen field's value and state. State information includes hidden or visible, and enabled or disabled. The contents of the field that can be validated include its current value, for duration fields the current value of each portion of the duration value (hours, minutes, seconds), for list fields the total number of rows in a list, or the current value of a row or rows within a list. When validating the value of a field, the validation attribute should be set indicating the type of validation. To validate multiple facets of a field, such as current value and enabled or disabled, multiple <field-expect> elements are required to validate each facet. To validate the value of multiple rows within a list, the <field-expect> element must contain on <row> element for each row to be validated. Note that only drop down lists may be validated by this element. For list view or list tile view fields, the <list-expect> element must be used. If the field does not match the expected criteria as specified by the <field-expect> element, a script error is thrown.

Structure

Contains:

- Text - The value that the field is expected to contain, or the value of the state being checked by the element.
- <row>

Contained By:

- <script>

Table 17. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
type	<p>This attribute specifies what is being checked for within the field by this element. Valid values include:</p> <ul style="list-style-type: none"> • string - The field's string value • bool - The field's Boolean value. • long - The field's integral number value. • decimal - The field's decimal number value. • validateValue - The field's value. • enabled - The field's enabled or disabled state. • visible - The field's visible or hidden state. • list - The field is a drop down list. • format - The label for a button field. 	string	string	No
special	<p>This attribute specifies that the field's value should or should not be equal to its defined special value. The value <code>Ⓣ</code> is true and the field is expected to be set to its special value. The value <code>Ⓕ</code> is false.</p>	string	f	No
part	<p>This attribute is valid only when checking a duration field. It specifies the portion of the duration value to be checked. Valid values for this attribute include:</p> <ul style="list-style-type: none"> • hours - The hours portion of the duration. • decimalHours - The hours portion of the duration as a decimal. • minutes - The minutes portion of the duration. • seconds - The seconds portion of the duration. 	string	none	Re-quired for duration field edit types. Otherwise ignored.

Name	Description	Data Type	Default Value	Required
row	The row index to be checked by the element when the field is a drop down list. This index is 0-based, meaning the first row in the list is at position 0. This attribute is only valid when the edit type of the field is Complex Table Drop Down, Data Table Selection, or List Selection. If the <code><field-expect></code> contains one or more <code><row></code> elements, the first row checked is indicated by this attribute. Additional <code><row></code> elements are expected to be contained in the list in the order in which their corresponding <code><row></code> elements are contained in the <code><field-expect></code> .	non-negative integer	none	Required for drop down list fields. Otherwise ignored.
count	This attribute specifies the expected number of rows in the list.	non-negative integer	none	No
common field attributes	This element contains the following common field attributes: <ul style="list-style-type: none"> name id label 	N/A	N/A	N/A
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

`<field-label-select>`

The `<field-label-select>` element allows for selecting, or “clicking”, the label of a field when that field label is defined as a hyperlink. The field’s label to be selected is specified using one of the `name` or `label` attributes. If the field’s label is not a hyperlink, or if it cannot be selected for some other reason (e.g. the action it executes is disabled) a script error will be thrown.

Structure

Contains:

- None

Contained By:

- `<script>`

Table 18. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common field attributes	This element includes the following common field element attributes: <ul style="list-style-type: none"> • name • label • id 			
common script attributes	A set of related attributes common to most elements within the Agency Test Script.	N/A	N/A	N/A

`<field-popup>`

The `<field-popup>` element will open the numeric popup screen that allows for the entry of numeric values into a detail screen field. This element is valid for fields with an edit type of Decimal Number, Integral Number, or Duration. This element can contain the `<edit-select>` and `<button-push>` elements to select values in the popup screen and to push the buttons on the popup to enter values, respectively. If this element is used for field with an edit type other than those it supports a script error will be thrown.

Structure

Contains:

- `<edit-select>`
- `<button-push>`

Contained By:

- `<script>`

Table 19. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
part	This attribute specifies the portion of a duration field for which the popup screen will be displayed. This attribute is ignored for other field edit types. Valid values for this attribute include: <ul style="list-style-type: none"> • hours • minutes • seconds 	string	none	Required for Duration fields. Otherwise ignored.
common field attributes	This element includes the following common field attributes: <ul style="list-style-type: none"> • name • label • id 	N/A	N/A	N/A
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

<edit-select>

The <edit-select> element selects the numeric values in a field by position. This element is contained by the <field-popup>, which specifies the field and, for duration fields, the part of the field in which the selection is made. The <edit-select> element selects the characters based on position within the field as specified by its `start` and `end` attributes. The first character is at position 0. All characters from the specified `start` up to and including the `end` character are selected.

Structure

Contains:

- None

Contained By:

- <field-popup>

Table 20. Attributes

Name	Description	Data Type	Default Value	Required
start	The first character within the field to select. Characters are specified using a 0-based index, meaning the first character is at position 0 within the field.	non-negative integer	0	Yes
end	The last character within the field to select. Characters are specified using a 0-based index, meaning the first character is at position 0 within the field.	non-negative integer	0	Yes

<field-set>

The `<field-set>` element sets the value of a detail screen field. The contents of the field specify the value to be set. The attributes name or label are used to specify which field to set. Other attributes can be used to set the value of the field to its defined special value, to check or uncheck a check box field, to select or deselect a radio button field, or to set just a portion of a date, time, date and time, or duration field. This element is used to set the value of most fields, regardless of field type. Other elements exist to allow for setting field values but should only be employed in less common situations. For most testing purposes the `<field-set>` element is sufficient. It is the element inserted into a test script generated by the Test Script Recorder within the Agentry Test Environment.

Structure

Contains:

- Text - The value to which the field will be set.

Contained By:

- `<script>`

Table 21. Attributes

Name	Description	Data Type	Default Value	Required
special	This attribute specifies whether the field should be set to its defined special value. The value <code>t</code> is true and the field will be set to its defined special value. The value <code>f</code> is false.	Boolean	f	No

Name	Description	Data Type	Default Value	Required
part	<p>This attribute specifies the portion of the field to set. This attribute is only valid for fields with an edit type of Date, Time, Date and Time, or Duration. Valid values for this attribute include:</p> <ul style="list-style-type: none"> • year • month • day • hours • minutes • seconds 	string	none	Required for date, time, and duration fields. Otherwise ignored.
checked	<p>This attribute specifies whether or not to check or select a field with an edit type of Button that is either a check box or radio button. The value \mathfrak{t} is true and will check or select the box or radio button. The \mathfrak{f} is false and will uncheck or deselect the button field.</p>	Boolean	f	Required for Button fields of type radio or check box. Otherwise ignored.
common field attributes	<p>This element includes the following common field attributes:</p> <ul style="list-style-type: none"> • name • label • id 	N/A	N/A	N/A
common script attributes	<p>A set of related attributes common to most elements within the Agentry Test Script.</p>	N/A	N/A	N/A

Agentry Test Script: List Elements Overview

The list-related elements of the Agentry Test Script allow for interaction with the various list controls of the Agentry Client. These include list screens, and the various list field edit types that can be defined for detail screens. The list-related elements support the selection of items

in a list, double-clicking items, checking for the expected values of list items and column headers, expected values of list headers and detail panes.

Common List Element Attributes

The following attributes are found in most of the list-related elements of the Agentry Test Script. The purpose and use of these attributes depends on the nature of the element for which they are set.

Name	Description	Data Type	Default Value	Required
id	The identifier of the list definition, typically set by the script recorder in the ATE and not modified manually. If the <code>id</code> attribute is specified, the <code>name</code> and <code>label</code> attributes cannot be present.	String	N/A	No
name	The resource name of the list the element affects or monitors. If the <code>name</code> attribute is specified, the <code>id</code> and <code>label</code> attributes cannot be present.	String	N/A	No
label	The label text of the list (if applicable) the element affects or monitors. If the <code>label</code> attribute is specified, the <code>name</code> and <code>id</code> attributes cannot be present.	String	N/A	No
row	The row within the list to be affected by the element. This may be the row number, with the first row in the list at position zero (0), or one of the values: <ul style="list-style-type: none"> selected - currently select row first - first row in the list last - last row in the list next - the next row after the currently selected one previous - the previous row after the currently selected one 	String	N/A	No

<list-double-click>

The `<list-double-click>` element allows for an item to be double-clicked within a list. The item to be double-clicked may be the currently selected item, or this element can specify the item. To specify an item the row number can be used or text can be specified to select the item. If a screen contains multiple lists the field can be found by specifying the name or label for the the field definition. This element can be used with list screens to double-click an item in

the list control on the screen, or with detail screens containing a field with an edit type of List View.

Structure

Contains:

- None

Contained By:

- `<script>`

Table 22. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common list attributes	This element includes the following common list attributes: <ul style="list-style-type: none"> • name • id • label • row 	N/A	N/A	N/A
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

`<list-expect>`

The `<list-expect>` element verifies the contents of a list control on a list screen. It can also verify a detail screen field with an edit type of List View. This element can verify the contents of the list, including the number of rows display, the number of selected rows, the values displayed in each column for each row, the presence of a value in a column in any row, the contents of the list's header label and detail pane, and the contents of the column header on each column within the list. When used on a detail screen containing multiple lists, the specific list to verify can be found using the field's name or label. If the list does not meet the expected criteria a script error is thrown. The expected values and state of the list and its items can be specified using the elements attributes as well as the elements it can contain.

Structure

Contains:

- Text - The value to use to locate the desired row to verify within the list. Text content and element content are mutually exclusive for the `<list-expect>` element.

- `<row>`
- `<column>`
- `<header>`
- `<columnheader>`
- `<detail>`

Contained By:

- `<script>`

Table 23. Attributes

Name	Description	Data Type	De- fault Value	Re- quired
column	This attribute specifies the column to be verified by the <code><list-expect></code> element. This may be either the column's header label or a numeric value indicating the columns position from left to right, with the left-most column at position 1.	string	None	No
count	This attribute specifies the total number of rows the list should contain. This may any numeric integer no less than zero.	non-nega- tive inte- ger	None	No
selected- count	This attribute specifies the total number of rows currently selected in the list. This may any numeric integer no less than zero.	non-nega- tive inte- ger	None	No
common list attributes	This element includes the following common list attributes: <ul style="list-style-type: none"> • name • id • label • row • column 	N/A	N/A	N/A
common script attrib- utes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

`<list-select>`

The `<list-select>` element will select an item in a list. This element can be used to select items in the list control of a list screen, in detail screen fields with an edit type of List View, or when the edit type of a detail screen field is List Tile View. This element may also be used to

deselect an already selected item in the list. List items can be selected based on row position or column value. The contents of the `<list-select>` element will be the value to search for in a specified column when select by column value. Of the row cannot be found or selected in the list a script error is thrown.

Structure

Contains:

- Text - The value of a column by which the item will be found and selected.

Contained By:

- `<script>`

Table 24. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
column	This attribute specifies the column by which the item will be found. This may be either the columns header label text or a numeric value specifying the column's position from left to right, with the left-most column at position 1. This attribute is set to "none" for List Tile View detail screen fields.	String	None	Re-quired when the <code>row</code> attribute is set to a value of "text."
select	This attribute specifies whether to select or deselect the row. The value <code>t</code> is true and will result in the row being selected. The value <code>f</code> is false and the row will be deselected.	String	<code>t</code>	No
common list attributes	This element includes the following common list attributes: <ul style="list-style-type: none"> • name • id • label • row 			
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

<list-sort-by>

The <list-sort-by> element will sort a list by a specified column. This element can sort the items of a list control on a list screen, or the items in detail screen field with an edit type of list view. The column to sort the list on is specified in the elements column attribute. If the specified column cannot be found or if the list cannot be sorted on the column a script error is thrown.

Structure

Contains:

- None

Contained By:

- <script>

Table 25. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
column	This attribute specifies the column upon which the list should be sorted. This may be either the column header label text or the position of the column from left to right, with the left-most column at position 1.	string	None	Yes
common list attributes	This element includes the following common list attributes: <ul style="list-style-type: none"> • name • id • label 	N/A	N/A	N/A
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

<detail>

The <detail> element can be contained in a <list-expect> element. When present the <detail> element must then contain a <text> element. The contents of the <text> element is then the text expected to be found in the detail pane of a list screen or a detail screen field with an edit type of List View. If the contents of the detail pane do not match the contents of the <text> element a script error is thrown.

Structure

Contains:

- `<text>` - This element contains the text expected to be found in the detail pane of the list.

Contained By:

- `<list-expect>`

Attributes: None

<header>

The `<header>` element can be contained in a `<list-expect>` element and, when present must contain a `<text>` element. The contents of the `<text>` element is then the text expected to be found in the lists header label. This element is valid only when the `<list-expect>` element containing it is for a list screen or a detail screen field with an edit type of List View. If the header pane does not match the contents of the `<text>` element a script error is thrown.

Structure

Contains:

- `<text>` - This element contains the expected text in the lists header label.

Contained By:

- `<list-expect>`

Attributes: None

<columnheader>

The `<columnheader>` element verifies the label displayed on a column header in a list. This element is contained in a `<list-expect>` element and, when present must contain a `<text>` element. The contents of the `<text>` element is the text expected to be in the label of the column header. This element can be used to verify header labels for columns in the list control of a list screen or the columns in a detail screen field with an edit type of List View. If the header of the specified column does not match the contents of the `<text>` element the or if the specified column cannot be found a script error is thrown. The column to verify is specified using the `<columnheader>` attribute `column`.

Structure

Contains:

- `<text>`

Contained By:

- `<list-expect>`

Table 26. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
column	This attribute can specify the column header to be verified as a part of the parent <code><list-expect></code> element processing. The contents of this attribute is either the column header label or the column index, with the left-most column at position 1.	string	none	No

`<row>`

The `<row>` element verifies the contents and state of a row in a list control. This element can be contained by the `<field-expect>` and `<list-expect>` elements. Multiple `<row>` elements may be contained by the same parent element to verify multiple rows in the same list. The `<row>` element can specify the row to be verified, or this may be specified by the parent element's row attribute. The contents of the row can include the text value indicating the expected contents of the row.

The `<row>` element may also contain one or more `<column>` elements. Each `<column>` element will verify the expected contents and/or state of the column it identifies within the row identified by the `<row>` element.

Structure

Contains:

- `<column>`

Contained By:

- `<field-expect>` - Only contained by this element when verifying the contents of a field with an edit type that displays a list control on the detail screen.
- `<list-expect>`
- Text - The expected contents of the row.

Table 27. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
row	This attribute contains the row position within the list. The top-most row is at position 1. Note that the same list may have a different row at the same numeric position based on sorting of the list.	positive integer	None	No
selected	This attribute specifies the expected selected state of the row. This is a Boolean value. The value <code>t</code> is true and the row is expected to be selected. The value <code>f</code> is false and the row is expected to not be selected.	Boolean	none	No

<menu-expect>

The `<menu-expect>` element verifies the contents and state of a menu. This element must contain at least one `<menu>` element specifying the menu to be verified. The `<menu>` element itself will likely contain other elements regarding the items within the menu.

Structure

Contains:

- `<menu>`

Contained By:

- `<script>`

Table 28. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep • frame 	N/A	N/A	N/A

<menu-select>

The <menu-select> element is used to select a menu item. This element must contain a single <menu> element, which in turn must contain a single <item> element. These elements specify the menu and item to be selected by the <menu-select> element.

Structure

Contains:

- <menu>

Contained By:

- <script>

Table 29. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none">• timeout• sleep• frame	N/A	N/A	N/A

<menu>

The <menu> element is contained by <menu-select> and <menu-expect>. The <menu> element identifies the menu to be acted on by the containing element. When contained by a <menu-expect> this element specifies what type of verification is to be performed. The <menu> element will contain one or more <item> elements identifying the menu item to be selected or verified. The <menu> element may also contain one or more <separator> elements when contained by a <menu-expect> element. The <separator> element will indicate the expected position of a menu separator.

Structure

Contains:

- Text - The name of the menu to be acted on by this element.
- <item>
- <separator>

Contained By:

- `<menu-select>`
- `<menu-expect>`

Table 30. Attributes

Name	Description	Data Type	De-fault Value	Required
type	<p>This attribute is only used when the <code><menu></code> element is contained by a <code><menu-expect></code> element. The type attribute specifies what is to be verified within the named menu. Valid options for this attribute include:</p> <ul style="list-style-type: none"> • exact - The <code><item></code> and <code><separator></code> elements contained by the <code><menu></code> element must match exactly with the contents of the menu. • sub-set - The <code><item></code> and <code><separator></code> elements contained by the <code><menu></code> element must exist within the menu, but others may also be present. • no separators - The <code><item></code> elements contained by the <code><menu></code> element must exist within the menu. Any separators within the menu are ignored and the <code><menu></code> element should contain no <code><separator></code> elements. 	string	exact	No - valid only when <code><menu></code> is contained by a <code><menu-expect></code>

`<item>`

The `<item>` element is contained by the `<menu>` element, which in turn can be contained by a `<menu-expect>` or `<menu-select>` element. When the ancestor element is a `<menu-expect>` the `<item>` element specifies the expected state and value of the menu item identified by the `<menu>` element. When the ancestor is a `<menu-select>` the `<item>` element specifies the menu item to be selected. For both use cases the contents of the `<item>` element is the name of the menu item.

When the ancestor of the `<item>` element is a `<menu-expect>` the `<item>` element includes attributes to specify the expected state and other information about the named menu item. These include the enabled/disabled state and whether or not the menu item is checked (selected). These attributes are ignored when the ancestor element is a `<menu-select>`.

In a `<menu-expect>` context, if the expected state of the menu item does not match the attributes of the `<item>` attribute a script error is thrown. In a `<menu-select>` context if the menu item cannot be selected a script error is thrown

Structure

Contains:

- Text - The name of the menu item to be acted on.

Contained By:

- `<menu-select>`
- `<menu-expect>`

Table 31. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
checked	This attribute is only valid when the ancestor element is a <code><menu-expect></code> . This is a Boolean attribute. The value <code>t</code> is true and indicates the menu item is expected to be checked. The value <code>f</code> is false and indicates the menu item is not expected to be checked.	Boolean	None	No
enabled	This attribute is only valid when the ancestor element is a <code><menu-expect></code> . This is a Boolean attribute. The value <code>t</code> is true and indicates the menu item is expected to be enabled. The value <code>f</code> is false and indicates the menu item is expected to be disabled.			

Agentry Test Script: Tree Elements Overview

The tree-related elements of the Agentry Test Script are used to work with tree controls presented on the Agentry client. This includes any detail screen fields that present a tree control. The tree-related elements should be used in place of the field-related elements for fields that present a tree control.

The elements in this group can be used to select nodes within a tree control, expand and collapse nodes, double-click nodes, a check for expected values of a node within the tree control.

Common Tree Element Attributes

The following list contains attributes found in most of the tree-related elements of the Agentry Test Script. The elements use these attributes differently depending on the purpose and behavior of the element.

Name	Description	Data Type	Default Value	Re-quired
name	The resource name of the tree control. If name is specified, label cannot be present.	String	N/A	No
label	The label of the tree control. If the label is specified, name cannot be present.	String	N/A	No
node	The node within the tree control to be affected by the element. The value of the attribute is the display value of the node in the tree control.	String	N/A	No
sibling	The sibling node to the currently selected node in the control. This attribute is set to the expected value of the node in the tree control.	String	N/A	No
child	The child node to the currently selected node in tree control. The value of this attribute is set to the expected value of the child node in the tree control.	String	N/A	No

<tree-select>

The <tree-select> element will select the specified node in a tree control. The node selected can be specified by its relationship to the currently selected node in the tree control, optionally in combination with that node's currently displayed text value.

Structure

Contains:

- Text - The text displayed for the tree node.

Contained By:

- <script>

Table 32. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common tree control attributes	This element includes the following common tree control attributes: <ul style="list-style-type: none"> • name • label • node • sibling • child 	N/A	N/A	N/A
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<tree-expect>

The <tree-expect> is used to verify the expected state and values of a tree control node. This element will contain one or more <node> elements. The <tree-expect> node specifies the node to be verified. Information the tree-expect node will verify includes the existence of the specified node and the total count of child nodes it contains. The <node> elements contained in a <tree-expect> then specify the expected child nodes of that node. The <tree-expect> node specifies the type of verification to perform for the specified node in the tree control. This options are to verify the child nodes match those <node> elements within <tree-expect> exactly, or to verify the child nodes include those <node> elements within the <tree-expect>. In the case of the latter verification, other child nodes may exist in the tree control. The first verification is referred to as an “exact” verification type. The latter is a “sub-set” verification. If the selected tree control node does not match the parameters of the <tree-expect> element, a script error is thrown.

Structure

Contains:

- Text - The expected contents of the specified node in the tree control.
- <node>

Contained By:

- <script>

Table 33. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
count	This attribute can specify the expected number of child nodes for the specified node being verified. If this attribute is not provided, the count of child nodes is not performed or checked.	non-negative integer	N/A	No
type	This attribute specifies the type of verification to perform. Valid values for this attribute are: <ul style="list-style-type: none"> exact - All child nodes for the tree control node must exactly match all <node> elements contained in the <tree-expect> element. sub-set - For each <node> element contained in the <tree-expect> element there must be a matching child node in the tree control. Additional child nodes may also exist. 	string	sub-set	No
common tree control attributes	This element includes the following common tree control attribute: <ul style="list-style-type: none"> label node sibling child 	N/A	N/A	N/A
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> timeout sleep 	N/A	N/A	N/A

<node>

The <node> element is contained by the <tree-expect> element. Multiple <node> elements may exist within the same <tree-expect>, with each representing an expected child node in the tree control. The contents of the <node> element is the text expected to be displayed for the node. If a <node> element does not match a corresponding node in the tree control, the containing <tree-expect> node will throw a script error. The order of the <node> elements must match the order of the expected nodes within the tree control.

Structure

Contains:

- Text - The expected value displayed for the child node in the tree control.

Contained By:

- `<tree-expect>`

Attributes: None

`<tree-expand>`

The `<tree-expand>` element will expand the currently selected node in a tree control. Optionally, a node can be specified, in which case the specified node will be selected and expanded by the element. If the node is already expanded this element will have no affect. If the specified node cannot be selected a script error is thrown.

Structure

Contains:

- Text - The currently displayed text of the tree control node to expand.

Contained By:

- `<script>`

Table 34. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common tree control attributes	<p>This element includes the following common tree control attributes:</p> <ul style="list-style-type: none"> • label • node • sibling • child 	N/A	N/A	N/A
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<tree-collapse>

The <tree-collapse> element will collapse a currently expanded node in a tree control. Optionally, the element can specify the node to be collapsed, in which case the node will first be selected and then collapsed. If a node in the tree control is not specified, the currently selected node will be collapsed. If the node is currently collapsed, this element will have no affect. If the specified tree control node cannot be selected a script error will be thrown. Note that if a node in the tree control exists but is currently hidden because its parent or ancestor node is not expanded, the <tree-collapse> node will not be able to select the node.

Structure

Contains:

- Text - The currently displayed value of the node to be collapsed.

Contained By:

- <script>

Table 35. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common tree control attributes	This element includes the following common tree control attributes: <ul style="list-style-type: none"> • label • node • sibling • child 	N/A	N/A	N/A
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<tree-toggle>

The <tree-toggle> element will toggle the specified node in a tree control; i.e., the specified node will be expanded if currently collapsed, or collapsed if currently expanded. If the specified node cannot be found in the tree control a script error will be thrown.

Structure

Contains:

- Text - The displayed text of the tree control node to be toggled

Contained By:

- `<script>`

Table 36. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common tree control attributes	<p>This element includes the following common tree control attributes:</p> <ul style="list-style-type: none"> • label • node • sibling • child 			
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

`<tree-double-click>`

The `<tree-double-click>` element will double-click a node in the tree control. Optionally the node to double-click can be specified. Otherwise the currently selected node will be double-clicked. The node to double-click must currently be visible in the tree control. If the specified node cannot be selected a script error will be thrown.

Structure

Contains:

- Text - The currently displayed text for the tree control node to be double-clicked.

Contained By:

- `<script>`

Table 37. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common tree control attributes	This element includes the following common tree control attributes: <ul style="list-style-type: none"> • label • node • sibling • child 	N/A	N/A	N/A
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<tree-count-visible>

The <tree-count-visible> element will verify the number of currently visible nodes in a tree control. This count is based on the nodes a user can currently see on the client's screen, not the total number of nodes in the tree control. If the specified number of nodes is not currently visible in the tree control a script error is thrown.

Structure

Contains:

- Text - A non-negative value specifying the number of nodes expected to be currently visible in the tree control.

Contained By:

- <script>

Table 38. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common tree control attributes	<p>This element includes the following common tree control attributes:</p> <ul style="list-style-type: none"> • label • node • sibling • child 	N/A	N/A	N/A
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

Agentry Test Script: Scanner Elements Overview

The scanner-related elements within the Agentry Test Script are used to test barcode scanner functionality. Included in this group of elements are those to start a barcode scan, set the scan value, and enable and disable the scanner simulator within the ATE.

It is important to note that these elements work directly through the barcode Scanner simulator within the Agentry Test Environment. The `<scan-enable>` element should be called to enable this simulator prior to calling the `<scan-start>` element, which passes the barcode value into the simulator, which in turn passes the value to the test client.

Alternately, if barcode scanner functionality is to be use, the barcode scanner can be enabled within the ATE prior to executing the test script. This will negate the need to enable it from within the script, which also requires the test client to be restarted.

<scan-data>

The `<scan-data>` element contains the default scan data to be passed by the Agentry Test Environment's scan simulator to the test client. This element does not perform the actual scan and does not directly pass the data to the test client. Rather, this element contains the data to be passed to the test client when a scan is called for. The element `<scan-start>` will perform the actual scan and can contain a value to pass to the test client. If `<scan-start>` does not contain a value, and if it is preceded by a `<scan-data>` element, the value in `<scan-data>` will be the one passed to the test client.

Structure

Contains:

- Text - The default value to pass to the test client from the ATE's scan simulator.

Contained By:

- `<script>`

Table 39. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep • frame 	N/A	N/A	N/A

`<scan-enable>`

The `<scan-enable>` element will enable or disable the scan simulator in the Agentry Test Environment. When this element is processed it should be followed by a `<restart-client>` element in the test script, as enabling or disabling the scan simulator requires the test client to be restarted. The contents of this element can be the text values `true` or `false`, where `true` will enable the scan simulator and `false` will disable it.

Structure

Contains:

- Text - The `true` or `false` value to enable or disable the scan simulator in the Agentry Test Environment.

Contained By:

- `<script>`

Table 40. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep • frame 	N/A	N/A	N/A

<scan-start>

The <scan-start> element will pass the value it contains to the test client through the Agentry Test Environment's scan simulator. Alternately, the value passed to the test client can be contained in a separate <scan-data> element. To pass this default value the proper syntax for the <scan-start> element is:

```
</ags:scan-start> -- OR -- </scan-start>
```

The use of an open and close marker for this element, such as: <scan-start></scan-start> will result in an error during test script playback.

Structure

Contains:

- Text - Option text value to be passed to the test client through the Agentry Test Environment's scan simulator.

Contained By:

- <script>

Table 41. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep • frame 	N/A	N/A	N/A

Agentry Test Script: SQL Elements Overview

The SQL elements within the Agentry Test Script are provided to allow for the interrogation of the back end system processing built into the mobile application from within the test script. These elements do not affect the test client in any way. Rather, they are added to a test script to connect to and run queries against a database. Values returned by these queries can then be checked for expected values to validate the proper back end processing of the mobile application.

The primary use case for these elements is to check the proper processing of the mobile application's transactions. Typically these statements would be added to the script after a transmit is performed within the script that includes sending transactions to the Agentry Server for processing.

However, the SQL elements can be added at any point within the script where it is desirable to check the data in the back end as it relates to the mobile applications synchronization processing. Updates or changes made to exchange information and similar data can certainly be verified using these elements if desired.

The SQL elements are not added to the test script by the script recorder, but rather should be manually added to the script once it has been created.

Note that these elements are intended only for connecting directly to a database server. However, they can be used in testing an application with other types of system connections, provided there is direct access available to the database that may be behind the Java or Web Service interface.

<dsn-create-sql>

The <dsn-create-sql> element will create a System Data Source Name in ODBC representing a connection to a SQL Server database. This is a permanent DSN added to the host system and is not needed if an existing System DSN is already present. This element includes attributes specifying the DSN name, the database instance, SQL Server host, and authentication method (SQL or Windows). Any other parameters for a DSN are set to their defaults as configured on the host system. When using this element within a script, it is recommended that a corresponding <dsn-remove-sql> element exists to remove the same DSN created to allow for the same test script to be played multiple times on the same host system. If the DSN cannot be created a script error is thrown.

Structure

Contains:

- None

Contained By:

- <script>

Table 42. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
dsn	This attribute specifies the name given to the DSN. This value can contain no whitespace and must adhere to all requirements of a DSN name as specified in ODBC.	string	None	Yes
server	This attribute specifies the host system upon which the SQL Server service is running.	string	None	Yes
database	This attribute specifies the database instance to which the DSN should connect.	string	None	Yes
authentication	<p>This attribute specifies the authentication method to be used to create a connection the SQL Server service and database. This may be one of:</p> <ul style="list-style-type: none"> • SQL - The SQL Server authentication method. Connections made using this DSN will require the user login and password of a user account configured in the SQL Server service with permissions to access the database instance. • WINDOWS - The integrated Windows authentication method. The test script that uses the DSN created must be run as a Windows user known to the SQL Server system and with permissions to access the target database. 	string	None	Yes
description	This attribute can contain a text value that will be added to the System DSN as a description.	string	None	No
overwrite	This attribute specifies whether or not to overwrite an existing System DSN with the same name as the one this element will created. This is a Boolean value, with true indicating the existing DSN should be overwritten and false indicating it should not.	Boolean	False	No

Name	Description	Data Type	De- fault Value	Re- quired
ignoreErrors	This attribute specifies whether or not any errors returned when creating the System DSN should be ignored. This is a Boolean value, with true indicating errors should be ignored and false indicating they should not be ignored. If this attribute is false or not specified, any error returned when attempting to create the System DSN will result in a script error being thrown.	Boolean	False	No
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<dsn-remove-sql>

The `<dsn-remove-sql>` element is used to remove an ODBC System Data Source Name for a SQL Server database. The name of the DSN to be removed is specified in the `dsn` attribute to the element. Note that this element will permanently remove the DSN and will not be possible to recover it once this element has been processed.

Structure

Contains:

- None

Contained By:

- `<script>`

Table 43. Attributes

Name	Description	Data Type	De- fault Value	Re- quired
dsn	The name of the ODBC System Data Source Name to be removed. The DSN named here must be for a SQL Server connection.	string	None	Yes

Name	Description	Data Type	De-fault Value	Re-quired
ignoreErrors	This attribute specifies whether or not any errors returned with the attempt to delete the DSN should be ignored. This is a Boolean value with true indicating errors should be ignored. If this attribute is false or not specified, errors will not be ignored and, if any are returned, a script error will be thrown.	Boolean	False	No
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<sql-command>

The <sql-command> element will execute a SQL command against a database system. This element must be contained by a <sql-connect> element that creates a connection to the database system against which the SQL command will be executed. The <sql-command> element can contain the SQL statement to be executed, or it may reference a text file containing the command or commands to be executed. Results from executing the commands against a database can be logged based on the settings of the containing <sql-connect> element. The <sql-command> element can contain a <sql-expect> element that can verify the return set of any SELECT statement run by the <sql-command> element. Based on the <sql-command> element's ignoreErrors attribute setting, errors in executing the SQL command can be ignored or not. If errors are not ignored (default behavior) a script error will be thrown in the event an error occurs in executing the SQL statement. The <sql-command> element must either reference a file in its commandFile attribute containing the SQL command(s) to execute, or the text contents of the element itself must include a single SQL command.

Structure

Contains:

- Text - The SQL command to be executed against the database. Should not be present if a file is referenced in the commandFile attribute.
- <sql-expect>

Contained By:

- <sql-connect>

Table 44. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
command-File	This attribute can contain the full path and file name, including file extension, of a plain text file containing the SQL command(s) to be executed against the database. This attribute should not be specified if the contents of the <code><sql-command></code> element include the SQL command to be executed.	string	none	No
ignoreErrors	This attribute specifies whether or not errors returned when attempting to execute the SQL command should be ignored. This is a Boolean value. When true errors will be ignored. When false errors will not be ignored and a script error will be thrown when an error occurs.	Boolean	False	No
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

`<sql-connect>`

The `<sql-connect>` element will open a connection to a database server using the specified connection protocols and user credentials. The contents of this element include the `<sql-command>` element, which contains the SQL command to execute against the database with which the `<sql-connect>` element opens a connection. This connection will be closed after the processing of the `<sql-connect>` element and its contents has completed. In order for this element open a connection the host system upon which the test script is being played must have the proper configuration in place for the connection. For example the ODBC System DSN for a SQL Server database, or the TNS Name for an Oracle database connection. The attributes of this element specify whether or not to save the results of execution of any SQL commands to a log file.

Structure

Contains:

- `<sql-command>`

Contained By:

- `<script>`

Table 45. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
dbConnection-Name	The name of the connection configuration component for the target database; e.g. DSN, TNS Name, etc.	String	None	Yes
dbConnectionType	<p>The type of connection to make. This must match the connection configuration component type named in dbConnectionName. Valid values for this attribute include:</p> <ul style="list-style-type: none"> • DB2 • Informix • Interbase • MsSQL • ODBC (recommended setting for SQL Server connections) • Oracle • Postgre • SQLBase • SQLServer • Sysbase 	String	None	Yes
dbConnectionUserID	The user ID to connect to the database system.	String	None	Yes
dbConnection-Password	The password for the user ID.	String	None	Yes
commandFile	The full path and file name, including extensions, of a plain text file containing the SQL command(s) to execute once connected. This attribute is optional and should if used when one or more <sql-command> elements are contained in the <sql-connect> element, will be executed before those commands.	String	None	No

Name	Description	Data Type	De-fault Value	Re-quired
saveQuery	This attribute specifies whether or not to log the SQL command executed against the database. This is a Boolean value, with true indicating the query should be saved. This attribute is ignored if saveFile is not present.	Boolean	False	No
saveResult	This attribute specifies whether or not to log the result of executing the SQL command(s) contained within the <sql-connect> element or any <sql-command> elements it contains. This is a Boolean value, with true indicating the results should be saved. Note that results are the responses of the database system, not the data returned by a query. This attribute is ignored if saveFile is not present.	Boolean	False	No
saveFile	This attribute can contain the full path and file name to which log messages from any SQL commands will be written. If this attribute is not set saveQuery and saveResult are ignored.	String	False	No
ignoreErrors	This attribute specifies whether or not any errors generated by the <sql-connection> element's operations should be ignored. This is a Boolean value, with true indicating errors should be ignored. If not set or set to false (default) any errors resulting from attempting to connect to the database, or in executing a SQL command, will result in a script error being thrown. Any <sql-command> elements contained in the <sql-connect> element can override this setting.	Boolean	False	No
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<sql-expect>

The <sql-expect> element is used to verify the return set from a SELECT statement executed from within a <sql-command> element. The <sql-expect> is contained within the <sql-command> that executed the command. It can then specify the expected number of records in the return set. The <sql-expect> in can also contain one or more <sql-row> elements, each of which will contain one or more <sql-column> elements. The <sql-row> and <sql-column> elements will verify the expected contents of the return set of the SELECT statement. The <sql-expect> element can also specify if the order of the return set should match the order of the <sql-row> elements it contains, or merely verify the existence of the same records, regardless of order. If the return set does not match the expected return a script error is thrown.

Structure

Contains:

- <sql-row>

Contained By:

- <sql-command>

Table 46. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
count	This attribute can specify the expected number of records, or row, in the return set. If this attribute is not specified, the number of rows is not factored into the validation of the return set.	non-negative integer	none	No
strictOrder	This attribute specifies whether or not the order of the rows in the return set is expected to match the order of <sql-row> elements contained in the <sql-expect> This is a Boolean attribute, with true specifying the order must match, and false specifying it does not. If false is specified, the rows in the return set can be in any order related to the order of <sql-row> elements. However, all <sql-row> elements must have corresponding <records>.			

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<sql-row>

The <sql-row> element is contained in a <sql-expect> element. The <sql-row> element is generally a container for one or more <sql-column> elements, which specify the expected data in a given row. The <sql-row> element itself does not include any specific expected values. The order of the <sql-row> elements may impact the verification of the expected return set depending on the settings of the <sql-expect> element in which it is contained.

Structure

Contains:

- <sql-column>

Contained By:

- <sql-expect>

Attributes: None

<sql-column>

The <sql-column> element is contained in a <sql-row> element and specifies the expected data returned in that column. The <sql-column> element can specify the name of the column whose data is to be verified. Alternately, the <sql-row> element can contain <sql-column> elements with no name specification, in which case the data in the columns of the record must be in the same order as the <sql-column> elements within the <sql-row>. If the data specified in the <sql-column> elements contents does no match the data in the column within the row of the return set a script error is thrown.

Structure

Contains:

- Text - The expected value of the column within the row.

Contained By:

- `<sql-row>`

Table 47. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
name	This attribute can specify the name of the column within the record of the return set whose data is to be verified by the <code><sql-column></code> element. If this attribute is omitted, the <code><sql-column></code> elements must match the order of the columns in the rows of the return set or a script error is thrown. Note the either all <code><sql-column></code> elements within the same <code><sql-row></code> must specify the name attribute, or none of them can.	String	None	No

Agentry Test Script: Tab Elements Overview

The tab-related elements in the Agentry Test Script language are provided to allow for the selection of tabs within a screen set on the Client, and to test for expected values within the labels of those tabs.

In later versions of Agentry, screens within a screen set can be presented without tabs and instead using a popup menu to allow users to select a different screen within the screen set. The tab elements will work with this interface option as well.

<tab-expect>

The `<tab-expect>` element can be used to verify the label of a tab. The contents of the element specify the label expected to be displayed for the tab. The tab may be specified based on the screen definition's name, the screen's position index, a position relative to the currently selected tab's position, or by specifying the first or last tab from left to right. If the tab's label does not match the text contents of the `<tab-expect>` element a script error is thrown. If the tab, name, and label attributes are all omitted from this element, the currently selected tab's label is verified.

Structure

Contains:

- Text - The expected label for the tab.

Contained By:

- `<script>`

Table 48. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
tab	<p>This attribute can be set to specify the tab to select based on its position relative to the currently selected tab, the first or last tab, or by specifying a positive integer matching the screen's position index within the screen set. Valid values for this attribute include:</p> <ul style="list-style-type: none"> • first - The first (or left-most) tab displayed. • last - The last (or right-most) tab displayed. • next - The next tab to the right of the currently selected tab. • previous - The previous tab to the right of the currently selected tab. • A positive integer matching the screen definitions position within the screen set. 	String	None	No
name	This attribute can be set to the screen definitions name to specify the tab whose label is to be verified.	String	None	No
label	This attribute can be set to specify the label for the tab whose label is to be verified. While supported for the <code><tab-expect></code> element, it makes little sense to use this attribute.	String	None	No
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<tab-select>

The `<tab-select>` element will select a tab on the client, in essence selecting the screen within a multi-screen screen set. The tab may be selected based on the tab's label (screen caption), the screen definition's name, the screen's position index, a position relative to the currently selected tab's position, or by specifying the first or last tab from left to right.

Structure**Contains:**

- None

Contained By:

- `<script>`

Table 49. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
tab	<p>This attribute can be set to specify the tab to select based on its position relative to the currently selected tab, the first or last tab, or by specifying a positive integer matching the screen's position index within the screen set. Valid values for this attribute include:</p> <ul style="list-style-type: none"> • first - The first (or left-most) tab displayed. • last - The last (or right-most) tab displayed. • next - The next tab to the right of the currently selected tab. • previous - The previous tab to the right of the currently selected tab. • A positive integer matching the screen definitions position within the screen set. 	String	None	No
name	This attribute can be set to the screen definition's name to specify the tab to be selected.	String	None	No
label	This attribute can be set to the label of the tab, which is the screen definitions caption attribute value, to be selected.	String	None	No
common script attributes	<p>This element includes the following common script attributes:</p> <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

Agency Test Script: Window Elements Overview

The window-related elements within the Agency Test Script are provided to allow for interaction with the windows presented by the mobile client application. This is, in essence, the same as interacting with the screen set itself.

The elements provided include those to close the window to either return to the previous window or to return to the window presenting the main screen set of the module, to check for expected values in the title bar of the currently displayed screen, and also those to support entering a signature value in a detail screen displaying a property of type Signature.

Note that the elements to close the current window should only be used if there is no navigational action available within the application to perform this operation. If such an action is present, it should be used just as if a user were executing that action.

<window-close>

The <window-close> element will close the current or specified window on the Client. Note that if an action button or some other control exists to close a window on the client that control should be used. If the specified window cannot be found a script error is thrown.

Structure

Contains:

- Text - The caption of the window to be closed.

Contained By:

- <script>

Table 50. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<window-close-main>

The <window-close-main> will close the test client running within the ATE. This will be the equivalent of selecting the **File | Exit** menu item in the client's menu bar.

Structure

Contains:

- None

Contained By:

- <script>

Table 51. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep 	N/A	N/A	N/A

<window-expect>

The <window-expect> element will verify the caption of the currently displayed screen on the client. If the screen caption does not match the expected value a script error is thrown.

Structure

Contains:

- Text - The expected caption of the screen/window displayed on the client.

Contained By:

- <script>

Table 52. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • window • timeout • sleep 	N/A	N/A	N/A

<window-sign>

The <window-sign> element is provided to enter a signature in a signature capture field. This element will enter either a default signature, which is an X character, or may enter a more sophisticated image in the signature capture field. To specify the signature to enter other than the default signature, the <window-sign> element must contain two or more <point> elements, each of which specifies a point to draw a line in the signature capture field.

Structure

Contains:

- `<point>`

Contained By:

- `<script>`

Table 53. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none">• window• control - for this element always set to IDC_SIGNATURE_CAPTURE.• timeout• sleep	N/A	N/A	N/A

`<point>`

The `<point>` element specifies a point within a signature capture control. The exact position is specified via two pixel coordinates, x and y. Two `<point>` elements contained within the same `<window-signature>` element will be connected. Subsequent `<point>` elements will continue to be connected, that is, the first and second `<point>` will be connected via a single line, then the second and third `<point>` elements will be connected, and so forth. If either the x or y coordinate of a point are outside the boundaries defined for the signature capture field a script error is thrown.

Structure

Contains:

- None

Contained By:

- `<window-sign>`

Table 54. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
x	The x, or horizontal coordinate for the point, specified in pixels, with the left-most pixel at position 0.	non-negative integer	none	Yes
y	The y, or vertical coordinate for the point, specified in pixels with the bottom-most pixel at position 0.	non-negative integer	none	Yes

Agency Test Script: Client Elements Overview

The client-related elements of the Agency Test Script are used to affect or interact with the client process itself. These elements can restart the client and retrieve and set registry key values.

Typically these elements are added manually to a test script, as the script recorded in the ATE does not support recording these values.

Those elements related to the registry should be used with caution and only by those with an understanding of the registry keys and values for the Agency Test Environment and Client. If present, the script must be run by a user with permission to change Windows registry keys on the host system for the ATE.

<client-restart>

The <client-restart> element will restart the client process running within the Agency Test Environment. As optional behavior for this element, using the <registry>, <key>, and <value> elements, one or more registry keys may be added, deleted, or modified during this restart. Note that this element will restart the test client running within the ATE. It will not restart the ATE itself.

Structure

Contains:

- <registry>

Contained By:

- <script>

Table 55. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

<registry>

The <registry> element can be used with the <restart-client> element to create or modify registry keys and values. This element requires the use of a <key> child element to specify the registry key to be affected.

Structure

Contains:

- <key>

Contained By:

- <restart-client>

Attributes:

- None

<key>

The <key> element specifies a registry key to be modified or created. Nested registry keys require nested <key> elements within the test script. If a specified key does not exist will be created. The reset attribute allows for the specified key to be deleted and recreated. If the <key> element contains a <value> element, the contents of the <value> will specify the value to which the named key should be set.

Structure

Contains:

- <key>
- <value>

Contained By:

- <registry>
- <key>

Table 56. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
name	This attribute specifies the name of the registry key to be affected by the <key> element. May also contain one of: <ul style="list-style-type: none"> HKLM - HKEY_LOCAL_MACHINE HKCU - HKEY_CURRENT_USER 	string	none	Yes
reset	This attributes specifies whether the named key should be reset. The value <code>t</code> is treated as true and will result in the key being deleted and recreated. The value <code>f</code> is false.	string	f	No

<value>

The <value> element specifies the value to be set for the named registry key. The <value> element must be contained by a <key> element. The <value> element names the registry key that is to be modified within the one named by the containing <key> element. The <value> element can specify that the named key's value is to be added, modified, or deleted. When adding or modifying the value of a key, the <value> element also specifies the data type of the registry key value.

Structure

Contains:

- Text - contains the value to be set for the key. Ignored when the `reset` attribute is set to true.

Contained By:

- <key>

Table 57. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
name	The name of the registry key to be affected by the <value> element.	String	None	Yes

Name	Description	Data Type	De-fault Value	Re-quired
type	Specifies the data type of the value expected for the registry key. Must be one of: <ul style="list-style-type: none"> binary dword string 	String	None	Yes - <i>ignored if reset is true.</i>
reset	Specifies whether or not the specified key should be deleted. The value <code>t</code> is true and will delete the value of named registry key. The value <code>f</code> is false.	String	<code>f</code>	No
common script attributes	A set of related attributes common to most elements within the Agentry Test Script.	N/A	N/A	N/A

Agentry Test Script: Client Host Elements overview

The client host-related elements provided in the Agentry Test Script are used to perform actions on the client host itself, outside the test client. This includes the ability to execute commands on the host system and to specify key's to be pressed on the client device.

While the test script runs only on the Agentry Test Environment, which can only be installed to a desktop, the interactions between the test client and host system can be scripted to mimic the behaviors of the target device type.

<command-line>

The `<command-line>` element allows for the execution of a command on the client device during test script playback. This element can specify how long to wait for the command to return and its expected exit code. The contents of the element are the commands to execute. Alternately a command may be specified using the `cmd` attribute of the element. These commands can be written to a temporary file that itself will then be executed by the `<command-line>` element. The default file extension for this file is `.bat`, though this can be overridden. If the specified command cannot be executed a script error is thrown.

Structure

Contains:

- Text - The command(s) to be executed by the element.

Contained By:

- `<script>`

Table 58. Attributes

Name	Description	Data Type	De-fault Value	Re-quired
cmd	This attribute specifies the command to be executed by the element, or to pass parameters to a temporary file created by the element that will be executed. The first value in this parameter should be the token %1 when used to pass parameters to a temporary script file. This should then be followed by the parameters, each separated by whitespace. The contents of the element should then be the command with the tokens %1 through %n for each parameter specified in the cmd attribute.	string	None	No
ext	This attribute specifies the file extension for the temporary file to contain the commands executed by the element. The default file extension is .bat. The value of this attribute can optionally include a leading period (.).	string	.bat	No
waitFor	This attribute specifies the duration of time in milliseconds to wait for the command to complete execution. The value -1 indicates to wait indefinitely. The value 0 indicates to execute the command and not wait for it to complete. The script will continue playback regardless of the result of executing the command. If the command does not exit before the specified duration elapses a script error is thrown.	integer	0	No
expectedExit	This attribute specifies the expected exit code to be returned by the processes executed by the command. This can be any non-negative integral number. This attribute is ignored if waitFor is set to 0. If this attribute is not specified, the exit code of the command will not be checked. If the exit code differs from the value specified a script error is thrown.	non-negative integer.	None	No

Name	Description	Data Type	De-fault Value	Re-quired
common script attributes	This element includes the following common script attributes: <ul style="list-style-type: none"> • timeout • sleep • frame 	N/A	N/A	N/A

<key-press>

The <key-press> element will enter a single keystroke, mimicking that keystroke on a standard keyboard. This element includes the option of including key the combinations using the Alt, Ctrl, and/or Shift keys. Only a single character or keystroke can be set using the <key-press> element.

Structure

Contains:

- None

Contained By:

- <script>

Table 59. Attributes

Name	Description	Data Type	Default Value	Required
key	<p>This attribute specifies the key stroke to enter. This may be one of the following values:</p> <ul style="list-style-type: none"> Any single printable character “backspace” - The backspace key. “tab” - The Tab key. “enter” - The Enter/Return key. “return” - The Enter/Return key. “pause” - The Pause key. “esc” - The Escape (Esc) key. “space” - The spacebar. “page up” - The Page Up key. “page down” - The Page Down key. “end” - The End key. “home” - The Home key. “left” - The left arrow key. “right” - The right arrow key. “up” - The up arrow key. “down” - The down arrow key. “print screen” - The Print Screen key. “insert” - The Insert key. “delete” The Delete key. “Fn” - <i>n</i> is any numeric value from 1 to 24. That function key is then pressed. 	character	None	Yes
shift	<p>This attribute specifies whether to depress the Shift key in combination with the key attribute. The value <code>t</code> is true and will depress the Shift key. The value <code>f</code> is false.</p>	Boolean	<code>f</code>	No
ctrl	<p>This attribute specifies whether to depress the Ctrl key in combination with the key attribute. The value <code>t</code> is true and will depress the Ctrl key. The value <code>f</code> is false.</p>	Boolean	<code>f</code>	No

Name	Description	Data Type	De-fault Value	Re-quired
alt	This attribute specifies whether to depress the <code>Alt</code> key in combination with the key attribute. The value <code>t</code> is true and will depress the <code>Alt</code> key. The value <code>f</code> is false.	Boolean	<code>f</code>	No
common script attributes	A set of related attributes common to most elements within the Agency Test Script.	N/A	N/A	N/A

Agency Java API

The Agency Java API is provided for Agency applications that make use of a Java system connection for data synchronization. This API exposes the mobile application data to the Java logic, system and user information, and other key data needed during synchronization.

com.syclo.agency package

utility package

java_logging package

AgencyHandler class

This is an implementation of the Java Logging API's Handler class that will route log messages to the Agency Server Java System Connection's log file.

Syntax

```
public class AgencyHandler extends Handler
```

Members

All members of AgencyHandler, including inherited members. **Variables**

Modifier and Type	Variable	Description
protected Formatter	<code>_defaultFormatter</code> on page 650	This is the default formatter used by this handler.

Constructors

Modifier and Type	Constructor	Description
public	<i>AgentryHandler()</i> on page 649	Constructs a new AgentryHandler object.

Methods

Modifier and Type	Method	Description
public void	<i>close()</i> on page 650	Agentry log files cannot be closed from Java; this method does nothing.
public void	<i>flush()</i> on page 650	The Agentry server always flushes its log files; this method does not do anything explicit.
protected LogLevel	<i>mapLogLevel(Level)</i> on page 650	Maps a Java log level to an Agentry log level, as described in the class documentation.
public void	<i>publish(LogRecord)</i> on page 650	Publishes a log record to the Agentry server.

Usage

It is also capable of routing messages to an Agentry user log file, if it receives objects of class `UserLogRecord` instead of objects of the base class `LogRecord`. `UserLogRecord` objects can be easily generated by using the `UserLogger` class.

Both the Java Logging API and the Agentry Server support the concept of log levels. The logging level of the Agentry Server is configured in the `AgentryLogging.ini` file, and the Agentry Server supports six levels, 0 through 5, with level 0 messages always being logged. For each system connection, there is a level setting in the configuration file, and all messages at or below that setting will be logged.

The levels in the Java Logging API map to the log levels in Agentry as follows:

Java Logging API	Agentry	Purpose
Level#SEVERE	Level 0	Severe errors, always logged
Level#WARNING	Level 1	Warnings or recoverable errors
Level#INFO	Level 2	"Now doing this" type messages

Java Logging API	Agentry	Purpose
Level#CONFIG	Level 2	Configuration settings and related messages. Note that this will map to the same Agentry level as Level . INFO messages.
Level#FINE	Level 3	Details of why a decision is being made. This is the level that the Server.debug and User.debug methods will log at.
Level#FINER	Level 4	Values being set from files, user input, etc; details of calculations.
Level#FINEST	Level 5	Really low level details. This is the level that the AJAPI classes and the Agentry Server will log at, so if you enable this you may get a lot of log messages. Because of this, using this level in end-user code is discouraged.

Configuration:

By default each `AgentryHandler` is initialized using the following `LogManager` configuration properties:

- `com.syclo.agentry.utility.java_logging.AgencyHandler.level`
- Specifies the default level (defaults to `Level#ALL`).
- `com.syclo.agentry.utility.java_logging.AgencyHandler.filter`
- Specifies the name of a Filter class (defaults to no filter).
- `com.syclo.agentry.utility.java_logging.AgencyHandler.formatter`
- Specifies the name of a Formatter class to use (defaults to an internal formatter that produces just the text of the message by itself; Agentry will add a timestamp to the message when it logs it).

AgentryHandler() constructor

Constructs a new `AgentryHandler` object.

Syntax

```
public    AgentryHandler ()
```

close() method

Agentry log files cannot be closed from Java; this method does nothing.

Syntax

```
public void close ()
```

flush() method

The Agentry server always flushes its log files; this method does not do anything explicit.

Syntax

```
public void flush ()
```

mapLogLevel(Level) method

Maps a Java log level to an Agentry log level, as described in the class documentation.

Syntax

```
protected LogLevel mapLogLevel ( Level javaLevel )
```

Parameters

- **javaLevel** – the Java log level

Returns

An LogLevel constant

Usage

If you have custom log levels, you can subclass this class and override this method to map your custom log levels to Agentry's levels. Otherwise, this method will make a best-guess attempt at mapping custom log levels, based on which of Java's levels the custom levels fall between (for example, if your custom level's integer value is greater than `Level.INFO` but less than `Level.WARNING`, it will be treated as equivalent to `Level.INFO`).

publish(LogRecord) method

Publishes a log record to the Agentry server.

Syntax

```
public void publish ( LogRecord record )
```

_defaultFormatter variable

This is the default formatter used by this handler.

Syntax

```
protected Formatter _defaultFormatter
```

Usage

It just does localization and parameter substitution, and otherwise returns the message without any other adornment.

AgentryJavaLoggingConfigurator class

This class is used by the Java System Connection to set up a default configuration for the Java Logging API.

Syntax

```
public class AgentryJavaLoggingConfigurator
```

Members

All members of *AgentryJavaLoggingConfigurator*, including inherited members. **Nested classes**

Modifier and Type	Class	Description
class	<i>ReallySimpleFormatter</i> on page 651	This is a basic formatter class that is used for the AgentryJavaCompiler log file handler (to mimic the format that com.syclo.agentry.utility.Logger was using).

Constructors

Modifier and Type	Constructor	Description
public	<i>AgentryJavaLoggingConfigurator()</i> on page 652	Configure the Java Logging API via the properties file at com/syclo/agentry/internal/logging.properties.

Usage

The default configuration will set up a single root logger using the *AgentryHandler* handler, which will result in all log messages being routed to the Java System Connection's log file on the Agentry Server.

AgentryJavaLoggingConfigurator.ReallySimpleFormatter class

This is a basic formatter class that is used for the *AgentryJavaCompiler* log file handler (to mimic the format that com.syclo.agentry.utility.Logger was using).

Syntax

```
class ReallySimpleFormatter extends Formatter
```

Members

All members of ReallySimpleFormatter, including inherited members. **Methods**

Modifier and Type	Method	Description
public String	<i>format(LogRecord)</i> on page 652	

Usage

It just prefixes the log message with a timestamp.

format(LogRecord) method

Syntax

```
public String format ( LogRecord record )
```

AgentryJavaLoggingConfigurator() constructor

Configure the Java Logging API via the properties file at com/syclo/agentry/internal/logging.properties.

Syntax

```
public AgentryJavaLoggingConfigurator () throws IOException
```

Exceptions

- **IOException** – if the logging properties file cannot be loaded.

Usage

In addition, add special file handlers for the various classes in com.syclo.agentry.internal, to mimic what those classes used to do with the deprecated com.syclo.agentry.utility.Logger class.

UserLogRecord class

This class is used to create a log record that, if received by an instance of AgentryHandler, will be routed to a user-specific log file on the Agentry server.

Syntax

```
public class UserLogRecord extends LogRecord
```

Members

All members of UserLogRecord, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>UserLogRecord(User, Level, String)</i> on page 655	Constructs a new UserLogRecord object.
public	<i>UserLogRecord(User, LogRecord)</i> on page 655	Constructs a new UserLogRecord object that copies all of its information from an existing LogRecord object.

Methods

Modifier and Type	Method	Description
public boolean	<i>equals(Object)</i> on page 655	
public Level	<i>getLevel()</i> on page 655	
public String	<i>getLoggerName()</i> on page 655	
public String	<i>getMessage()</i> on page 656	
public long	<i>getMillis()</i> on page 656	
public Object[]	<i>getParameters()</i> on page 656	
public ResourceBundle	<i>getResourceBundle()</i> on page 656	
public String	<i>getResourceBundleName()</i> on page 656	
public long	<i>getSequenceNumber()</i> on page 656	
public String	<i>getSourceClassName()</i> on page 656	
public String	<i>getSourceMethodName()</i> on page 656	
public int	<i>getThreadID()</i> on page 657	
public Throwable	<i>getThrown()</i> on page 657	
public User	<i>getUser()</i> on page 657	Returns the user that this log record pertains to.
public int	<i>hashCode()</i> on page 657	

Modifier and Type	Method	Description
public void	<i>setLevel(Level)</i> on page 657	
public void	<i>setLoggerName(String)</i> on page 657	
public void	<i>setMessage(String)</i> on page 657	
public void	<i>setMillis(long)</i> on page 658	
public void	<i>setParameters(Object[])</i> on page 658	
public void	<i>setResourceBundle(ResourceBundle)</i> on page 658	
public void	<i>setResourceBundle-Name(String)</i> on page 658	
public void	<i>setSequenceNumber(long)</i> on page 658	
public void	<i>setSourceClassName(String)</i> on page 658	
public void	<i>setSourceMethodName(String)</i> on page 658	
public void	<i>setThreadID(int)</i> on page 658	
public void	<i>setThrown(Throwable)</i> on page 659	
public String	<i>toString()</i> on page 659	

Usage

You can create instances of this class directly and hand them off to `Logger` objects to log user-specific messages, but it is far more convenient to use the `UserLogger` class, which is a subclass of `Logger` that handles converting `LogRecord` objects to `UserLogRecord` objects for you.

Note that it is perfectly acceptable for one of these objects to find its way to any other type of `Handler` object; other handlers will log these objects the same way they would log any other `LogRecord` object.

UserLogRecord(User, Level, String) constructor

Constructs a new UserLogRecord object.

Syntax

```
public UserLogRecord ( User user , Level level , String msg )
```

Parameters

- **user** – The user to log the message for
- **level** – The message level
- **msg** – The log message

UserLogRecord(User, LogRecord) constructor

Constructs a new UserLogRecord object that copies all of its information from an existing LogRecord object.

Syntax

```
public UserLogRecord ( User user , LogRecord record )
```

Parameters

- **user** – The user to log the message for
- **record** – The LogRecord object containing the original log message information.

Usage

This will copy all of the information from the existing log record, including message parameters.

equals(Object) method

Syntax

```
public boolean equals ( Object obj )
```

getLevel() method

Syntax

```
public Level getLevel ()
```

getLoggerName() method

Syntax

```
public String getLoggerName ()
```

getMessage() method

Syntax

```
public String getMessage ()
```

getMillis() method

Syntax

```
public long getMillis ()
```

getParameters() method

Syntax

```
public Object[] getParameters ()
```

getResourceBundle() method

Syntax

```
public ResourceBundle getResourceBundle ()
```

getResourceBundleName() method

Syntax

```
public String getResourceBundleName ()
```

getSequenceNumber() method

Syntax

```
public long getSequenceNumber ()
```

getSourceClassName() method

Syntax

```
public String getSourceClassName ()
```

getSourceMethodName() method

Syntax

```
public String getSourceMethodName ()
```


getThreadID() method

Syntax

```
public int getThreadID ()
```

getThrown() method

Syntax

```
public Throwable getThrown ()
```

getUser() method

Returns the user that this log record pertains to.

Syntax

```
public User getUser ()
```

Returns

the User object.

hashCode() method

Syntax

```
public int hashCode ()
```

setLevel(Level) method

Syntax

```
public void setLevel ( Level level )
```

setLoggerName(String) method

Syntax

```
public void setLoggerName ( String name )
```

setMessage(String) method

Syntax

```
public void setMessage ( String message )
```

setMillis(long) method

Syntax

```
public void setMillis ( long millis )
```

setParameters(Object[]) method

Syntax

```
public void setParameters ( Object[] parameters )
```

setResourceBundle(ResourceBundle) method

Syntax

```
public void setResourceBundle ( ResourceBundle bundle )
```

setResourceBundleName(String) method

Syntax

```
public void setResourceBundleName ( String name )
```

setSequenceNumber(long) method

Syntax

```
public void setSequenceNumber ( long seq )
```

setSourceClassName(String) method

Syntax

```
public void setSourceClassName ( String sourceClassName )
```

setSourceMethodName(String) method

Syntax

```
public void setSourceMethodName ( String sourceMethodName )
```

setThreadID(int) method

Syntax

```
public void setThreadID ( int threadID )
```

setThrown(Throwable) method

Syntax

```
public void setThrown ( Throwable thrown )
```

toString() method

Syntax

```
public String toString ()
```

UserLogger class

This class is used in combination with `AgentryHandler` to route messages to the user-specific log files on the Agentry Server.

Syntax

```
public class UserLogger extends Logger
```

Members

All members of `UserLogger`, including inherited members. **Methods**

Modifier and Type	Method	Description
public User	<i>getUser()</i> on page 660	Returns the user that this logger is logging for.
public static UserLogger	<i>getUserLogger(String, User)</i> on page 660	Returns an instance of <code>UserLogger</code> that uses the named <code>Logger</code> instance to log messages to the Agentry server for the given user.
public static UserLogger	<i>getUserLogger(String, String, User)</i> on page 660	Returns an instance of <code>UserLogger</code> that uses the named <code>Logger</code> instance to log messages to the Agentry server for the given user.
public void	<i>log(LogRecord)</i> on page 661	

Usage

It acts as a child logger of an existing `Logger` object. When its log methods are called, it creates log records of the class `UserLogRecord` instead of `LogRecord`, and then passes those log records to its parent logger. If these log records eventually find their way to an `AgentryHandler` instance, that instance will know to route the messages to user-specific logs on the Agentry server, if those logs are enabled via the `AgentryLogging.ini` file.

getUser() method

Returns the user that this logger is logging for.

Syntax

```
public User getUser ()
```

Returns

the `User` object.

getUserLogger(String, User) method

Returns an instance of `UserLogger` that uses the named `Logger` instance to log messages to the Agentry server for the given user.

Syntax

```
public static UserLogger getUserLogger (String name , User  
user )
```

Parameters

- **name** – The logger name. Note that the same `Logger` object may be used by multiple `UserLogger` objects to log messages for multiple users.
- **user** – The user to log messages for.

Returns

A new `UserLogger` object, or `null` if `Logger.getLogger(String)` would have returned `null` for the same logger name.

getUserLogger(String, String, User) method

Returns an instance of `UserLogger` that uses the named `Logger` instance to log messages to the Agentry server for the given user.

Syntax

```
public static UserLogger getUserLogger (String name , String  
resourceBundleName , User user )
```

Parameters

- **name** – The logger name. Note that the same `Logger` object may be used by multiple `UserLogger` objects to log messages for multiple users.
- **resourceBundleName** – The name of a resource bundle to use for localizing messages for this logger. May be `null` if no localization is necessary.
- **user** – The user to log messages for.

Returns

A new `UserLogger` object, or null if `Logger.getLogger(String)` would have returned null for the same logger name.

log(LogRecord) method

Syntax

```
public void log ( LogRecord record )
```

log4j package

AgentryAppender class

AgentryAppender is a Log4j Appender that will route log messages to the Agentry Server Java System Connection's log file.

Syntax

```
public class AgentryAppender extends AppenderSkeleton
```

Members

All members of AgentryAppender, including inherited members. **Variables**

Modifier and Type	Variable	Description
public static final String	<i>AGEN-TRY_USER_MDC_KEY</i> on page 664	MDC key for adding an Agentry User object to the Log4j MDC.

Constructors

Modifier and Type	Constructor	Description
public	<i>AgentryAppender()</i> on page 663	Constructs a new AgentryAppender object.

Methods

Modifier and Type	Method	Description
protected void	<i>append(LoggingEvent)</i> on page 663	
public void	<i>close()</i> on page 663	
protected LogLevel	<i>mapLogLevel(Level)</i> on page 663	Maps a Log4j log level to an Agentry log level, as described in the class documentation.

Modifier and Type	Method	Description
public boolean	<i>requiresLayout()</i> on page 663	

Usage

It is also capable of routing messages to an Agentry user log file, if the key named by AGENTRY_USER_MDC_KEY is set to an Agentry User object in Log4j's MDC.

Both Log4j and the Agentry Server support the concept of log levels. The logging level of the Agentry Server is configured in the `AgentryLogging.ini` file, and the Agentry Server supports six levels, 0 through 5, with level 0 messages always being logged. For each system connection, there is a level setting in the configuration file, and all messages at or below that setting will be logged.

The levels in Log4j map to the log levels in Agentry as follows:

Log4j	Agentry	Purpose
<code>Level.FATAL</code> , <code>Level.ERROR</code>	Level 0	Severe errors, always logged
<code>Level.WARN</code>	Level 1	Warnings or recoverable errors
<code>Level.INFO</code>	Level 2	"Now doing this" type messages
<code>Level.DEBUG</code>	Level 3	Details of why a decision is being made.
<code>Level.TRACE</code>	Level 4	Values being set from files, user input, etc; details of calculations

Note that there is currently no Log4j level that corresponds to log level 5 in Agentry. That log level is generally used by very low-level logging in the Agentry Server itself, so its use in end-user applications would be discouraged anyways.

`AgentryAppender` supports the basic settings used by other Log4j appenders, including layouts. It is not required to have a layout configured for the appender; if no layout is configured, then messages will be sent directly to Agentry with no special formatting.

NOTE: If you want to use this class you must have Log4j on your Agentry server's class path, as configured in `Agentry.ini`. Log4j is not distributed with the Agentry Server. It can be obtained from the *Apache Log4j* web site. This appender is intended for use with Log4j 1.2.x; it may work with later versions, but there is no guarantee.

AgentryAppender() constructor

Constructs a new AgentryAppender object.

Syntax

```
public AgentryAppender ()
```

append(LoggingEvent) method

Syntax

```
protected void append ( LoggingEvent event )
```

close() method

Syntax

```
public void close ()
```

mapLogLevel(Level) method

Maps a Log4j log level to an Agentry log level, as described in the class documentation.

Syntax

```
protected LogLevel mapLogLevel ( Level log4jLevel )
```

Parameters

- **log4jLevel** – the Java log level

Returns

An LogLevel constant

Usage

If you have custom log levels, you can subclass this class and override this method to map your custom log levels to Agentry's levels. Otherwise, this method will make a best-guess attempt at mapping custom log levels, based on which of Java's levels the custom levels fall between (for example, if your custom level's integer value is greater than `Level.INFO` but less than `Level.WARNING`, it will be treated as equivalent to `Level.INFO`).

requiresLayout() method

Syntax

```
public boolean requiresLayout ()
```

AGENTRY_USER_MDC_KEY variable

MDC key for adding an Agentry User object to the Log4j MDC.

Syntax

```
public static final String AGENTRY_USER_MDC_KEY
```

Usage

If a user is set in the MDC when a message is logged, then that message will be routed to the user's log file in the Agentry Server, instead of to the Java System Connection's log file.

DataTableMapIterator< K, V > class

This is a helper class that makes it easy to return Map objects from the iterator method of a DataTable.

Syntax

```
public class DataTableMapIterator< K, V > extends  
java.util.Iterator< DataTableObject >
```

Members

All members of *DataTableMapIterator< K, V >*, including inherited members. **Methods**

Modifier and Type	Method	Description
public	<i>DataTableMapIterator(Map< K, V >)</i> on page 664	Constructs a new <i>DataTableMapIterator</i> object.
public boolean	<i>hasNext()</i> on page 665	
public <i>DataTableObject</i>	<i>next()</i> on page 665	Returns a new <i>DataTableObject</i> that contains the key and value of the next entry in the map.
public void	<i>remove()</i> on page 665	

Usage

It iterates over the contents of the map and returns each map entry as a *DataTableObject*.

DataTableMapIterator(Map< K, V >) method

Constructs a new *DataTableMapIterator* object.

Syntax

```
public    DataTableMapIterator ( Map< K, V > map )
```


Parameters

- **map** – The Map to iterate over.

hasNext() method

Syntax

```
public boolean hasNext ()
```

next() method

Returns a new DataTableObject that contains the key and value of the next entry in the map.

Syntax

```
public DataTableObject next ()
```

remove() method

Syntax

```
public void remove ()
```

Logger class

Deprecated. Use java.util.logging or log4j. This class implements basic log file functionality, if you wish to log to another file other than Agentry's server or user log files.

Syntax

```
public class Logger
```

Members

All members of Logger, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>Logger(String, boolean)</i> on page 666	Create a new log file.

Methods

Modifier and Type	Method	Description
public void	<i>appendDebug(String)</i> on page 666	Append a message to the active debug buffer.
public void	<i>beginDebug(String)</i> on page 667	Opens up a multiline debug buffer and adds the given message to it.

Modifier and Type	Method	Description
public synchronized void	<i>debug(String)</i> on page 667	Write a message to the log file.
public synchronized void	<i>debug(String, Map< String, String >, String)</i> on page 667	Write the contents of a Map object to the log file.
public void	<i>endDebug(String)</i> on page 667	Close the debug buffer and write it to the log file.
public boolean	<i>isDebugMode()</i> on page 668	Returns whether a debug buffer has been opened by beginDebug and is still open.

Usage

If you need more powerful logging than what this class provides, you should look at the logging functionality that was added in JDK 1.4, or use Apache log4j or something like it.

Logger(String, boolean) constructor

Create a new log file.

Syntax

```
public    Logger ( String fileName ,    boolean overwrite ) throws
IOException
```

Parameters

- **fileName** – The log file name
- **overwrite** – `true` means to overwrite any existing file, `false` means append to any existing file.

Exceptions

- **IOException** – if an error occurs.

appendDebug(String) method

Append a message to the active debug buffer.

Syntax

```
public void appendDebug ( String message )
```

Parameters

- **message** – The message

beginDebug(String) method

Opens up a multiline debug buffer and adds the given message to it.

Syntax

```
public void beginDebug ( String message )
```

Parameters

- **message** – The debugging message

Usage

Additional messages can be added by calling appendDebug. The contents of the buffer will not be written to the log file until endDebug is called.

debug(String) method

Write a message to the log file.

Syntax

```
public synchronized void debug ( String message )
```

Parameters

- **message** – The message

debug(String, Map< String, String >, String) method

Write the contents of a Map object to the log file.

Syntax

```
public synchronized void debug ( String header , Map< String,  
String > messages , String footer )
```

Parameters

- **header** – The header message
- **messages** – The map to dump
- **footer** – The footer message

endDebug(String) method

Close the debug buffer and write it to the log file.

Syntax

```
public void endDebug ( String message )
```

Parameters

- **message** – The final debug message

isDebugMode() method

Returns whether a debug buffer has been opened by beginDebug and is still open.

Syntax

```
public boolean isDebugMode ()
```

Returns

true if a debug buffer is active.

AgentryException class

Base class for checked exceptions that can be thrown from various methods in the AJAPI.

Syntax

```
public class AgentryException extends Exception
```

Derived classes

- *BusinessLogicException* on page 671
- *FatalTransactionException* on page 694
- *FatalTransactionExceptionStop* on page 697
- *LoginException* on page 706
- *RetryTransactionException* on page 735
- *RetryTransactionWithChangeException* on page 737
- *StepletAbortException* on page 786
- *StepletStopException* on page 787

Members

All members of AgentryException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.

Modifier and Type	Constructor	Description
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.

Methods

Modifier and Type	Method	Description
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

Throwing this base class from a method will generally result in an error being logged in the server event log, and the client getting a "Server error - please contact your administrator (13)" message.

AgentryException(String) constructor

Constructs a new AgentryException object.

Syntax

```
public AgentryException ( String message )
```

Parameters

- **message** – The error message.

AgentryException(String, Throwable) constructor

Constructs a new AgentryException object.

Syntax

```
public AgentryException ( String message , Throwable cause )
```

Parameters

- **message** – The error message.
- **cause** – The causing exception.

AgentryException(String, String, String, Throwable) constructor

Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.

Syntax

```
public AgentryException (String title , String text , String okLabel ,  
Throwable cause )
```

Parameters

- **title** – The window title.
- **text** – The window text. This will also be used as the message text of the exception when it is logged (it will be returned when getMessage() is called).
- **okLabel** – The window button label.
- **cause** – The causing exception

Usage

For now this is pretty much supported only by steplets that are being used as part of Agentry transactions.

AgentryException(String, String, String) constructor

Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.

Syntax

```
public AgentryException (String title , String text , String  
okLabel )
```

Parameters

- **title** – The window title.
- **text** – The window text. This will also be used as the message text of the exception when it is logged (it will be returned when getMessage() is called).
- **okLabel** – The window button label.

Usage

For now this is pretty much supported only by steplets that are being used as part of Agentry transactions.

getNotificationText() method

Returns the notification window text.

Syntax

```
public final String getNotificationText ()
```

Returns

the text

getNotificationTitle() method

Returns the notification window title.

Syntax

```
public final String getNotificationTitle ()
```

Returns

the title

getOkButtonLabel() method

Returns the notification window button label.

Syntax

```
public final String getOkButtonLabel ()
```

Returns

the label

BusinessLogicException class

This exception represents an error condition that should be reported to the client.

Syntax

```
public class BusinessLogicException extends AgentryException
```

Members

All members of BusinessLogicException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>BusinessLogicException(String)</i> on page 672	Construct a new exception.
public	<i>BusinessLogicException(String, Throwable)</i> on page 673	Constructs a new BusinessLogicException object.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

It can be thrown from any of the various client-related session methods, such as the methods on `Steplet`, `DataTable`, and `ComplexTable`. The error message in this exception will be displayed in the client's transmit window; in addition, if the `printBusinessLogicStackTrace` configuration option is enabled in the `agentry.ini` file for the Java system connection, then a full stack trace for the exception will also be displayed in the client's transmit window.

BusinessLogicException(String) constructor

Construct a new exception.

Syntax

```
public BusinessLogicException ( String message )
```

Parameters

- **message** – The error message

BusinessLogicException(String, Throwable) constructor

Constructs a new BusinessLogicException object.

Syntax

```
public BusinessLogicException ( String message , Throwable
cause )
```

Parameters

- **message** – The error message
- **cause** – The causing exception

ComplexTableSession class

The ComplexTableSession class encapsulates the processing involved in a complex table synchronization within an Agency-based application.

Syntax

```
public class ComplexTableSession extends Session
```

Members

All members of ComplexTableSession, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>ComplexTableSession(String, Server, SessionData, User)</i> on page 674	Construct a new session.

Inherited members from Session

Modifier and Type	Member	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agency application.
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.

Modifier and Type	Member	Description
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
protected	<i>Session(String, Server, SessionData, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, SessionData)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

This class contains the session data for the complex table synchronization, as well the User object for the user performing the synchronization, and a reference to the Server singleton object.

ComplexTableSession(String, Server, SessionData, User) constructor

Construct a new session.

Syntax

```
public ComplexTableSession ( String tableName , Server server ,
SessionData sessionData , User user )
```

Parameters

- **tableName** – The complex table name, as configured in the Agentry application.
- **server** – The *Server* object that the Java System connection was configured to use.
- **sessionData** – Session data for this transmit.
- **user** – The client user performing the transmit.

Usage

This constructor is called by the *Server.createComplexTableSession* method. Subclasses should implement a constructor with the same signature as this one.

ComplexTable< CTOBJ > class

The ComplexTable class encapsulates the data retrieval for a complex table within an Agentry-based application.

Syntax

```
public abstract class ComplexTable< CTOBJ > extends
AgentryJavaBackEndManagedObject
```

Members

All members of ComplexTable< CTOBJ >, including inherited members. **Variables**

Modifier and Type	Variable	Description
protected SycloCalendar	<i>_clientLastDataUpdateTime</i> on page 685	Storage for the client's last update time that was passed into the constructor.
protected boolean	<i>_rebuilding</i> on page 686	This member will be set by the Agentry Server to whatever the willRebuildTable() method returns.
protected ComplexTableSession	<i>_session</i> on page 686	Storage for the session that was passed into the constructor.

Methods

Modifier and Type	Method	Description
public void	<i>build()</i> on page 678	This method is called by the Agentry Server after it calls willRebuildTable(), but before it calls dataIterator() or deleteIterator().
final boolean	<i>checkForReload()</i> on page 678	This method is called by the Agentry Server to indicate whether a reload should be done.
public	<i>ComplexTable(ComplexTableSession, GregorianCalendar)</i> on page 679	Constructs a new ComplexTable object.

Modifier and Type	Method	Description
public abstract Iterator< CTOBJ >	<i>dataIterator()</i> on page 679	This method should return either a partial or complete list of the records for the complex table, depending on the result of calling <i>willRebuildTable()</i> .
public Iterator< CTOBJ >	<i>deleteIterator()</i> on page 680	This method should return a list of the records that have been deleted from the complex table since the client's last update time, which can be retrieved via <i>getClientLastDataUpdateTime()</i> .
public SycloCalendar	<i>getClientLastDataUpdateTime()</i> on page 681	Returns the time when the client last retrieved data for this complex table.
public GregorianCalendar	<i>getNewDataUpdateTime()</i> on page 681	Returns the new "last data update time", as set by a prior call to <i>setNewDataUpdateTime</i> .
public ComplexTableSession	<i>getSession()</i> on page 681	Return the session for this complex table transmission.
public final void	<i>initialize()</i> on page 681	Deprecated. Subclass overrides of this method should be renamed to <i>build()</i> . This method has been renamed to <i>build()</i> .
public boolean	<i>isRebuilding()</i> on page 682	This method returns the previously cached result of the Agentry Server's call to the <i>willRebuildTable()</i> method.
final int	<i>lastUpdateDate()</i> on page 682	Returns the day-of-month component of the complex table data's last update time.
final int	<i>lastUpdateHours()</i> on page 682	Returns the hour component of the complex table data's last update time.
final int	<i>lastUpdateMinutes()</i> on page 682	Returns the minute component of the complex table data's last update time.

Modifier and Type	Method	Description
final int	<i>lastUpdateMonth()</i> on page 683	Returns the month component of the complex table data's last update time.
final int	<i>lastUpdateSeconds()</i> on page 683	Returns the seconds component of the complex table data's last update time.
final int	<i>lastUpdateYear()</i> on page 683	Returns the year component of the complex table data's last update time.
public final boolean	<i>reload()</i> on page 684	Deprecated. Subclass overrides of this method should be renamed to <i>willRebuildTable()</i> . This method has been renamed to <i>willRebuildTable()</i> .
public void	<i>setNewDataUpdateTime(GregorianCalendar)</i> on page 684	This method sets the new "last data update time" for the complex table.
public boolean	<i>willRebuildTable()</i> on page 685	This method is called by the Agentry server to determine whether the server should rebuild the client's complex table from scratch.

Usage

When implementing a complex table using a Java Interface system connection, this class will always be extended.

Complex tables in Agentry support both full and incremental updates. It is up to an implementing subclass to determine which mechanism to use, based on whether the underlying data source supports time-stamped data. If it does, an implementation can return a "last update" timestamp along with the complex table data. The next time a client requests an update of the table, it will pass back that timestamp, and this class can then return only the data changes (additions, updates, and deletions) that have occurred since that time.

Data objects are returned from this class to the Agentry server via iterators. The data objects themselves are structured in the same manner as the data objects for a Steplet, in that their data is stored in public member fields that are accessed directly by Agentry, and are mapped to corresponding fields in the Complex Table when the table is defined in the Agentry Editor.

The synchronization process for a complex table involves determining first which records have been deleted from the complex table and retrieving those records' unique identifiers. Then, the records that have been added or updated since the last client update are retrieved. The Agentry Server will call the methods of this class in the following sequence:

1. Constructor - passes in the client's last update date
2. `willRebuildTable()` - tells Agentry whether to expect a full or incremental update
3. `build()` - can be used to gather records
4. `dataIterator()` - returns new or updated records
5. `lastUpdate*` methods - returns the new data timestamp (set via a call to `setNewDataUpdateTime`)
6. `deleteIterator()` - returns keys for deleted records; may not be called if `willRebuildTable()` returned true
7. `lastUpdate*` methods (again - should return the same value as the previous call); may not be called if `willRebuildTable()` returned true

Note that for each client transmit, a new instance of this class is created by the Agentry server.

build() method

This method is called by the Agentry Server after it calls `willRebuildTable()`, but before it calls `dataIterator()` or `deleteIterator()`.

Syntax

```
public void build () throws AgentryException
```

Exceptions

- **AgentryException class** – if an error occurs.

Usage

It can be used to build up the data sets for the complex table, if it is more convenient to query both the updates and the deletes in a single place rather than querying them separately in the iterator methods.

checkForReload() method

This method is called by the Agentry Server to indicate whether a reload should be done.

Syntax

```
final boolean checkForReload () throws AgentryException
```

Returns

true if the table should be reloaded

Exceptions

- **AgentryException class** – if an error occurs

Usage

It calls `willRebuildTable()` and saves the result into the `_rebuilding` member variable.

ComplexTable(ComplexTableSession, GregorianCalendar) method

Constructs a new `ComplexTable` object.

Syntax

```
public ComplexTable ( ComplexTableSession session ,
GregorianCalendar clientLastDataUpdate )
```

Parameters

- **session** – A `ComplexTableSession` object that provides the access to the session data pertinent to the complex table application component.
- **clientLastDataUpdate** – This argument contains the date and time of the client's last data update.

Usage

Subclasses need to implement the same constructor and pass the parameters back to this constructor via `super()`.

This constructor will set the return value of the default implementation of the `willRebuildTable()` method to `true` if the `clientLastDataUpdate` parameter contains the invalid date/time value, or to `false` if not.

dataIterator() method

This method should return either a partial or complete list of the records for the complex table, depending on the result of calling `willRebuildTable()`.

Syntax

```
public abstract Iterator< CTOBJ > dataIterator () throws
AgentryException
```

Returns

an `Iterator` object that will iterate over the records of the complex table.

Exceptions

- **AgentryException class** – if an error occurs.

Usage

It returns these records in the form of a `Iterator` object that iterates over a list of objects. Subclasses should override this method to return the desired data.

If `willRebuildTable()` returns `false` (indicating that the client will receive an incremental update), then this method should only return those records that have been added to the complex table since the client's last update time (which can be retrieved via `getClientLastDataUpdateTime()`). If `willRebuildTable()` returns `true` (indicating that the client should rebuild its complex table from scratch), then this method should return the complete set of complex table records.

deleteIterator() method

This method should return a list of the records that have been deleted from the complex table since the client's last update time, which can be retrieved via `getClientLastDataUpdateTime()`.

Syntax

```
public Iterator< CTOBJ > deleteIterator () throws  
AgentryException
```

Returns

a `Iterator` object that will iterate over the records that have been deleted from the complex table since the client's last update.

Exceptions

- **AgentryException class** – if an error occurs.

Usage

It returns these records in the form of a `Iterator` object that iterates over a list of objects that identify the deleted records via their unique keys. Subclasses should override this method to return the desired data. Note that the objects returned do not need to be fully populated, they only need to contain their unique identifiers (which are configured in the Agentry application).

The actual behavior of this method is dependent on the result of the `willRebuildTable()` method. If that method returns `false`, then this method should return a list of deleted records. If that method returns `true`, then this method should return an empty iterator. The default behavior of this method is to return an empty iterator, so if your `willRebuildTable()` implementation will always return `true`, then you don't need to override this method.

Note that in some versions of Agentry, this method may not be called at all if `willRebuildTable()` returns `true`, since there should be no deleted objects in that case.

getClientLastDataUpdateTime() method

Returns the time when the client last retrieved data for this complex table.

Syntax

```
public SycloCalendar getClientLastDataUpdateTime ()
```

Returns

the client's last data update time.

getNewDataUpdateTime() method

Returns the new "last data update time", as set by a prior call to setNewDataUpdateTime.

Syntax

```
public GregorianCalendar getNewDataUpdateTime ()
```

Returns

The new update time, or null if setNewDataUpdateTime () hasn't been called yet.

getSession() method

Return the session for this complex table transmission.

Syntax

```
public ComplexTableSession getSession ()
```

Returns

the session

initialize() method [deprecated]

Deprecated. Subclass overrides of this method should be renamed to build(). This method has been renamed to build().

Syntax

```
public final void initialize () throws AgentryException
```

Exceptions

- **AgentryException** class – not thrown
- **UnsupportedOperationException** – to report that the method is no longer used.

Usage

Override that method instead of this one. This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

isRebuilding() method

This method returns the previously cached result of the Agentry Server's call to the willRebuildTable() method.

Syntax

```
public boolean isRebuilding ()
```

Returns

the value of the _rebuilding member variable.

lastUpdateDate() method

Returns the day-of-month component of the complex table data's last update time.

Syntax

```
final int lastUpdateDate ()
```

Returns

The day-of-month component of the complex table data's last update time.

Usage

This method is called by the Agentry Server to retrieve the last update time.

lastUpdateHours() method

Returns the hour component of the complex table data's last update time.

Syntax

```
final int lastUpdateHours ()
```

Returns

The hour component of the complex table data's last update time.

Usage

This method is called by the Agentry Server to retrieve the last update time.

lastUpdateMinutes() method

Returns the minute component of the complex table data's last update time.

Syntax

```
final int lastUpdateMinutes ()
```

Returns

The minute component of the complex table data's last update time.

Usage

This method is called by the Agentry Server to retrieve the last update time.

lastUpdateMonth() method

Returns the month component of the complex table data's last update time.

Syntax

```
final int lastUpdateMonth ()
```

Returns

The month component of the complex table data's last update time.

Usage

This method is called by the Agentry Server to retrieve the last update time.

lastUpdateSeconds() method

Returns the seconds component of the complex table data's last update time.

Syntax

```
final int lastUpdateSeconds ()
```

Returns

The seconds component of the complex table data's last update time.

Usage

This method is called by the Agentry Server to retrieve the last update time.

lastUpdateYear() method

Returns the year component of the complex table data's last update time.

Syntax

```
final int lastUpdateYear ()
```

Returns

The year component of the complex table data's last update time.

Usage

This method is called by the Agentry Server to retrieve the last update time.

reload() method [deprecated]

Deprecated. Subclass overrides of this method should be renamed to `willRebuildTable()`. This method has been renamed to `willRebuildTable()`.

Syntax

```
public final boolean reload () throws AgentryException
```

Returns

Nothing. Always throws `UnsupportedOperationException`.

Exceptions

- **AgentryException** class – not thrown
- **UnsupportedOperationException** – to report that the method is no longer used.

Usage

Override that method instead of this one. This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

setNewDataUpdateTime(GregorianCalendar) method

This method sets the new "last data update time" for the complex table.

Syntax

```
public void setNewDataUpdateTime ( GregorianCalendar  
dataUpdateTime )
```

Parameters

- **dataUpdateTime** – The last update time of the complex table's data.

Exceptions

- **IllegalArgumentException** – if `dataUpdateTime` is null, or is set to the Agentry "invalid" date/time value.

Usage

By default, the last update time will be "now"; however, if your underlying data source has a concept of a last update time, you can pass that on to the Agentry Server by calling this method from the constructor, `build()`, or `dataIterator()` to set it.

Note that calling this method does not affect the return value of `getClientLastDataUpdateTime()`; the two values are maintained separately.

willRebuildTable() method

This method is called by the Agentry server to determine whether the server should rebuild the client's complex table from scratch.

Syntax

```
public boolean willRebuildTable () throws AgentryException
```

Returns

`true` if the client's table should be rebuilt from scratch, or `false` if the client's existing table should be updated.

Exceptions

- **AgentryException class** – if an error occurs.

Usage

If you return `true` from this method, then a subsequent call to `dataIterator()` should return the full set of complex table records and `deleteIterator()` should return nothing; if you return `false` from this method then `dataIterator()` and `deleteIterator()` should return incremental changes.

Note that if the client's last data update date was invalid, then the Agentry Server will assume that the table needs to be reloaded and will not call this method at all.

When the Agentry server calls this method, it will store the result of the call into the `_rebuilding` field. That field will also be initialized by the constructor to `true` if `_clientLastDataUpdateTime` is invalid or `false` if it is not, so that the field will contain the correct value even if the Agentry server does not call this method.

The default implementation of this method returns the value of `_rebuilding` as set by the constructor.

_clientLastDataUpdateTime variable

Storage for the client's last update time that was passed into the constructor.

Syntax

```
protected SycloCalendar _clientLastDataUpdateTime
```

Usage

This is a `SycloCalendar` instead of a `GregorianCalendar` because it is possible for it to contain Agentry's "invalid" timestamp.

_rebuilding variable

This member will be set by the Agentry Server to whatever the willRebuildTable() method returns.

Syntax

```
protected boolean _rebuilding
```

Usage

It can be read via the isRebuilding() method.

_session variable

Storage for the session that was passed into the constructor.

Syntax

```
protected ComplexTableSession _session
```

DataTableObject class

This class represents a single row in an Agentry data table, which consists of a key/value pair.

Syntax

```
public class DataTableObject extends Object
```

Members

All members of DataTableObject, including inherited members. **Variables**

Modifier and Type	Variable	Description
public String	<i>code</i> on page 688	The code/key for the row.
public String	<i>value</i> on page 688	The value for the row.

Constructors

Modifier and Type	Constructor	Description
public	<i>DataTableObject(String, String)</i> on page 687	Constructs a new DataTableObject object.

Methods

Modifier and Type	Method	Description
public String	<i>code()</i> on page 687	Deprecated. Use getKey(). Return the code/key for this row.
public boolean	<i>equals(Object)</i> on page 687	

Modifier and Type	Method	Description
public String	<i>getKey()</i> on page 687	Return the code/key for this row.
public String	<i>getValue()</i> on page 688	Return the value for this row.
public int	<i>hashCode()</i> on page 688	
public String	<i>value()</i> on page 688	Deprecated. Use <i>getValue()</i> . Return the value for this row.

DataTableObject(String, String) constructor

Constructs a new *DataTableObject* object.

Syntax

```
public    DataTableObject ( String code1 ,    String value1 )
```

Parameters

- **code1** – The code/key for the row
- **value1** – The value for the row

code() method [deprecated]

Deprecated. Use *getKey()*. Return the code/key for this row.

Syntax

```
public String code ()
```

Returns

the code

equals(Object) method

Syntax

```
public boolean equals ( Object obj )
```

getKey() method

Return the code/key for this row.

Syntax

```
public String getKey ()
```

Returns

the code

getValue() method

Return the value for this row.

Syntax

```
public String getValue ()
```

Returns

the value

hashCode() method

Syntax

```
public int hashCode ()
```

value() method [deprecated]

Deprecated. Use getValue(). Return the value for this row.

Syntax

```
public String value ()
```

Returns

the value

code variable

The code/key for the row.

Syntax

```
public String code
```

value variable

The value for the row.

Syntax

```
public String value
```

DataTableSession class

The DataTableSession class encapsulates the processing involved in a data table retrieval within an Agentry-based application.

Syntax

```
public class DataTableSession extends Session
```


Members

All members of `DataTableSession`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i><code>DataTableSession(String, Server, SessionData, User)</code></i> on page 690	Construct a new session.

Inherited members from `Session`

Modifier and Type	Member	Description
public final void	<i><code>debug(String)</code></i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i><code>getName()</code></i> on page 777	Returns the name of the session, as configured in the Agentry application.
public Server	<i><code>getServer()</code></i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i><code>getSessionData()</code></i> on page 778	Returns the session data for this session.
public User	<i><code>getUser()</code></i> on page 778	Returns the user for this session, if any.
protected	<i><code>Session(String, Server, SessionData, User)</code></i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <code>getName()</code> , <code>getServer()</code> , <code>getSessionData()</code> , and <code>getUser()</code> methods.
protected	<i><code>Session(String, Server, SessionData)</code></i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <code>getName()</code> , <code>getServer()</code> , and <code>getSessionData()</code> methods.
public void	<i><code>sessionAborted()</code></i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

This class contains the session data for the data table retrieval, as well the User object for the user performing the retrieval, and a reference to the Server singleton object.

DataTableSession(String, Server, SessionData, User) constructor

Construct a new session.

Syntax

```
public    DataTableSession ( String tableName ,    Server server ,
SessionData sessionData ,    User user )
```

Parameters

- **tableName** – The data table name, as configured in the Agentry application.
- **server** – The `Server` object that the Java System connection was configured to use.
- **sessionData** – Session data for this transmit.
- **user** – The client user performing the transmit.

Usage

This constructor is called by the `Server.createDataTableSession` method. Subclasses should implement a constructor with the same signature as this one.

DataTable< DTOBJ extends DataTableObject > class

The `DataTable` class in the AJAPI encapsulates the synchronization of a data table.

Syntax

```
public abstract class DataTable< DTOBJ extends
DataTableObject > extends AgentryJavaBackEndManagedObject
```

Members

All members of `DataTable< DTOBJ extends DataTableObject >`, including inherited members. **Variables**

Modifier and Type	Variable	Description
protected SycloCalendar	<code>_clientLastDataUpdateTime</code> on page 694	Client's last data update date.
protected DataTableSession	<code>_session</code> on page 694	Storage for the session that was passed into the constructor.

Methods

Modifier and Type	Method	Description
public	<i>DataTable(DataTableSession, GregorianCalendar)</i> on page 692	Constructs a new DataTable.
public SycloCalendar	<i>getClientLastDataUpdateTime()</i> on page 692	Returns the client's last data update date and time.
public DataTableSession	<i>getSession()</i> on page 692	Returns the session data for this data table transmission.
public void	<i>initialize()</i> on page 693	This method is called after this object is constructed.
public abstract boolean	<i>isOutOfDate()</i> on page 693	Returns whether the client's data is out of date, based on the last update date passed in via the constructor.
public abstract Iterator< DTOBJ >	<i>iterator()</i> on page 693	Builds an Iterator object that will iterate through the DataTableObject objects that contain the rows for the data table.

Usage

When a data table is created in the Agentry Editor, a subclass is automatically defined that extends this class. The designer must implement this subclass in order to retrieve the data for the data table. Such an implementation should retrieve the data table's data from some source, construct a list or set of DataTableObject objects, and then return an iterator object that iterates over that list.

The DataTable class provides two main pieces of functionality. First, it determines if the data table needs to be reloaded and, second, it provides the factory method to construct a Iterator object to process the returned data.

The Agentry server will call the methods of this class in the following sequence:

1. Constructor
2. initialize()
3. isOutOfDate()
4. iterator() (only if isOutOfDate() returned true)

Note that for each client transmit, a new instance of this class is created by the Agentry server.

DataTable(DataTableSession, GregorianCalendar) method

Constructs a new DataTable.

Syntax

```
public    DataTable ( DataTableSession session ,    GregorianCalendar  
clientLastDataUpdate )
```

Parameters

- **session** – An object of type DataTableSession that provides access to pertinent session data for the data table.
- **clientLastDataUpdate** – This argument contains the date and time that the data table was last updated on the client application. This value can be accessed through the protected member `_clientLastDataUpdateTime`.

Usage

This method is called by the Agentry Server (indirectly via whatever subclasses you define) whenever a data table request is received from the client application. Extensions of this class should take the same arguments as this class, and pass those arguments on to this parent constructor.

getClientLastDataUpdateTime() method

Returns the client's last data update date and time.

Syntax

```
public    SycloCalendar    getClientLastDataUpdateTime ()
```

Returns

the date and time of the client's last update. This might be the Agentry "Invalid Date" time, which indicates that the client did not have any copy of the data table at all; note that it is generally safe to not check for the "invalid date" value, since it is a time far in the past (circa 1900) and therefore should fail any check against a recent date and time.

getSession() method

Returns the session data for this data table transmission.

Syntax

```
public    DataTableSession    getSession ()
```

Returns

the session data

initialize() method

This method is called after this object is constructed.

Syntax

```
public void initialize () throws AgentryException
```

Exceptions

- **AgentryException class** – if an error occurs

Usage

Initialization can be performed here if it might result in an exception.

isOutOfDate() method

Returns whether the client's data is out of date, based on the last update date passed in via the constructor.

Syntax

```
public abstract boolean isOutOfDate () throws AgentryException
```

Returns

true if the client's data is out of date, or false if not.

Exceptions

- **AgentryException class** – if an error occurs

Usage

If this returns true, then the Agentry Server will call iterator() to retrieve the latest data for the data table and send it to the client.

iterator() method

Builds an Iterator object that will iterate through the DataTableObject objects that contain the rows for the data table.

Syntax

```
public abstract Iterator< DTOBJ > iterator () throws  
AgentryException
```

Returns

an Iterator object.

Exceptions

- **AgentryException class** – if an error occurs.

Usage

This is called by the Agentry Server to retrieve the actual data for the data table, when the `isOutOfDate()` method returns `true`. Note that once the data has been sent to the client, the server will change the last update date and time on the client to "now".

_clientLastDataUpdateTime variable

Client's last data update date.

Syntax

```
protected SycloCalendar _clientLastDataUpdateTime
```

_session variable

Storage for the session that was passed into the constructor.

Syntax

```
protected DataTableSession _session
```

FatalTransactionException class

This exception can be thrown from a transactional Steplet to indicate that the transaction failed in a way that cannot be corrected or recovered from.

Syntax

```
public class FatalTransactionException extends
AgentryException
```

Members

All members of `FatalTransactionException`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>FatalTransactionException(String)</i> on page 696	Constructs a new <code>FatalTransactionException</code> object.
public	<i>FatalTransactionException(String, Throwable)</i> on page 696	Constructs a new <code>FatalTransactionException</code> object.
public	<i>FatalTransactionException(String, String, String)</i> on page 696	Constructs a new <code>FatalTransactionException</code> object that will cause an error notification window to appear on the client.

Modifier and Type	Constructor	Description
public	<i>FatalTransactionException(String, String, String, Throwable)</i> on page 697	Constructs a new FatalTransactionException object that will cause an error notification window to appear on the client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

It will remove the transaction from the client and save it to the failed transactions queue on the Agentry Server. It will optionally display an error notification window to the client user.

This exception combines and replaces the functionality of the `SycloFatalExceptionWithMessage` and `SycloFatalExceptionWithoutMessage` exceptions from version 4 of the AJAPI.

FatalTransactionException(String) constructor

Constructs a new FatalTransactionException object.

Syntax

```
public FatalTransactionException ( String message )
```

Parameters

- **message** – The error message, which will be logged

Usage

No error notification window will be displayed on the client.

This constructor creates an exception that results in a "Fatal without message" transaction error.

FatalTransactionException(String, Throwable) constructor

Constructs a new FatalTransactionException object.

Syntax

```
public FatalTransactionException ( String message , Throwable  
cause )
```

Parameters

- **message** – The error message, which will be logged
- **cause** – The underlying exception

Usage

No error notification window will be displayed on the client.

This constructor creates an exception that results in a "Fatal without message" transaction error.

FatalTransactionException(String, String, String) constructor

Constructs a new FatalTransactionException object that will cause an error notification window to appear on the client.

Syntax

```
public FatalTransactionException ( String title , String text ,  
String okLabel )
```

Parameters

- **title** – The window title for the error displayed on the client.

- **text** – The window text for the error displayed on the client.
- **okLabel** – The label for the acknowledgment button in the client's error window.

Usage

This constructor creates an exception that results in a "Fatal with message" transaction error.

FatalTransactionException(String, String, String, Throwable) constructor

Constructs a new FatalTransactionException object that will cause an error notification window to appear on the client.

Syntax

```
public FatalTransactionException ( String title , String text ,
String okLabel , Throwable cause )
```

Parameters

- **title** – The window title for the error displayed on the client.
- **text** – The window text for the error displayed on the client.
- **okLabel** – The label for the acknowledgment button in the client's error window.
- **cause** – The causing exception

Usage

This constructor creates an exception that results in a "Fatal with message" transaction error.

FatalTransactionExceptionStop class

This exception can be thrown from a transactional Steplet to indicate that the transaction failed in a way that cannot be corrected or recovered from.

Syntax

```
public class FatalTransactionExceptionStop extends
AgentryException
```

Members

All members of FatalTransactionExceptionStop, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>FatalTransactionExceptionStop(String, String, String)</i> on page 698	Constructs a new FatalTransactionExceptionStop object that will cause an error notification window to appear on the client.

Modifier and Type	Constructor	Description
public	<i>FatalTransactionExceptionStop(String, String, String, Throwable)</i> on page 699	Constructs a new FatalTransactionExceptionStop object that will cause an error notification window to appear on the client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

It will remove the transaction from the client and save it to the failed transactions queue on the Agentry Server. It will display an error notification window to the client user. The transmit will always be stopped (the user will not be given the choice to stop)

FatalTransactionExceptionStop(String, String, String) constructor

Constructs a new FatalTransactionExceptionStop object that will cause an error notification window to appear on the client.

Syntax

```
public FatalTransactionExceptionStop ( String title , String text ,
String okLabel )
```

Parameters

- **title** – The window title for the error displayed on the client.
- **text** – The window text for the error displayed on the client.
- **okLabel** – The label for the acknowledgment button in the client's error window.

Usage

This constructor creates an exception that results in a "Fatal with message, always stop" transaction error.

FatalTransactionExceptionStop(String, String, String, Throwable) constructor

Constructs a new FatalTransactionExceptionStop object that will cause an error notification window to appear on the client.

Syntax

```
public FatalTransactionExceptionStop ( String title , String text ,
String okLabel , Throwable cause )
```

Parameters

- **title** – The window title for the error displayed on the client.
- **text** – The window text for the error displayed on the client.
- **okLabel** – The label for the acknowledgment button in the client's error window.
- **cause** – The causing exception

Usage

This constructor creates an exception that results in a "Fatal with message, always stop" transaction error.

FetchSession class

The FetchSession class encapsulates the processing involved in a fetch.

Syntax

```
public class FetchSession extends Session
```

Members

All members of FetchSession, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>FetchSession(String, Server, SessionData, User)</i> on page 702	Construct a new session.

Methods

Modifier and Type	Method	Description
public void	<i>beginClientExchange()</i> on page 702	This method is called by the server prior to the "Client Exchange Steps" within the fetch are executed.
public void	<i>beginFetchObjectRead()</i> on page 702	This method is called by the server prior to the execution of the "Object Read Steps" for the fetch.
public void	<i>beginFetchRemoval()</i> on page 703	This method is called by the server prior to the "Removal Steps" within the fetch are executed.
public void	<i>beginServerExchange()</i> on page 703	This method is called by the server prior to the "Server Exchange Steps" within the fetch are executed.
public void	<i>endClientExchange()</i> on page 703	This method is called after the "Client Exchange Steps" for the fetch have been successfully completed.
public void	<i>endFetchObjectRead()</i> on page 703	This method is called after the "Object Read Steps" for the fetch have been successfully completed.
public void	<i>endFetchRemoval()</i> on page 704	This method is called after the "Removal Steps" for the fetch have been successfully completed.
public void	<i>endServerExchange()</i> on page 704	This method is called after the "Server Exchange Steps" for the fetch have been successfully completed.

Inherited members from Session

Modifier and Type	Member	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agentry application.
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
protected	<i>Session(String, Server, SessionData, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, SessionData)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

In brief, a fetch is the component of the application that defines how data is synchronized with the back end system. It is made up of steps (which are implemented by the Steplet class in the Java system connection), each of which perform a specific task related to the synchronization process. These steps are organized into groups within the fetch for specific areas of the data synchronization. These areas include the "Object Read", "Client Exchange", "Server Exchange", and "Removal" steps.

This class contains methods to perform actions before and after each of these groups of steps. The default implementation of these methods in the AJAPI perform no additional specific actions. A designer can extend this class if special processing is required before or after each

of these groups of steps are processed. If this class is extended, the `Server` class must also be extended and its `createFetchSession()` method must be overridden to return the designer implemented subclass of the `FetchSession` class.

FetchSession(String, Server, SessionData, User) constructor

Construct a new session.

Syntax

```
public FetchSession ( String fetchName , Server server ,  
SessionData sessionData , User user )
```

Parameters

- **fetchName** – The fetch name, as configured in the Agentry application.
- **server** – The `Server` object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.
- **user** – The client user performing the fetch.

Usage

This constructor is called by the `Server.createFetchSession` method. Subclasses should implement a constructor with the same signature as this one.

beginClientExchange() method

This method is called by the server prior to the "Client Exchange Steps" within the fetch are executed.

Syntax

```
public void beginClientExchange ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginFetchObjectRead() method

This method is called by the server prior to the execution of the "Object Read Steps" for the fetch.

Syntax

```
public void beginFetchObjectRead ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginFetchRemoval() method

This method is called by the server prior to the "Removal Steps" within the fetch are executed.

Syntax

```
public void beginFetchRemoval ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginServerExchange() method

This method is called by the server prior to the "Server Exchange Steps" within the fetch are executed.

Syntax

```
public void beginServerExchange ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

endClientExchange() method

This method is called after the "Client Exchange Steps" for the fetch have been successfully completed.

Syntax

```
public void endClientExchange ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

endFetchObjectRead() method

This method is called after the "Object Read Steps" for the fetch have been successfully completed.

Syntax

```
public void endFetchObjectRead ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

endFetchRemoval() method

This method is called after the "Removal Steps" for the fetch have been successfully completed.

Syntax

```
public void endFetchRemoval ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

endServerExchange() method

This method is called after the "Server Exchange Steps" for the fetch have been successfully completed.

Syntax

```
public void endServerExchange ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

LoginBlockedException class

This exception is thrown from the login methods of the Server class to indicate that the user has been blocked from accessing this system connection.

Syntax

```
public class LoginBlockedException extends LoginException
```

Members

All members of LoginBlockedException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>LoginBlockedException()</i> on page 706	Constructs a new LoginBlockedException object.
public	<i>LoginBlockedException(String)</i> on page 706	Constructs a new LoginBlockedException object.
public	<i>LoginBlockedException(String, Throwable)</i> on page 706	Constructs a new LoginBlockedException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from **AgentryException**

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

It indicates that no further logins should be attempted until the problem is corrected. It can be used, for example, to indicate that the user does not have sufficient privileges to access a remote system, or that the user's account has been locked out due to too many incorrect passwords.

LoginBlockedException() constructor

Constructs a new LoginBlockedException object.

Syntax

```
public LoginBlockedException ()
```

Usage

The client will report a default error message.

LoginBlockedException(String) constructor

Constructs a new LoginBlockedException object.

Syntax

```
public LoginBlockedException ( String message )
```

Parameters

- **message** – The error message, which will be displayed on the client.

LoginBlockedException(String, Throwable) constructor

Constructs a new LoginBlockedException object.

Syntax

```
public LoginBlockedException ( String message , Throwable  
cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

LoginException class

This is the base class for all login/authentication exceptions.

Syntax

```
public class LoginException extends AgentryException
```

Derived classes

- *LoginBlockedException* on page 704
- *LoginSkippedException* on page 709
- *PasswordExpiredCannotChangeException* on page 711
- *PasswordExpiredException* on page 713

- *PasswordInvalidException* on page 716
- *PasswordWarningCannotChangeException* on page 718
- *PasswordWarningException* on page 720

Members

All members of *LoginException*, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from *AgentryException*

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new <i>AgentryException</i> object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new <i>AgentryException</i> object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new <i>AgentryException</i> object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new <i>AgentryException</i> object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.

Modifier and Type	Member	Description
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

Subclasses of this exception are thrown from the login methods of the Server class.

If given message text, this class will place the message text into the `notificationText` field of `AgentryException` as well as using it for the exception's message. This is done to make it easier for the Java Back End to retrieve the unadulterated message text, since the C++ exception wrapper classes sometimes augment the exception's message text.

LoginException() constructor

Constructs a new login exception.

Syntax

```
public    LoginException ()
```

Usage

The client will report a default error message.

LoginException(String) constructor

Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Syntax

```
public    LoginException ( String message )
```

Parameters

- **message** – The error message.

LoginException(String, Throwable) constructor

Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Syntax

```
public    LoginException ( String message ,    Throwable cause )
```

Parameters

- **message** – The error message.
- **cause** – The underlying exception that triggered this exception.

LoginSkippedException class

This exception can be thrown from the login methods of the Server class to indicate that the system connection is not authenticating the user at all, and that some other system connection must be relied upon to perform the authentication.

Syntax

```
public class LoginSkippedException extends LoginException
```

Members

All members of LoginSkippedException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>LoginSkippedException()</i> on page 710	Constructs a new LoginSkippedException object.
public	<i>LoginSkippedException(String)</i> on page 710	Constructs a new LoginSkippedException object.
public	<i>LoginSkippedException(String, Throwable)</i> on page 711	Constructs a new LoginSkippedException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.

Modifier and Type	Member	Description
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

It is functionally equivalent to setting the `enableAuthentication` setting for the Java System connection (in the `Agentry.ini` file) to `false`, except that it can be thrown on a per-user basis.

LoginSkippedException() constructor

Constructs a new `LoginSkippedException` object.

Syntax

```
public LoginSkippedException ()
```

Usage

The client will report a default error message.

LoginSkippedException(String) constructor

Constructs a new `LoginSkippedException` object.

Syntax

```
public LoginSkippedException (String message)
```

Parameters

- **message** – The error message, which will be displayed on the client.

LoginSkippedException(String, Throwable) constructor

Constructs a new LoginSkippedException object.

Syntax

```
public LoginSkippedException ( String message , Throwable
cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

PasswordExpiredCannotChangeException class

This exception can be thrown from the login methods of the Server class to indicate that the user's password has expired, and that this system connection does not support changing it.

Syntax

```
public class PasswordExpiredCannotChangeException extends
LoginException
```

Members

All members of PasswordExpiredCannotChangeException, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	<i>PasswordExpiredCannotChangeException()</i> on page 713	Constructs a new PasswordExpiredCannotChangeException object.
public	<i>PasswordExpiredCannotChangeException(String)</i> on page 713	Constructs a new LoginBlockedException object.
public	<i>PasswordExpiredCannotChangeException(String, Throwable)</i> on page 713	Constructs a new LoginBlockedException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from **AgentryException**

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The user will not be allowed to proceed with their transmission until they change their password via some other means (such as via another system).

This exception should be used instead of `PasswordExpiredException` if the various password-changing methods of `User` have not been implemented.

`PasswordExpiredCannotChangeException()` constructor

Constructs a new `PasswordExpiredCannotChangeException` object.

Syntax

```
public PasswordExpiredCannotChangeException ()
```

Usage

The client will report a default error message.

`PasswordExpiredCannotChangeException(String)` constructor

Constructs a new `LoginBlockedException` object.

Syntax

```
public PasswordExpiredCannotChangeException ( String message )
```

Parameters

- **message** – The error message, which will be displayed on the client.

`PasswordExpiredCannotChangeException(String, Throwable)` constructor

Constructs a new `LoginBlockedException` object.

Syntax

```
public PasswordExpiredCannotChangeException ( String message ,  
Throwable cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

`PasswordExpiredException` class

This exception can be thrown from the login methods of the `Server` class to indicate that the user's password has expired and must be changed before the client's transmission will be allowed to proceed.

Syntax

```
public class PasswordExpiredException extends LoginException
```

Members

All members of `PasswordExpiredException`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>PasswordExpiredException()</i> on page 715	Constructs a new PasswordExpiredException object.
public	<i>PasswordExpiredException(String)</i> on page 715	Constructs a new LoginBlockedException object.
public	<i>PasswordExpiredException(String, Throwable)</i> on page 715	Constructs a new LoginBlockedException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.

Modifier and Type	Member	Description
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The user will be prompted to enter a new password; if they do, the various password-changing methods of the User class will be invoked.

PasswordExpiredException() constructor

Constructs a new PasswordExpiredException object.

Syntax

```
public PasswordExpiredException ()
```

Usage

The client will report a default error message.

PasswordExpiredException(String) constructor

Constructs a new LoginBlockedException object.

Syntax

```
public PasswordExpiredException ( String message )
```

Parameters

- **message** – The error message, which will be displayed on the client.

PasswordExpiredException(String, Throwable) constructor

Constructs a new LoginBlockedException object.

Syntax

```
public PasswordExpiredException ( String message , Throwable cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

PasswordInvalidException class

This exception can be thrown from the various login methods of the Server class to indicate that the user's password was wrong.

Syntax

```
public class PasswordInvalidException extends LoginException
```

Members

All members of PasswordInvalidException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>PasswordInvalidException()</i> on page 717	Constructs a new PasswordInvalidException object.
public	<i>PasswordInvalidException(String)</i> on page 717	Constructs a new PasswordInvalidException object.
public	<i>PasswordInvalidException(String, Throwable)</i> on page 718	Constructs a new PasswordInvalidException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.

Modifier and Type	Member	Description
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

It can also be used to indicate other transitory authentication failures (such as the remote system being unreachable).

PasswordInvalidException() constructor

Constructs a new PasswordInvalidException object.

Syntax

```
public PasswordInvalidException ()
```

Usage

The client will report a default error message.

PasswordInvalidException(String) constructor

Constructs a new PasswordInvalidException object.

Syntax

```
public PasswordInvalidException ( String message )
```

Parameters

- **message** – The error message, which will be displayed on the client.

PasswordInvalidException(String, Throwable) constructor

Constructs a new PasswordInvalidException object.

Syntax

```
public PasswordInvalidException ( String message , Throwable
cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

PasswordWarningCannotChangeException class

This exception can be thrown from the login methods of the Server class to indicate that the user's password is going to expire soon, and that this system connection does not support changing it.

Syntax

```
public class PasswordWarningCannotChangeException extends
LoginException
```

Members

All members of PasswordWarningCannotChangeException, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	<i>PasswordWarningCannotChangeException()</i> on page 720	Constructs a new PasswordWarningCannotChangeException object.
public	<i>PasswordWarningCannotChangeException(String)</i> on page 720	Constructs a new PasswordWarningCannotChangeException object.
public	<i>PasswordWarningCannotChangeException(String, Throwable)</i> on page 720	Constructs a new PasswordWarningCannotChangeException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.

Modifier and Type	Member	Description
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The user will not be allowed to proceed with their transmission until they change their password via some other means (such as via another system).

This exception should be used instead of PasswordWarningException if the various password-changing methods of User have not been implemented.

PasswordWarningCannotChangeException() constructor

Constructs a new PasswordWarningCannotChangeException object.

Syntax

```
public PasswordWarningCannotChangeException ()
```

Usage

The client will report a default error message.

PasswordWarningCannotChangeException(String) constructor

Constructs a new PasswordWarningCannotChangeException object.

Syntax

```
public PasswordWarningCannotChangeException ( String message )
```

Parameters

- **message** – The error message, which will be displayed on the client.

PasswordWarningCannotChangeException(String, Throwable) constructor

Constructs a new PasswordWarningCannotChangeException object.

Syntax

```
public PasswordWarningCannotChangeException ( String message ,  
Throwable cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

PasswordWarningException class

This exception can be thrown from the login methods of the Server class to indicate that the user's password is going to expire soon.

Syntax

```
public class PasswordWarningException extends LoginException
```

Members

All members of PasswordWarningException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>PasswordWarningException()</i> on page 722	Constructs a new PasswordWarningException object.
public	<i>PasswordWarningException(String)</i> on page 722	Constructs a new PasswordWarningException object.
public	<i>PasswordWarningException(String, Throwable)</i> on page 722	Constructs a new PasswordWarningException object.

Inherited members from LoginException

Modifier and Type	Member	Description
public	<i>LoginException()</i> on page 708	Constructs a new login exception.
public	<i>LoginException(String)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.
public	<i>LoginException(String, Throwable)</i> on page 708	Constructs a new login exception with the given error message, which will be passed to the Agentry client.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.

Modifier and Type	Member	Description
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The user will be given the opportunity to change their password in response to this exception; if they do, the various password-changing methods of the User class will be invoked.

PasswordWarningException() constructor

Constructs a new PasswordWarningException object.

Syntax

```
public PasswordWarningException ()
```

Usage

The client will report a default error message.

PasswordWarningException(String) constructor

Constructs a new PasswordWarningException object.

Syntax

```
public PasswordWarningException ( String message )
```

Parameters

- **message** – The error message, which will be displayed on the client.

PasswordWarningException(String, Throwable) constructor

Constructs a new PasswordWarningException object.

Syntax

```
public PasswordWarningException ( String message , Throwable cause )
```

Parameters

- **message** – The error message, which will be displayed on the client.
- **cause** – The exception that triggered this exception.

PushSession class

The PushSession class encapsulates the user-independent part of the processing involved in a push within an Agentry-based application.

Syntax

```
public class PushSession extends Session
```

Members

All members of PushSession, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>PushSession(String, Server, SessionData)</i> on page 726	Construct a new session.

Methods

Modifier and Type	Method	Description
public final void	<i>beginPushError()</i> on page 726	Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession. This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using PushUserSession.
public void	<i>beginPushReadStep()</i> on page 727	This method is called by the server prior to the execution of the "Object Read" steps for the push.
public void	<i>beginPushRemoval()</i> on page 727	This method is called by the server prior to the execution of the "Removal" steps for the push.

Modifier and Type	Method	Description
public final void	<i>beginPushResponse()</i> on page 727	Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of <code>PushUserSession</code> . This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using <code>PushUserSession</code> .
public void	<i>beginPushRetrieval()</i> on page 728	This method is called by the server prior to the execution of the "Retrieval" steps for the push.
public final void	<i>endPushError()</i> on page 728	Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of <code>PushUserSession</code> . This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using <code>PushUserSession</code> .
public void	<i>endPushReadStep()</i> on page 728	This method is called by the server after the "Object Read" Steps for the push have been successfully executed.
public void	<i>endPushRemoval()</i> on page 729	This method is called by the server after the "Removal" steps for the push have been successfully executed.
public final void	<i>endPushResponse()</i> on page 729	Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of <code>PushUserSession</code> . This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using <code>PushUserSession</code> .

Modifier and Type	Method	Description
public void	<i>endPushRetrieval()</i> on page 729	This method is called by the server after the "Retrieval" steps for the push have been successfully executed.

Inherited members from Session

Modifier and Type	Member	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agentry application.
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
protected	<i>Session(String, Server, SessionData, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, SessionData)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

In brief, a push is an application component that defines the transfer of data between the client and the server. Unlike fetches, the transfer within a push is initiated by the server, rather than

the client. A push is made up of steps, grouped in varying categories. The steps in each of these groups are run separately. The groups are "Retrieval", "Removal", "Object Read", "Response" and "Error"; this class handles the "Retrieval", "Removal", and "Object Read" step groups, as those are independent of users. The remaining groups, "Response" and "Error", are user-dependent and are handled by the `PushUserSession` class.

This class contains methods to perform actions before and after each of these groups of steps. The default implementation of these methods in the `AJAPI` perform no additional specific actions. A designer can extend this class if special processing is required before or after each of these groups of steps are processed. If this class is extended, the `Server` class must also be extended and its `createPushSession` method must be overridden to return the new subclass.

PushSession(String, Server, SessionData) constructor

Construct a new session.

Syntax

```
public PushSession (String pushName , Server server , SessionData  
sessionData )
```

Parameters

- **pushName** – The fetch name, as configured in the Agentry application.
- **server** – The `Server` object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.

Usage

This constructor is called by the `Server.createPushSession` method. Subclasses should implement a constructor with the same signature as this one.

beginPushError() method [deprecated]

Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of `PushUserSession`. This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using `PushUserSession`.

Syntax

```
public final void beginPushError ()
```

Usage

DeprecatedAs of Agentry 5.2.8, code for this method should be moved into a subclass of `PushUserSession`.

beginPushReadStep() method

This method is called by the server prior to the execution of the "Object Read" steps for the push.

Syntax

```
public void beginPushReadStep () throws AgentryException
```

Exceptions

- **AgentryException class** – if the push should be aborted for some reason. Throwing an exception from this method will prevent the object read steps from being executed on other system connections as well, and will cause sessionAborted to be invoked.

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginPushRemoval() method

This method is called by the server prior to the execution of the "Removal" steps for the push.

Syntax

```
public void beginPushRemoval () throws AgentryException
```

Exceptions

- **AgentryException class** – if the push should be aborted for some reason. Throwing an exception from this method will prevent the removal steps from being executed on other system connections as well, and will cause sessionAborted to be invoked.

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginPushResponse() method [deprecated]

Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession. This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using PushUserSession.

Syntax

```
public final void beginPushResponse ()
```

Usage

DeprecatedAs of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession.

beginPushRetrieval() method

This method is called by the server prior to the execution of the "Retrieval" steps for the push.

Syntax

```
public void beginPushRetrieval () throws AgentryException
```

Exceptions

- **AgentryException class** – if the push should be aborted for some reason. Throwing an exception from this method will prevent the retrieval steps from being executed on other system connections as well, and will cause sessionAborted to be invoked.

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

endPushError() method [deprecated]

Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession. This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using PushUserSession.

Syntax

```
public final void endPushError ()
```

Usage

DeprecatedAs of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession.

endPushReadStep() method

This method is called by the server after the "Object Read" Steps for the push have been successfully executed.

Syntax

```
public void endPushReadStep ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

endPushRemoval() method

This method is called by the server after the "Removal" steps for the push have been successfully executed.

Syntax

```
public void endPushRemoval ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

endPushResponse() method [deprecated]

Deprecated. As of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession. This method is no longer supported; it is here as a final declaration in order to force subclasses to be converted over to using PushUserSession.

Syntax

```
public final void endPushResponse ()
```

Usage

DeprecatedAs of Agentry 5.2.8, code for this method should be moved into a subclass of PushUserSession.

endPushRetrieval() method

This method is called by the server after the "Retrieval" steps for the push have been successfully executed.

Syntax

```
public void endPushRetrieval ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

PushUserSession class

The PushSession class encapsulates the user-independent part of the processing involved in a push within an Agentry-based application.

Syntax

```
public class PushUserSession extends Session
```

Members

All members of PushUserSession, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>PushUserSession(String, Server, SessionData, User)</i> on page 732	Construct a new session.

Methods

Modifier and Type	Method	Description
public void	<i>beginDisablePush()</i> on page 733	This method is called by the server prior to disabling a user push on any of the system connections.
public void	<i>beginEnablePush()</i> on page 733	This method is called by the server prior to enabling a user push on any of the system connections.
public void	<i>beginPushError()</i> on page 733	This method is called by the server prior to the execution of the "Error Steps" for the push.
public void	<i>beginPushResponse()</i> on page 734	This method is called by the server prior to the execution of the "Response Steps" for the push.
public void	<i>disablePush()</i> on page 734	This method is called when a user requests that a push be disabled on their behalf.
public void	<i>enablePush()</i> on page 734	This method is called when a user requests that a push be enabled on their behalf.
public void	<i>endDisablePush()</i> on page 734	This method is called after a user push has been disabled on all of the system connections.
public void	<i>endEnablePush()</i> on page 735	This method is called after a user push has been enabled on all of the system connections.

Modifier and Type	Method	Description
public void	<i>endPushError()</i> on page 735	This method is called by the server after the "Error Steps" for the push have been successfully executed.
public void	<i>endPushResponse()</i> on page 735	This method is called by the server after the "Response Steps" for the push have been successfully executed.

Inherited members from Session

Modifier and Type	Member	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agentry application.
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
protected	<i>Session(String, Server, SessionData, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, SessionData)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.

Modifier and Type	Member	Description
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

In brief, a push is an application component that defines the transfer of data between the client and the server. Unlike fetches, the transfer within a push is initiated by the server, rather than the client. A push is made up of steps, grouped in varying categories. The steps in each of these groups are run separately. The groups are "Retrieval", "Removal", "Object Read", "Response" and "Error"; this class handles the "Response" and "Error" groups, which are user-dependent.

This class contains methods to perform actions before and after each of these groups of steps. The default implementation of these methods in the AJ-API perform no additional specific actions. A designer can extend this class if special processing is required before or after each of these groups of steps are processed. If this class is extended, the `Server` class must also be extended and its `createPushUserSession` method must be overridden to return the new subclass.

This class also provides methods for notifying the Agentry server when it should enable or disable push events for a particular user.

PushUserSession(String, Server, SessionData, User) constructor

Construct a new session.

Syntax

```
public PushUserSession ( String pushName , Server server ,
SessionData sessionData , User user )
```

Parameters

- **pushName** – The fetch name, as configured in the Agentry application.
- **server** – The `Server` object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.
- **user** – The client user performing the fetch.

Usage

This constructor is called by the `Server.createPushUserSession` method. Subclasses should implement a constructor with the same signature as this one.

beginDisablePush() method

This method is called by the server prior to disabling a user push on any of the system connections.

Syntax

```
public void beginDisablePush () throws AgentryException
```

Exceptions

- **AgentryException class** – to block disabling of the user push.

Usage

It can be used to start a remote transaction.

beginEnablePush() method

This method is called by the server prior to enabling a user push on any of the system connections.

Syntax

```
public void beginEnablePush () throws AgentryException
```

Exceptions

- **AgentryException class** – if an error occurs.

Usage

It can be used to start a remote transaction.

beginPushError() method

This method is called by the server prior to the execution of the "Error Steps" for the push.

Syntax

```
public void beginPushError () throws AgentryException
```

Exceptions

- **AgentryException class** – if the push should be aborted for some reason. Throwing an exception from this method will prevent the response steps from being executed on other system connections as well, and will cause sessionAborted to be invoked.

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginPushResponse() method

This method is called by the server prior to the execution of the "Response Steps" for the push.

Syntax

```
public void beginPushResponse () throws AgentryException
```

Exceptions

- **AgentryException class** – if the push should be aborted for some reason. Throwing an exception from this method will prevent the response steps from being executed on other system connections as well, and will cause sessionAborted to be invoked.

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

disablePush() method

This method is called when a user requests that a push be disabled on their behalf.

Syntax

```
public void disablePush () throws AgentryException
```

Exceptions

- **AgentryException class** – if an error occurs.

enablePush() method

This method is called when a user requests that a push be enabled on their behalf.

Syntax

```
public void enablePush () throws AgentryException
```

Exceptions

- **AgentryException class** – if an error occurs.

endDisablePush() method

This method is called after a user push has been disabled on all of the system connections.

Syntax

```
public void endDisablePush ()
```

Usage

It can be used to commit a remote transaction. It cannot fail.

endEnablePush() method

This method is called after a user push has been enabled on all of the system connections.

Syntax

```
public void endEnablePush ()
```

Usage

It can be used to commit a remote transaction. It cannot fail.

endPushError() method

This method is called by the server after the "Error Steps" for the push have been successfully executed.

Syntax

```
public void endPushError ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

endPushResponse() method

This method is called by the server after the "Response Steps" for the push have been successfully executed.

Syntax

```
public void endPushResponse ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

RetryTransactionException class

This exception can be thrown from a transactional Steplet to indicate that the transaction failed temporarily, and that it should be retried by the client.

Syntax

```
public class RetryTransactionException extends  
AgentryException
```

Members

All members of RetryTransactionException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>RetryTransactionException(String, String, String)</i> on page 737	Constructs a new RetryTransactionException object.
public	<i>RetryTransactionException(String, String, String, Throwable)</i> on page 737	Constructs a new RetryTransactionException object.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The client will resend the transaction immediately, and if it fails a second time then the transaction will be marked as fatal.

This exception is equivalent to the "Retry Without Change" transaction error fatality.

RetryTransactionException(String, String, String) constructor

Constructs a new `RetryTransactionException` object.

Syntax

```
public RetryTransactionException ( String title , String
notification , String okButtonLabel )
```

Parameters

- **title** – The window title for the notification displayed on the client.
- **notification** – The window text for the notification displayed on the client.
- **okButtonLabel** – The label for the acknowledgment button in the client's notification window.

RetryTransactionException(String, String, String, Throwable) constructor

Constructs a new `RetryTransactionException` object.

Syntax

```
public RetryTransactionException ( String title , String text ,
String okLabel , Throwable cause )
```

Parameters

- **title** – The window title for the error displayed on the client.
- **text** – The window text for the error displayed on the client.
- **okLabel** – The label for the acknowledgment button in the client's error window.
- **cause** – The causing exception

RetryTransactionWithChangeException class

This exception can be thrown from a transactional Steplet to indicate that the transaction failed in a correctable manner, and that it should be retried by the client after prompting the user to make changes to the transaction.

Syntax

```
public class RetryTransactionWithChangeException extends
AgentryException
```

Members

All members of `RetryTransactionWithChangeException`, including inherited members.

Constructors

Modifier and Type	Constructor	Description
public	<i>RetryTransactionWithChangeException(String, String, String)</i> on page 739	Constructs a new RetryTransactionWithChangeException object.
public	<i>RetryTransactionWithChangeException(String, String, String, Throwable)</i> on page 739	Constructs a new RetryTransactionWithChangeException object.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

This exception is equivalent to the "Retry With Change" transaction error fatality.

RetryTransactionWithChangeException(String, String, String) constructor

Constructs a new `RetryTransactionWithChangeException` object.

Syntax

```
public    RetryTransactionWithChangeException ( String title ,
String text ,   String okLabel )
```

Parameters

- **title** – The window title for the notification displayed on the client.
- **text** – The text for the notification window displayed on the client, which should describe to the user what changes they need to make in order for the transaction to succeed.
- **okLabel** – The label for the acknowledgment button in the client's notification window.

RetryTransactionWithChangeException(String, String, String, Throwable) constructor

Constructs a new `RetryTransactionWithChangeException` object.

Syntax

```
public    RetryTransactionWithChangeException ( String title ,
String text ,   String okLabel ,   Throwable cause )
```

Parameters

- **title** – The window title for the error displayed on the client.
- **text** – The window text for the error displayed on the client.
- **okLabel** – The label for the acknowledgment button in the client's error window.
- **cause** – The causing exception

Server class

The `Server` Java class is intended to encapsulate the Java system connection within the Agentry Server.

Syntax

```
public class Server extends AgentryJavaBackEndManagedObject
```

Members

All members of `Server`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>Server()</i> on page 750	Constructs a new <code>Server</code> object.

Methods

Modifier and Type	Method	Description
public ComplexTableSession	<i>createComplexTableSession(String, SessionData, User)</i> on page 750	Factory method that creates a new ComplexTableSession object.
public DataTableSession	<i>createDataTableSession(String, SessionData, User)</i> on page 751	Factory method that creates a new DataTableSession object.
public final FetchSession	<i>createFetchSession(String, Server, SessionData, User)</i> on page 752	Deprecated. Use createFetchSession(String, SessionData, User) (i.e. remove the Server argument). This method is no longer supported; override createFetchSession(String, SessionData, User) instead.
public FetchSession	<i>createFetchSession(String, SessionData, User)</i> on page 752	Factory method that creates a new FetchSession object.
public final FetchSession	<i>createPushSession(String, Server, SessionData)</i> on page 753	Deprecated. Use createPushSession(String, SessionData) (i.e. remove the Server argument). This method is no longer supported; override createPushSession(String, SessionData) instead.
public PushSession	<i>createPushSession(String, SessionData)</i> on page 753	Factory method that creates a new PushSession object for a push session that is not tied to a specific user.
public final FetchSession	<i>createPushUserSession(String, Server, SessionData, User)</i> on page 754	Deprecated. Use createPushUserSession(String, SessionData, User) (i.e. remove the Server argument). This method is no longer supported; override createPushUserSession(String, SessionData, User) instead.
public PushUserSession	<i>createPushUserSession(String, SessionData, User)</i> on page 754	Factory method that creates a new PushUserSession object for a push session.

Modifier and Type	Method	Description
public final ServiceEventSession	<i>createServiceEventSession(String, Server, SessionData)</i> on page 755	Deprecated. Use createServiceEventSession(String, SessionData) (i.e. remove the Server argument). This method is no longer supported; override createServiceEventSession(String, SessionData) instead.
public ServiceEventSession	<i>createServiceEventSession(String, SessionData)</i> on page 756	Factory method that creates a new ServiceEventSession object.
public final FetchSession	<i>createTransactionSession(String, Server, SessionData, User)</i> on page 756	Deprecated. Use createTransactionSession(String, SessionData, User) (i.e. remove the Server argument). This method is no longer supported; override createTransactionSession(String, SessionData, User) instead.
public TransactionSession	<i>createTransactionSession(String, SessionData, User)</i> on page 757	Factory method that creates a new TransactionSession object.
public final User	<i>createUser(String, int)</i> on page 757	Deprecated. Use createUser(String) instead. This method is no longer supported; override createUser(String) instead.
public User	<i>createUser(String)</i> on page 758	Factory method that creates a new User object.
public final void	<i>debug(String)</i> on page 758	Writes a debug message to the Agentry Server's Java System Connection log file.
public final String	<i>decryptPassword(String)</i> on page 759	Decodes a password that has been encoded via Agentry's quickPW utility.
public File	<i>findConfigurationFile(String)</i> on page 759	Locates a configuration file in the Agentry application's deployment returns a.

Modifier and Type	Method	Description
public static final String	<i>getImplementationVersion()</i> on page 759	Retrieves the implementation version of the AJAPI release that this Server class is from.
public static Server	<i>getInstance()</i> on page 760	Return the singleton instance of this class.
public static final String	<i>getSpecificationVersion()</i> on page 760	Retrieves the specification version of the AJAPI that this Server implements.
public String	<i>getTimeZone()</i> on page 760	This is called by the Agentry server to find out what time zone is being used by whatever remote server this implementation is communicating with.
public LoginEnumeration	<i>login(String, String, SessionData)</i> on page 761	Deprecated. Override login(User, String, SessionData) instead. This method is called when a user initially connects to the Agentry Server from a client application and that server's system connection's enableAuthentication option is set to true in the Agentry.ini file.
public void	<i>login(User, String, SessionData)</i> on page 761	This method authenticates a client user against the Java System Connection.

Modifier and Type	Method	Description
public LoginEnumeration	<i>loginBlocked(String, StringBuffer)</i> on page 762	Deprecated. Override loginBlocked(User, String, StringBuffer, SessionData) instead. This method is called by the Agentry Server when authentication of a client user is blocked, either because this class returned a blocked login from the login(String, String, SessionData) or loginPreviousUser(String, String, SessionData) methods, or because another system connection blocked the login.
public void	<i>loginBlocked(User, StringBuffer, SessionData)</i> on page 763	Deprecated. Override loginBlocked(User, String, StringBuffer, SessionData) instead. This method is called by the Agentry Server when authentication of a client user is blocked, either because this class threw LoginBlockedException from the login, loginPreviousUser, or loginFailed methods, or because another system connection blocked the login.
public void	<i>loginBlocked(User, String, StringBuffer, SessionData)</i> on page 763	This method is called by the Agentry Server when authentication of a client user is blocked, either because this class threw LoginBlockedException from the login, loginPreviousUser, or loginFailed methods, or because another system connection blocked the login.

Modifier and Type	Method	Description
public LoginEnumeration	<i>loginFailed(String, StringBuffer)</i> on page 764	Deprecated. Override loginFailed(User, String, LoginFailureReason, StringBuffer, SessionData) instead. This method is called by the Agentry Server when authentication of a client user fails, either because this class returned a failed login from the login or loginPreviousUser methods, or because another system connection failed the login.
public void	<i>loginFailed(User, String, LoginFailureReason, StringBuffer, SessionData)</i> on page 765	This method is called by the Agentry Server when authentication of a client user fails, either because this class threw PasswordInvalidException from the login or loginPreviousUser methods, or because another system connection reported a login failure.
public LoginEnumeration	<i>loginPreviousUser(String, String, SessionData)</i> on page 766	Deprecated. Override loginPreviousUser(User, String, SessionData) instead. This method is called when a user has previously logged into Agentry successfully, and is now logging in again due to having been disconnected.
public void	<i>loginPreviousUser(User, String, SessionData)</i> on page 766	This method is called when a user has previously logged into Agentry successfully, and is now logging in again due to having been disconnected.
final void	<i>setDebugEnabled(boolean)</i> on page 767	Deprecated. This is only here because the Agentry server will call it. Setter method called by the Agentry server to enable/disable debugging.

Modifier and Type	Method	Description
public void	<i>shutdown()</i> on page 767	This method is called by the Agentry Server when the Java system connection is being shut down.
public void	<i>startup()</i> on page 767	This method is called by the Agentry Server when the Java system connection starts up and creates an instance of this class; it is called immediately after the class is constructed.

Usage

The bulk of the methods within this class are factory methods for various other object types. If the designer overrides one of the classes constructed by these factory methods, a subclass of the `Server` class must also be created. This implementation must override the appropriate factory methods to construct objects of the appropriate type.

In addition to these factory methods, there are also methods related to login and logout, server startup and shutdown, and debugging. By default, these methods perform no application-related processing; they merely print a message to the debug log indicating that these events have or are about to occur. If additional processing is required for an application, these methods should be overridden in a subclass of the `Server` class.

When an Agentry Server with a Java System Connection is started, the server will construct a singleton instance of the `Server` class or a subclass of it, as specified by the `serverClass` setting in the Java system connection section of the `Agentry.ini` file. This `Server` object will persist until the Agentry Server is shutdown. If the designer has implemented an extension of the `Server` class, this new class must be named in the `serverClass` option in the `Agentry.ini` file in the section containing the configuration options for the Java Interface system connection.

Note, the constructor is public even though this class is a singleton. This is mainly for legacy reasons: the `Agentry.ini` file names a subclass of this class instead of naming a factory class, and we kept it that way rather than changing it to take a factory class name to maintain compatibility with AJAPI version 4 (since the server supports both versions of the AJAPI).

As such, in this version of the AJAPI, this class is not a strict singleton, although it should be treated as such. Applications should not try to create another instance of it, although it is acceptable (since it is hard to avoid) to do so in unit tests.

Eventually, this class will likely become a true singleton and the Agentry server will adopt the factory pattern to instantiate it, so you may wish to do so now in unit tests using a factory similar to:

```
public class ServerFactory
{
    private static class LazyHolder
    {
        private static final Server _instance = new Server();
    }

    public static Server getServerInstance()
    {
        return LazyHolder._instance;
    }
}
```

(Note, this example follows the "Initialization on Demand Holder" pattern; see [this article on Wikipedia](#) for information on how/why it works.)

Server.LoginEnumeration enum

Deprecated. These constants are only used by the deprecated login methods. New code should be fixed to use the new exception-based login methods. Return values for the login method.

Members

All members of LoginEnumeration, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>Login_Invalid</i> on page 747	Indicates that the login and password are not valid for any user profile on the system or that the login attempt failed for some reason other than the user being blocked, or the user having an expired password.
public	<i>Login_InvalidBlocked</i> on page 748	Indicates that the user has been blocked from accessing the remote system.
public	<i>Login_Pass</i> on page 748	Indicates that the user was not validated by the system connection.
public	<i>Login_Valid</i> on page 748	Indicates that the user ID and password are valid and that the user should be allowed to proceed with their transmission to the Agentry Server.

Modifier and Type	Variable	Description
public	<i>Login_ValidPasswordExpired</i> on page 748	Indicates that the user ID is valid but that the password has expired.
public	<i>Login_ValidPasswordExpired-NoChange</i> on page 748	Indicates that the user ID is valid, but that the password has expired.
public	<i>Login_ValidPasswordWarning</i> on page 749	Indicates that user ID and password values are valid, but that the password will be expiring in the near future.
public	<i>Login_ValidPasswordWarning-NoChange</i> on page 749	Indicates that the user ID and password values are valid, but that the password will be expiring in the near future.

Methods

Modifier and Type	Method	Description
public void	<i>throwException()</i> on page 747	Throws an exception that corresponds to this enum value.

throwException() method

Throws an exception that corresponds to this enum value.

Syntax

```
public void throwException () throws LoginException
```

Exceptions

- **LoginException** class – the exception that corresponds to this enum.

Login_Invalid variable

Indicates that the login and password are not valid for any user profile on the system or that the login attempt failed for some reason other than the user being blocked, or the user having an expired password.

Syntax

```
public Login_Invalid
```

Login_InvalidBlocked variable

Indicates that the user has been blocked from accessing the remote system.

Syntax

```
public    Login_InvalidBlocked
```

Login_Pass variable

Indicates that the user was not validated by the system connection.

Syntax

```
public    Login_Pass
```

Usage

Another system connection must validate the user before the user will be allowed to access the Agentry Server from the client application. Returning this value has the same effect as setting `enableAuthentication` to `false` in the Java section of `Agentry.ini`, except that it can be used on a per-user basis.

Login_Valid variable

Indicates that the user ID and password are valid and that the user should be allowed to proceed with their transmission to the Agentry Server.

Syntax

```
public    Login_Valid
```

Login_ValidPasswordExpired variable

Indicates that the user ID is valid but that the password has expired.

Syntax

```
public    Login_ValidPasswordExpired
```

Usage

The user will be prompted to change their password on the client application before being allowed to proceed with the transmission.

Login_ValidPasswordExpiredNoChange variable

Indicates that the user ID is valid, but that the password has expired.

Syntax

```
public    Login_ValidPasswordExpiredNoChange
```

Usage

The user will not be allowed to change the password from within the Agentry-based application and will not be allowed to proceed with their transmission until the password has been updated.

Login_ValidPasswordWarning variable

Indicates that user ID and password values are valid, but that the password will be expiring in the near future.

Syntax

```
public    Login_ValidPasswordWarning
```

Usage

The user will be prompted to change his or her password on the client application, but can bypass the change and still be allowed to proceed with the transmission to the Agentry Server.

Login_ValidPasswordWarningNoChange variable

Indicates that the user ID and password values are valid, but that the password will be expiring in the near future.

Syntax

```
public    Login_ValidPasswordWarningNoChange
```

Usage

The user will be notified that their password is near its expiration, but will not be allowed to modify the password value from within the Agentry-based application.

Server.LoginFailureReason enum

These are used in loginFailed to indicate the reason why a login failed.

Members

All members of LoginFailureReason, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>NoBackEndsAuthenticated</i> on page 750	Indicates that the login failed because no back-ends were configured to authenticate.
public	<i>PasswordExpiredCannotChange</i> on page 750	Indicates that the login failed because a password was expired and Agentry cannot change it.

Modifier and Type	Variable	Description
public	<i>PasswordInvalid</i> on page 750	Indicates that the login failed due to an invalid password.

NoBackEndsAuthenticated variable

Indicates that the login failed because no back-ends were configured to authenticate.

Syntax

```
public    NoBackEndsAuthenticated
```

PasswordExpiredCannotChange variable

Indicates that the login failed because a password was expired and Agentry cannot change it.

Syntax

```
public    PasswordExpiredCannotChange
```

PasswordInvalid variable

Indicates that the login failed due to an invalid password.

Syntax

```
public    PasswordInvalid
```

Server() constructor

Constructs a new `Server` object.

Syntax

```
public    Server ()
```

Usage

This constructor is called by the Agentry server when the Java Interface system connection is started.

See the main class documentation for notes on why this constructor is public even though this class is a singleton.

createComplexTableSession(String, SessionData, User) method

Factory method that creates a new `ComplexTableSession` object.

Syntax

```
public ComplexTableSession createComplexTableSession (String


```

Parameters

- **tableName** – The name of the complex table being processed, as configured by the designer in the Agentry Editor.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.
- **user** – Represents the client user that is retrieving the complex table.

Returns

A new ComplexTableSession object.

Usage

This method is called by the Agentry Server whenever a complex table synchronization is to be processed. If the ComplexTableSession class is extended, then this method must be overridden to return the new subclass.

createDataTableSession(String, SessionData, User) method

Factory method that creates a new DataTableSession object.

Syntax

```
public DataTableSession createDataTableSession ( String
tableName , SessionData sessionData , User user )
```

Parameters

- **tableName** – The name of the data table being processed, as configured by the designer in the Agentry Editor.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.
- **user** – Represents the client user that is retrieving the complex table.

Returns

A new DataTableSession object.

Usage

This method is called by the Agentry Server whenever a data table retrieval is to be processed. If the DataTableSession class is extended, then this method must be overridden to return the new subclass.

createFetchSession(String, Server, SessionData, User) method [deprecated]
Deprecated. Use createFetchSession(String, SessionData, User) (i.e. remove the Server argument). This method is no longer supported; override createFetchSession(String, SessionData, User) instead.

Syntax

```
public final FetchSession createFetchSession ( String name ,  
Server server , SessionData data , User user )
```

Parameters

- **name** – not used
- **server** – not used
- **data** – not used
- **user** – not used

Returns

nothing, throws UnsupportedOperationException.

Exceptions

- **UnsupportedOperationException** – to indicate that it is no longer supported.

Usage

This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

createFetchSession(String, SessionData, User) method

Factory method that creates a new FetchSession object.

Syntax

```
public FetchSession createFetchSession ( String fetchName ,  
SessionData sessionData , User user )
```

Parameters

- **fetchName** – The name of the fetch being processed, as configured by the designer in the Agentry Editor.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.
- **user** – Represents the client user that is performing the fetch.

Returns

A new FetchSession object.

Usage

It is called by the Agentry Server whenever a fetch is requested by the client application. If the `FetchSession` class is extended, then this method must be overridden to return the new subclass.

createPushSession(String, Server, SessionData) method [deprecated]

Deprecated. Use `createPushSession(String, SessionData)` (i.e. remove the `Server` argument). This method is no longer supported; override `createPushSession(String, SessionData)` instead.

Syntax

```
public final FetchSession createPushSession ( String name ,
Server server , SessionData data )
```

Parameters

- **name** – not used
- **server** – not used
- **data** – not used

Returns

nothing, throws `UnsupportedOperationException`.

Exceptions

- **UnsupportedOperationException** – to indicate that it is no longer supported.

Usage

This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

createPushSession(String, SessionData) method

Factory method that creates a new `PushSession` object for a push session that is not tied to a specific user.

Syntax

```
public PushSession createPushSession ( String pushName ,
SessionData sessionData )
```

Parameters

- **pushName** – The name of the push being processed, as configured by the designer in the Agentry Editor.

- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.

Returns

A new PushSession object.

Usage

It is called by the Agentry Server whenever a push is being processed by the server. If the PushSession class is extended, then this method must be overridden to return the new subclass.

createPushUserSession(String, Server, SessionData, User) method [deprecated]

Deprecated. Use createPushUserSession(String, SessionData, User) (i.e. remove the Server argument). This method is no longer supported; override createPushUserSession(String, SessionData, User) instead.

Syntax

```
public final FetchSession createPushUserSession ( String name ,  
Server server , SessionData data , User user )
```

Parameters

- **name** – not used
- **server** – not used
- **data** – not used
- **user** – not used

Returns

nothing, throws UnsupportedOperationException.

Exceptions

- **UnsupportedOperationException** – to indicate that it is no longer supported.

Usage

This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

createPushUserSession(String, SessionData, User) method

Factory method that creates a new PushUserSession object for a push session.

Syntax

```
public PushUserSession createPushUserSession ( String pushName ,  
SessionData sessionData , User user )
```

Parameters

- **pushName** – The name of the push being processed, as configured by the designer in the Agentry Editor.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.
- **user** – Represents the client user that is performing the push.

Returns

A new PushUserSession object.

Usage

It is called by the Agentry Server whenever the user-specific steps of a push are being processed by the server. If the PushUserSession class is extended, then this method must be overridden to return the new subclass.

createServiceEventSession(String, Server, SessionData) method [deprecated]

Deprecated. Use createServiceEventSession(String, SessionData) (i.e. remove the Server argument). This method is no longer supported; override createServiceEventSession(String, SessionData) instead.

Syntax

```
public final ServiceEventSession createServiceEventSession
( String serviceName , Server server , SessionData sessionData )
```

Parameters

- **serviceName** – not used
- **server** – not used
- **sessionData** – not used

Returns

nothing, throws UnsupportedOperationException.

Exceptions

- **UnsupportedOperationException** – to indicate that it is no longer supported.

Usage

This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

createServiceEventSession(String, SessionData) method

Factory method that creates a new ServiceEventSession object.

Syntax

```
public ServiceEventSession createServiceEventSession ( String  
serviceName , SessionData sessionData )
```

Parameters

- **serviceName** – The name of the service event being processed, as configured by the designer in the Agentry Editor.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.

Returns

A new ServiceEventSession object.

Usage

This method is called by the Agentry Server whenever a service event is to be processed. If the ServiceEventSession class is extended, then this method must be overridden to return the new subclass.

createTransactionSession(String, Server, SessionData, User) method [deprecated]

Deprecated. Use createTransactionSession(String, SessionData, User) (i.e. remove the Server argument). This method is no longer supported; override createTransactionSession(String, SessionData, User) instead.

Syntax

```
public final FetchSession createTransactionSession ( String  
name , Server server , SessionData data , User user )
```

Parameters

- **name** – not used
- **server** – not used
- **data** – not used
- **user** – not used

Returns

nothing, throws UnsupportedOperationException.

Exceptions

- **UnsupportedOperationException** – to indicate that it is no longer supported.

Usage

This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

createTransactionSession(String, SessionData, User) method

Factory method that creates a new TransactionSession object.

Syntax

```
public TransactionSession createTransactionSession (String
transactionName , SessionData sessionData , User user )
```

Parameters

- **transactionName** – The name of the transaction being processed, as configured by the designer in the Agentry Editor.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.
- **user** – Represents the client user that is performing the transaction.

Returns

A new TransactionSession object.

Usage

This method is called by the Agentry Server whenever a transaction is to be processed. If the TransactionSession class is extended, then this method must be overridden to return the new subclass.

createUser(String, int) method [deprecated]

Deprecated. Use createUser(String) instead. This method is no longer supported; override createUser(String) instead.

Syntax

```
public final User createUser (String name , int x)
```

Parameters

- **name** – not used
- **x** – not used

Returns

nothing, throws UnsupportedOperationException.

Exceptions

- **UnsupportedOperationException** – to indicate that it is no longer supported.

Usage

This method is only here to cause compilation errors in legacy code, and may be removed in a future release.

createUser(String) method

Factory method that creates a new User object.

Syntax

```
public User createUser ( String name )
```

Parameters

- **name** – The user name value from the Agentry client application.

Returns

A new User object.

Usage

This method is called by the Agentry Server to create objects that represent client users. If a new subclass of User is created for an application, then this method must be extended to construct the new subclass.

debug(String) method

Writes a debug message to the Agentry Server's Java System Connection log file.

Syntax

```
public final void debug ( String serverMessage )
```

Parameters

- **serverMessage** – The message to log

Usage

This message will only appear in the log file if debug logging is turned on for the Java system connection.

This method is a convenience method that calls into the Java Logging API to do the actual logging, and assumes that Agentry's default Java Logging configuration is in place (which will route log messages back to the Agentry server). It will log to a logger named "com.syclo.agentry.Server", at the FINE level (which translates to log detail level 3 in Agentry).

When invoked outside of Agentry (e.g. in unit tests), this will log to the console, as that is Java's normal default logging configuration.

decryptPassword(String) method

Decodes a password that has been encoded via Agentry's quickPW utility.

Syntax

```
public final String decryptPassword ( String password )
```

Parameters

- **password** – The encoded password.

Returns

The decoded password.

findConfigurationFile(String) method

Locates a configuration file in the Agentry application's deployment returns a.

Syntax

```
public File findConfigurationFile ( String filename )
```

Parameters

- **filename** – the name of the configuration file

Returns

aFile object referencing the file in the correct directory

Usage

File object that can be used to access the file. If the file does not exist, this method will return an object that can be used to create the file in an appropriate location.

getImplementationVersion() method

Retrieves the implementation version of the AJAPI release that this Server class is from.

Syntax

```
public static final String getImplementationVersion ()
```

Returns

The implementation version, which is the same as the full version (including build number) of the Agentry Server that this class's JAR file came with. This will return the empty string if you are running out of raw class files and not a released AJAPI JAR.

Usage

This is the same as the version of the Agentry Server that the AJAPI JAR file came with.

This information is retrieved from the AJAPI JAR manifest file via the Package class.

Note that this does not necessarily reflect the version of the API that this JAR implements; for example, if this JAR implements version 4 of the AJAPI, but it comes with version 5.0.0.3 of the Agentry server, this will return 5.0.0.3, not 4! If you want to know the AJAPI version that is implemented, call `getSpecificationVersion()` instead.

getInstance() method

Return the singleton instance of this class.

Syntax

```
public static Server getInstance ()
```

Returns

the running instance of this class

Usage

Note that this method will not create the server; you have to call the constructor once to do that (see the constructor docs for the reasons why).

getSpecificationVersion() method

Retrieves the specification version of the AJAPI that this Server implements.

Syntax

```
public static final String getSpecificationVersion ()
```

Returns

The AJAPI version that this AJAPI package implements. This will return the empty string if you are running out of raw class files and not a released AJAPI JAR.

Usage

This will tell you what version of the AJAPI you are using, but not what version of Agentry it came with; if you want to know the latter, call `getImplementationVersion()` instead.

getTimeZone() method

This is called by the Agentry server to find out what time zone is being used by whatever remote server this implementation is communicating with.

Syntax

```
public String getTimeZone ()
```

Returns

The timezone name.

Usage

This will override the `timeZoneName` setting in the `Agentry.ini` file.

If this method returns a name that the Agentry server doesn't recognize (and odds are it won't, especially on Windows), then it needs to be mapped in the `[TimeZoneAliases]` section of the `Agentry.ini` file. If this method returns an empty string (the default implementation), then the `timeZoneName` setting in the `Agentry.ini` file will be used.

login(String, String, SessionData) method [deprecated]

Deprecated. Override `login(User, String, SessionData)` instead. This method is called when a user initially connects to the Agentry Server from a client application and that server's system connection's `enableAuthentication` option is set to true in the `Agentry.ini` file.

Syntax

```
public LoginEnumeration login (String userId , String password ,
SessionData sessionData )
```

Parameters

- **userId** – The user ID from the client application for the current user.
- **password** – The password for the current user, as entered on the client application.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.

Returns

One of the constants from `Server.LoginEnumeration`.

Usage

The return value of this function indicates whether or not the user ID and password are valid. By default, this method returns the enumerated value `Login_Pass`, which means that this system connection is not responsible for authenticating the user. Override this method to implement logic to perform full validation of the user against a remote system.

login(User, String, SessionData) method

This method authenticates a client user against the Java System Connection.

Syntax

```
public void login (User user , String password , SessionData
sessionData ) throws LoginException
```

Parameters

- **user** – The User object that identifies the client user. The user name can be read from this object.

- **password** – The password for the current user, as entered on the client application.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.

Exceptions

- **LoginException class** – if the login fails for any reason, or if the login succeeds but an exceptional condition exists (such as an expired or soon-to-be expired password).

Usage

This method is called when a user initially connects to the Agentry Server from a client application and the `enableAuthentication` option is set to `true` in the Java section of the `Agentry.ini` file. Override this method to implement logic to perform full validation of the user against a remote system.

This method should return normally if the authentication of the user succeeds. If login fails for any reason, the appropriate `LoginException` subclass should be thrown. An exception should also be thrown for other conditions such as expired or soon-to-be-expiring passwords. By default, this method throws `LoginSkippedException`, which means that this system connection is not responsible for authenticating the user (it is equivalent to setting `enableAuthentication` to `false` in `Agentry.ini`, except that it can be thrown on a per-user basis).

If you throw `PasswordInvalidException` or `LoginBlockedException`, then either `loginFailed` or `loginBlocked`, respectively, will be called. From those methods you can return a more detailed error message indicating why the login failed.

loginBlocked(String, StringBuffer) method [deprecated]

Deprecated. Override `loginBlocked(User, String, StringBuffer, SessionData)` instead. This method is called by the Agentry Server when authentication of a client user is blocked, either because this class returned a blocked login from the `login(String, String, SessionData)` or `loginPreviousUser(String, String, SessionData)` methods, or because another system connection blocked the login.

Syntax

```
public LoginEnumeration loginBlocked ( String userId ,  
StringBuffer error )
```

Parameters

- **userId** – The user ID for the user whose login attempt has failed.
- **error** – A `StringBuffer` that contains the error message that was returned by the system connection that blocked the login. This message will ultimately be logged on the server and displayed on the client. The error message can be changed by modifying the contents of this buffer.

Returns

Always returns `Server.LoginEnumeration#Login_InvalidBlocked`

Usage

It should clean up any user-related resources; it can also return additional information about why the login failed.

loginBlocked(User, StringBuffer, SessionData) method [deprecated]

Deprecated. Override `loginBlocked(User, String, StringBuffer, SessionData)` instead. This method is called by the Agentry Server when authentication of a client user is blocked, either because this class threw `LoginBlockedException` from the `login`, `loginPreviousUser`, or `loginFailed` methods, or because another system connection blocked the login.

Syntax

```
public void loginBlocked ( User user ,  StringBuffer error ,
SessionData sessionData )
```

Parameters

- **user** – The User object for the user whose login attempt was blocked.
- **error** – A `StringBuffer` that contains the error message that was returned by the system connection that blocked the login. This message will ultimately be logged on the server and displayed on the client. The error message can be changed by modifying the contents of this buffer.
- **sessionData** – The session data. In addition to its usual contents, this data will contain additional information about which system connection blocked the login. This information will be available as the values for the SDML keys `failed.backend.id` (the system connection number) and `failed.backend.name` (the system connection name, as configured in `Agentry.ini`).

Usage

It should clean up any user-related resources.

loginBlocked(User, String, StringBuffer, SessionData) method

This method is called by the Agentry Server when authentication of a client user is blocked, either because this class threw `LoginBlockedException` from the `login`, `loginPreviousUser`, or `loginFailed` methods, or because another system connection blocked the login.

Syntax

```
public void loginBlocked ( User user ,  String userId ,  StringBuffer
error ,  SessionData sessionData )
```

Parameters

- **user** – The User object for the user whose login attempt has blocked. This can be `null`, if the login was blocked by another system connection before the `createUser` and `login` methods of this system connection were called.
- **userId** – The user name of the user that was logging in. This parameter might be useful if `user` is `null`, but you still need to take some sort of action for the user for some reason. If `user` is not `null`, then this parameter will be equal to the user name contained in `user`.
- **error** – A `StringBuffer` that contains the error message that was returned by the system connection that blocked the login. This message will ultimately be logged on the server and displayed on the client. The error message can be changed by modifying the contents of this buffer.
- **sessionData** – The session data. In addition to its usual contents, this data will contain additional information about which system connection blocked the login. This information will be available as the values for the SDML keys `failed.backend.id` (the system connection number) and `failed.backend.name` (the system connection name, as configured in `Agentry.ini`).

Usage

It should clean up any user-related resources.

loginFailed(String, StringBuffer) method [deprecated]

Deprecated. Override `loginFailed(User, String, LoginFailureReason, StringBuffer, SessionData)` instead. This method is called by the Agentry Server when authentication of a client user fails, either because this class returned a failed login from the `login` or `loginPreviousUser` methods, or because another system connection failed the login.

Syntax

```
public LoginEnumeration loginFailed ( String userId ,  
StringBuffer error )
```

Parameters

- **userId** – The user ID for the user whose login attempt has failed.
- **error** – A `StringBuffer` that will be written to the user's debug log by the Agentry server when this method returns. It should be used to log useful error information within the user's debug log about why the login failed; it will always be logged regardless of the Agentry server's debug settings.

Returns

`Login_Invalid` or `Login_InvalidBlocked`. If the latter is returned, then `loginBlocked(String, StringBuffer)` will be invoked as well.

Usage

It should clean up any user-related resources; it can also return additional information about why the login failed.

loginFailed(User, String, LoginFailureReason, StringBuffer, SessionData) method

This method is called by the Agentry Server when authentication of a client user fails, either because this class threw `PasswordInvalidException` from the `login` or `loginPreviousUser` methods, or because another system connection reported a login failure.

Syntax

```
public void loginFailed ( User user , String userId ,
LoginFailureReason reason , StringBuffer error , SessionData
sessionData ) throws LoginBlockedException
```

Parameters

- **user** – The User object for the user whose login attempt has failed. This can be `null`, if the login was failed by another system connection before the `createUser` and `login` methods of this system connection were called.
- **userId** – The user name of the user that was logging in. This parameter might be useful if `user` is `null`, but you still need to take some sort of action for the user for some reason. If `user` is not `null`, then this parameter will be equal to the user name contained in `user`.
- **reason** – The reason for the login failure.
- **error** – A `StringBuffer` that contains the error message that was returned by the system connection that failed the login. This message will ultimately be logged on the server and displayed on the client. The error message can be changed by modifying the contents of this buffer.
- **sessionData** – The session data. In addition to its usual contents, this data will contain additional information about which system connection failed the login, if the failure reason was not `NoBackEndsAuthenticated`. This information will be available as the values for the SDML keys `failed.backend.id` (the system connection number) and `failed.backend.name` (the system connection name, as configured in `Agentry.ini`).

Exceptions

- **LoginBlockedException class** – if the login failure should be treated as a blocked login instead. This will trigger a subsequent call to `loginBlocked` in this system connection, as well as the equivalent in other system connections.

Usage

It should clean up any user-related resources.

loginPreviousUser(String, String, SessionData) method [deprecated]

Deprecated. Override loginPreviousUser(User, String, SessionData) instead. This method is called when a user has previously logged into Agentry successfully, and is now logging in again due to having been disconnected.

Syntax

```
public LoginEnumeration loginPreviousUser ( String userId ,  
String password ,  SessionData sessionData )
```

Parameters

- **userId** – The user ID from the client application for the current user.
- **password** – The password for the current user, as entered on the client application.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.

Returns

One of the constants from Server.LoginEnumeration.

Usage

This method is only called if both the enableAuthentication and enablePreviousUserAuthentication options are true in the Agentry.ini file for the Java system connection. It should function in the same manner as login(String, String, SessionData).

The default implementation of this method returns Login_Pass.

loginPreviousUser(User, String, SessionData) method

This method is called when a user has previously logged into Agentry successfully, and is now logging in again due to having been disconnected.

Syntax

```
public void loginPreviousUser ( User user ,  String password ,  
SessionData sessionData ) throws LoginException
```

Parameters

- **user** – The User object that identifies the client user. The user name can be read from this object.
- **password** – The password for the current user, as entered on the client application.
- **sessionData** – Provides access to current session data, such as Agentry Server Data Markup Language (SDML) values.

Exceptions

- **LoginException class** – if the login fails for any reason, or if the login succeeds but an exceptional condition exists (such as an expired or soon-to-be expired password).

Usage

This method is only called if both the `enableAuthentication` and `enablePreviousUserAuthentication` options are true in the `Agentry.ini` file for the Java system connection. It should function in the same manner as `login(User, String, SessionData)`.

The default implementation of this method throws `LoginSkippedException`.

setDebugEnabled(boolean) method [deprecated]

Deprecated. This is only here because the Agentry server will call it. Setter method called by the Agentry server to enable/disable debugging.

Syntax

```
final void setDebugEnabled ( boolean debug )
```

Parameters

- **debug** – true if debugging is enabled, false if not.

Usage

Subclasses must never call this.

shutdown() method

This method is called by the Agentry Server when the Java system connection is being shut down.

Syntax

```
public void shutdown ()
```

Usage

It should perform any cleanup that needs to be done.

startup() method

This method is called by the Agentry Server when the Java system connection starts up and creates an instance of this class; it is called immediately after the class is constructed.

Syntax

```
public void startup ()
```

Usage

It exists as a complement to the shutdown method. Current versions of Agentry ignore the value returned by this method, so there's really nothing you can do here that you couldn't just do in the constructor.

ServiceEvent class

This class implements a Java Callback Service Event in Agentry.

Syntax

```
public class ServiceEvent extends
AgentryJavaBackEndManagedObject
```

Members

All members of ServiceEvent, including inherited members. **Variables**

Modifier and Type	Variable	Description
protected Server	<i>_server</i> on page 770	The active Server implementation in the Agentry Java system connection.
protected SessionData	<i>_sessionData</i> on page 770	Session data for this service event session.

Constructors

Modifier and Type	Constructor	Description
public	<i>ServiceEvent(Server, SessionData, CallbackInterface)</i> on page 769	Constructs a new ServiceEvent object.

Methods

Modifier and Type	Method	Description
public final void	<i>dataReceived(Object)</i> on page 769	This method should be called once you have obtained an object's data from the remote system.

Usage

In a Java Callback Service Event, a remote enterprise system initiates a call into this class, then retrieves data for a single Agentry object and passes that data back to the Agentry server. (The Agentry object can be a collection, if you need to handle multiple objects of the same type at once.) The general process works like this:

1. The enterprise system somehow triggers a call to a method in your custom subclass of `ServiceEvent`.
2. Your custom method acquires data from the enterprise system for a single Agentry object and stores it into a custom Java object. This object is implemented the same way as objects returned by `ComplexTable` or `Steplet` - it must contain public fields that can be mapped back to fields in an Agentry object.
3. Your custom method then calls the `dataReceived` method with the new object, which communicates that object back to Agentry.
4. Agentry processes the object, maps it to the Agentry object, and fires the various steps defined for the Service Event in the application for handling the object, which in turn will cause the various methods of `ServiceEventSession` to be invoked.

How the enterprise system actually triggers a call into this class is up to you. Possible methods might include remote RMI calls into the Agentry JVM, receiving JMS messages, or whatever else you can come up with.

ServiceEvent(Server, SessionData, CallbackInterface) constructor

Constructs a new `ServiceEvent` object.

Syntax

```
public ServiceEvent ( Server server , SessionData sessionData ,
CallbackInterface cbi )
```

Parameters

- **server** – The active `Server` object, which will be stored into `_server`.
- **sessionData** – The session data for this service event, which will be stored into `_sessionData`.
- **cbi** – The native callback object provided by Agentry; do not use this object directly, it will be handled by the `dataReceived` method of this class.

Usage

Subclasses should provide a constructor with the same arguments, and pass them untouched to this constructor. This constructor will store its arguments into the corresponding member variables of this class, which can then be accessed directly by subclasses.

dataReceived(Object) method

This method should be called once you have obtained an object's data from the remote system.

Syntax

```
public final void dataReceived ( Object data ) throws
AgentryException
```

Parameters

- **data** – The object to send back to Agentry.

Exceptions

- **AgentryException class** – if an error occurs.

Usage

It will pass that object back to the Agentry server, which will then copy the object's data to the corresponding object in the Agentry application using the mappings configured in the Agentry Editor for this service event.

_server variable

The active Server implementation in the Agentry Java system connection.

Syntax

```
protected Server _server
```

_sessionData variable

Session data for this service event session.

Syntax

```
protected SessionData _sessionData
```

ServiceEventSession class

The ServiceEventSession class encapsulates the processing involved in a service event.

Syntax

```
public class ServiceEventSession extends Session
```

Members

All members of ServiceEventSession, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>ServiceEventSession(String, Server, SessionData)</i> on page 773	Construct a new session.

Methods

Modifier and Type	Method	Description
public void	<i>beginDataAndUpdateSteps()</i> on page 773	This method is called by the server prior to the execution of the "Data State Steps" and "Update Steps" (which are grouped together) for the service event.
public void	<i>beginReadSteps()</i> on page 773	This method is called by the server prior to the execution of the "Read Steps" for the service event.
public void	<i>beginServiceEventError()</i> on page 774	This method is called by the server prior to the execution of the "Error Steps" for the service event.
public void	<i>endDataAndUpdateSteps()</i> on page 774	This method is called after the "Data State Steps" and "Update Steps" (which are grouped together) for the service event have been successfully completed.
public void	<i>endReadSteps()</i> on page 774	This method is called after the "Read Steps" for the service event have been successfully completed.
public void	<i>endServiceEventError()</i> on page 774	This method is called after the "Error Steps" for the service event have been successfully completed.

Inherited members from Session

Modifier and Type	Member	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agentry application.

Modifier and Type	Member	Description
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
protected	<i>Session(String, Server, SessionData, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, SessionData)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

In brief, a service event is the component of the application that defines how data is synchronized between the Agentry Server and another external server application. It is made up of steps (which are implemented by the Steplet class in the Java system connection), each of which perform a specific task related to the synchronization process. These steps are organized into groups within the service event for specific areas of the data synchronization. These areas include the "Read", "Data State", "Update", and "Error Handling" steps.

This class contains methods to perform actions before and after each of these groups of steps. The default implementation of these methods in the AJAPI perform no additional specific actions. A designer can extend this class if special processing is required before or after each of these groups of steps are processed. If this class is extended, the Server class must also be extended and its `createServiceEventSession` method must be overridden to return the designer implemented subclass of the `ServiceEventSession` class.

ServiceEventSession(String, Server, SessionData) constructor

Construct a new session.

Syntax

```
public ServiceEventSession ( String serviceEventName , Server
server , SessionData sessionData )
```

Parameters

- **serviceEventName** – The fetch name, as configured in the Agentry application.
- **server** – The `Server` object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.

Usage

This constructor is called by the `Server.createServiceEventSession` method. Subclasses should implement a constructor with the same signature.

beginDataAndUpdateSteps() method

This method is called by the server prior to the execution of the "Data State Steps" and "Update Steps" (which are grouped together) for the service event.

Syntax

```
public void beginDataAndUpdateSteps ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginReadSteps() method

This method is called by the server prior to the execution of the "Read Steps" for the service event.

Syntax

```
public void beginReadSteps ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

beginServiceEventError() method

This method is called by the server prior to the execution of the "Error Steps" for the service event.

Syntax

```
public void beginServiceEventError ()
```

Usage

Any processing that should take place prior to the execution of these steps should be implemented in this method.

endDataAndUpdateSteps() method

This method is called after the "Data State Steps" and "Update Steps" (which are grouped together) for the service event have been successfully completed.

Syntax

```
public void endDataAndUpdateSteps ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

endReadSteps() method

This method is called after the "Read Steps" for the service event have been successfully completed.

Syntax

```
public void endReadSteps ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

endServiceEventError() method

This method is called after the "Error Steps" for the service event have been successfully completed.

Syntax

```
public void endServiceEventError ()
```

Usage

Any processing that should take place after the execution of these steps should be implemented in this method.

Session class

This is the base class for the various session types in Agentry.

Syntax

```
public class Session extends AgentryJavaBackEndManagedObject
```

Derived classes

- *ComplexTableSession* on page 673
- *DataTableSession* on page 688
- *FetchSession* on page 699
- *PushSession* on page 723
- *PushUserSession* on page 729
- *ServiceEventSession* on page 770
- *TransactionSession* on page 793

Members

All members of Session, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
protected	<i>Session(String, Server, Session-Data, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, Session-Data)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.

Methods

Modifier and Type	Method	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agentry application.

Modifier and Type	Method	Description
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

A session generally refers to the carrying out of a particular action, fetch, push, transaction, etc. in an Agentry application. It contains information about the server and user executing the session, and also holds a SessionData object that provides access back into the Agentry server to retrieve application-specific data.

Session(String, Server, SessionData, User) constructor

Construct a new session, and save each of the arguments so that they can be retrieved later via the getName(), getServer(), getSessionData(), and getUser() methods.

Syntax

```
protected Session ( String name , Server server , SessionData
sessionData , User user )
```

Parameters

- **name** – The session name, as configured in the Agentry application.
- **server** – The Server object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.
- **user** – The client user performing the fetch.

Session(String, Server, SessionData) constructor

Construct a new session, and save each of the arguments so that they can be retrieved later via the getName(), getServer(), and getSessionData() methods.

Syntax

```
protected Session ( String name , Server server , SessionData
sessionData )
```


Parameters

- **name** – The session name, as configured in the Agentry application.
- **server** – The Server object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.

debug(String) method

Write the given message to a debug log, if debugging is enabled.

Syntax

```
public final void debug ( String message )
```

Parameters

- **message** – The message to log

Usage

If this is a user-specific session then the message will be written to the user log, otherwise it will be written to the server log.

This method is simply a convenience method that calls `User.debug()` if the session has a user, or `Server.debug()` if it doesn't.

getName() method

Returns the name of the session, as configured in the Agentry application.

Syntax

```
public String getName ()
```

Returns

The session name.

getServer() method

Returns the Server singleton object that the Java system connection is currently using.

Syntax

```
public Server getServer ()
```

Returns

The server instance

Usage

This will be an instance of the class named in the `serverClass` option of the Java system connection in the `agentry.ini` file.

getSessionData() method

Returns the session data for this session.

Syntax

```
public SessionData getSessionData ()
```

Returns

The session data.

getUser() method

Returns the user for this session, if any.

Syntax

```
public User getUser ()
```

Returns

an object of whatever class is being returned by the active implementation of the `Server` class, or `null` if this session is not user-specific.

sessionAborted() method

This is called if the session is aborted (e.g., by an exception).

Syntax

```
public void sessionAborted ()
```

Steplet class

The `Steplet` class within the AJAPI encapsulates the data synchronization for a step application component.

Syntax

```
public abstract class Steplet extends  
AgentryJavaBackEndManagedObject
```

Members

All members of `Steplet`, including inherited members. **Variables**

Modifier and Type	Variable	Description
protected Session	<code>_session</code> on page 785	Session data, set by the constructor.

Constructors

Modifier and Type	Constructor	Description
public	<i>Steplet(FetchSession)</i> on page 781	Constructs a new Steplet that will be used as part of a fetch.
public	<i>Steplet(PushSession)</i> on page 781	Constructs a new Steplet that will be used as part of a push.
public	<i>Steplet(PushUserSession)</i> on page 781	Constructs a new Steplet that will be used as part of a push.
public	<i>Steplet(TransactionSession)</i> on page 781	Constructs a new Steplet that will be used as part of a transaction.
public	<i>Steplet(ServiceEventSession)</i> on page 782	Constructs a new Steplet that will be used as part of a service event.

Methods

Modifier and Type	Method	Description
public abstract boolean	<i>doSteplet()</i> on page 782	Perform the necessary actions of this steplet.
public String	<i>getNotificationText()</i> on page 783	This method is called for Transaction Error-Handling Steplets.
public String	<i>getNotificationTitle()</i> on page 783	This method is called for Transaction Error-Handling Steplets.
public String	<i>getOkButtonLabel()</i> on page 783	This method is called for Transaction Error-Handling Steplets.
public Object	<i>getReturnData()</i> on page 784	The Agentry server will call this method to obtain the data produced by the doSteplet method.
public Session	<i>getSession()</i> on page 784	Returns the Session object for this steplet.

Modifier and Type	Method	Description
public String	<i>notificationText()</i> on page 785	Deprecated. Override getNotificationText() instead. The default implementation of this method will call that method. This method may be made final or removed in a future release. Use getNotificationText() instead.
public String	<i>notificationTitle()</i> on page 785	Deprecated. Override getNotificationTitle() instead. The default implementation of this method will call that method. This method may be made final or removed in a future release. Use getNotificationTitle() instead.
public String	<i>okButtonLabel()</i> on page 785	Deprecated. Override getOkButtonLabel() instead. The default implementation of this method will call that method. This method may be made final or removed in a future release. Use getOkButtonLabel() instead.

Usage

The Agency Editor provides a template subclass of this class for each step in the application that uses a Java system connection; these subclasses must override the methods in this class to implement their behavior. A basic steplet needs to implement `doSteplet()` and `getReturnData()`. Steplets that will be used in Agency transaction error-handling steps can also override the `notificationTitle()`, `notificationText()`, and `okButtonLabel()` methods to control the contents of a failed transaction's error notification windows.

A typical steplet implementation will implement `doSteplet()` to retrieve a set of data from a remote system, package that data into an object or array of objects, and save the data in a member field. The implementation of the `getReturnData()` method will then return that data as either a single object or an array of objects. These objects will in turn contain publicly-visible fields that are mapped within the Agency application (via the Agency Editor) to the fields of corresponding Agency objects. The Agency server will read data directly from the fields of these objects; the server will not use getter/setter methods to read them.

A steplet can throw `StepletStopException` to stop processing of itself but allow processing of subsequence steplets in a session to continue, or it can throw `StepletAbortException` to stop processing of itself and any subsequent steplets in the session. It can also throw a `BusinessLogicException` exception to report an error message to the client's transmit window. Steplets that are used in transaction steps can also throw `RetryTransactionException`, `RetryTransactionWithChangeException`, or `FatalTransactionException` to abort the transaction in various ways.

Steplet(FetchSession) constructor

Constructs a new Steplet that will be used as part of a fetch.

Syntax

```
public    Steplet ( FetchSession session )
```

Parameters

- **session** – Fetch session information, stored into the `_session` member variable.

Steplet(PushSession) constructor

Constructs a new Steplet that will be used as part of a push.

Syntax

```
public    Steplet ( PushSession session )
```

Parameters

- **session** – Push session information, stored into the `_session` member variable

Steplet(PushUserSession) constructor

Constructs a new Steplet that will be used as part of a push.

Syntax

```
public    Steplet ( PushUserSession session )
```

Parameters

- **session** – Push user session information, stored into the `_session` member variable

Steplet(TransactionSession) constructor

Constructs a new Steplet that will be used as part of a transaction.

Syntax

```
public    Steplet ( TransactionSession session )
```

Parameters

- **session** – Transaction session information, stored into the `_session` member variable

Steplet(ServiceEventSession) constructor

Constructs a new Steplet that will be used as part of a service event.

Syntax

```
public Steplet ( ServiceEventSession session )
```

Parameters

- **session** – Service event session information, stored into the `_session` member variable

doSteplet() method

Perform the necessary actions of this steplet.

Syntax

```
public abstract boolean doSteplet () throws AgentryException
```

Returns

For fetch, push, transaction update step, and service event read, data, and update steplets: `true` if the steplet produced data that the Agentry server should read from the `_returnData` field, or `false` if no data was produced. For transaction data state and error handling steplets, and service event error handling steplets: the meaning of the return value is configured in the Agentry Editor.

Exceptions

- **AgentryException class** – if an error occurs.

Usage

A steplet can obtain various parameters from Agentry via the session information stored in the `_session` member variable. Steplet objects that retrieve data that will be read by the Agentry server should store that data in a public field named `_returnData`, which the Agentry Server will read via reflection if this method returns `true`.

A steplet can throw `StepletStopException` to stop processing of itself but allow processing of subsequent steplets in a session to continue, or it can throw `StepletAbortException` to stop processing of itself and any subsequent steplets in the session. It can also throw a `BusinessLogicException` exception to report an error message to the client's transmit window. Steplets used in transactions can also throw any of the transaction-related exceptions (`FatalTransactionException`, `RetryTransactionWithChangeException`, and `RetryTransactionException`).

getNotificationText() method

This method is called for Transaction Error-Handling Steplets.

Syntax

```
public String getNotificationText ()
```

Returns

the notification window text, or an empty string to use the text from the original exception that caused this error-handling steplet to be invoked.

Usage

It is intended to provide the error-handling steplet with a chance to override the notification window text from the original transaction failure exception. This method overrides the window text for the notification window; if it returns an empty string, then the text from the original exception will be used.

getNotificationTitle() method

This method is called for Transaction Error-Handling Steplets.

Syntax

```
public String getNotificationTitle ()
```

Returns

the notification window title, or an empty string to use the title from the original exception that caused this error-handling steplet to be invoked.

Usage

It is intended to provide the error-handling steplet with a chance to override the notification window text from the original transaction failure exception. This method overrides the window title for the notification window; if it returns an empty string, then the title from the original exception will be used.

getOkButtonLabel() method

This method is called for Transaction Error-Handling Steplets.

Syntax

```
public String getOkButtonLabel ()
```

Returns

the button label, or an empty string to use the label from the original exception that caused this error-handling steplet to be invoked.

Usage

It is intended to provide the error-handling steplet with a chance to override the notification window text from the original transaction failure exception. This method overrides the button label for the notification window; if it returns an empty string, then the button label from the original exception will be used.

getReturnData() method

The Agentry server will call this method to obtain the data produced by the doSteplet method.

Syntax

```
public Object getReturnData ()
```

Returns

A data object or an array of data objects

Usage

It will only be called if doSteplet returned `true` and the steplet is being used by a data or fetch step in the Agentry application.

The objects returned by this method will be mapped to Agentry objects according to the field mappings defined in the Agentry application. This method should return either a single such object, or an array of them.

The implementation of this method typically should be very simple, in that it should just return some data that was built up in doSteplet. All of the "heavy lifting" should be done in that method.

Migration tip: If you are migrating an application from AJAPI version 4 (which returned steplet data via a special field named `_returnData`), just override this method to return the value of your steplet's `_returnData` field. If you do not override this method, the Agentry server will attempt to read that field anyways (for backwards compatibility), but this behavior is considered deprecated and should not be relied on in future versions of Agentry. It's also faster to have this method return the field, since otherwise the Agentry server will need to use reflection to read it.

getSession() method

Returns the Session object for this steplet.

Syntax

```
public Session getSession ()
```

Returns

the session.

notificationText() method [deprecated]

Deprecated. Override getNotificationText() instead. The default implementation of this method will call that method. This method may be made final or removed in a future release. Use getNotificationText() instead.

Syntax

```
public String notificationText ()
```

Returns

the text

notificationTitle() method [deprecated]

Deprecated. Override getNotificationTitle() instead. The default implementation of this method will call that method. This method may be made final or removed in a future release. Use getNotificationTitle() instead.

Syntax

```
public String notificationTitle ()
```

Returns

the title

okButtonLabel() method [deprecated]

Deprecated. Override getOkButtonLabel() instead. The default implementation of this method will call that method. This method may be made final or removed in a future release. Use getOkButtonLabel() instead.

Syntax

```
public String okButtonLabel ()
```

Returns

the label

_session variable

Session data, set by the constructor.

Syntax

```
protected Session _session
```

StepletAbortException class

This exception can be thrown from a Steplet object to abort processing of that steplet and any subsequent steplets in the session.

Syntax

```
public class StepletAbortException extends AgentryException
```

Members

All members of StepletAbortException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>StepletAbortException(String)</i> on page 787	Constructs a new StepletAbortException object.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The Agentry Server will roll back any transactional work that has been done so far for the step, and will not execute any subsequent steps. It will trigger the `sessionAborted` method of the appropriate session object.

StepletAbortException(String) constructor

Constructs a new StepletAbortException object.

Syntax

```
public StepletAbortException ( String message )
```

Parameters

- **message** – The error message to log to the server's event log.

StepletStopException class

This exception can be thrown from a Steplet object to stop the processing of the currently executing step and signal to the Agentry Server that any transaction-based work completed thus far should be committed.

Syntax

```
public class StepletStopException extends AgentryException
```

Members

All members of StepletStopException, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>StepletStopException(String)</i> on page 788	Constructs a new StepletStopException object.

Inherited members from AgentryException

Modifier and Type	Member	Description
public	<i>AgentryException(String)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, Throwable)</i> on page 669	Constructs a new AgentryException object.
public	<i>AgentryException(String, String, String, Throwable)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.
public	<i>AgentryException(String, String, String)</i> on page 670	Constructs a new AgentryException object, for methods that support throwing exceptions that may appear in the client.

Modifier and Type	Member	Description
public final String	<i>getNotificationText()</i> on page 671	Returns the notification window text.
public final String	<i>getNotificationTitle()</i> on page 671	Returns the notification window title.
public final String	<i>getOkButtonLabel()</i> on page 671	Returns the notification window button label.

Usage

The Agentry Server will then continue processing any remaining steps.

StepletStopException(String) constructor

Constructs a new *StepletStopException* object.

Syntax

```
public StepletStopException ( String message )
```

Parameters

- **message** – The error message to log

SycloCalendar class

This class extends *GregorianCalendar* with methods for detecting Agentry's "invalid date" value.

Syntax

```
public class SycloCalendar extends GregorianCalendar
```

Members

All members of *SycloCalendar*, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>SycloCalendar(GregorianCalendar)</i> on page 789	Constructs a new <i>SycloCalendar</i> object using the data from an existing <i>GregorianCalendar</i> object and the default locale.
public	<i>SycloCalendar(GregorianCalendar, Locale)</i> on page 790	Constructs a new <i>SycloCalendar</i> object using the data from an existing <i>GregorianCalendar</i> object and the given locale.

Modifier and Type	Constructor	Description
public	<i>SycloCalendar()</i> on page 790	Constructs a new SycloCalendar object.
public	<i>SycloCalendar(int, int, int, int, int, int)</i> on page 790	Constructs a new SycloCalendar object.
public	<i>SycloCalendar(int, int, int, int, int)</i> on page 791	Constructs a new SycloCalendar object.
public	<i>SycloCalendar(int, int, int)</i> on page 791	Constructs a new SycloCalendar object.
public	<i>SycloCalendar(Locale)</i> on page 791	Constructs a new SycloCalendar object.
public	<i>SycloCalendar(TimeZone, Locale)</i> on page 791	Constructs a new SycloCalendar object.
public	<i>SycloCalendar(TimeZone)</i> on page 792	Constructs a new SycloCalendar object.

Methods

Modifier and Type	Method	Description
public static GregorianCalendar	<i>getInvalidTimeAndDate()</i> on page 792	Returns the Agentry invalid date value.
public boolean	<i>isInvalidTimeAndDate()</i> on page 792	Checks to see if this object contains Agentry's "invalid date" value.
public static boolean	<i>isInvalidTimeAndDate(GregorianCalendar)</i> on page 792	Returns whether the given date matches Agentry's "invalid date" value.

SycloCalendar(GregorianCalendar) constructor

Constructs a new SycloCalendar object using the data from an existing GregorianCalendar object and the default locale.

Syntax

```
public    SycloCalendar ( GregorianCalendar cal )
```

Parameters

- **cal** – The existing object

Usage

If you need a non-default locale, use `SycloCalendar(GregorianCalendar, Locale)`; this constructor cannot read the locale from the given calendar object because `GregorianCalendar` does not provide a method for doing that.

SycloCalendar(GregorianCalendar, Locale) constructor

Constructs a new `SycloCalendar` object using the data from an existing `GregorianCalendar` object and the given locale.

Syntax

```
public SycloCalendar ( GregorianCalendar cal , Locale locale )
```

Parameters

- **cal** – The existing object
- **locale** – The locale

SycloCalendar() constructor

Constructs a new `SycloCalendar` object.

Syntax

```
public SycloCalendar ()
```

SycloCalendar(int, int, int, int, int, int) constructor

Constructs a new `SycloCalendar` object.

Syntax

```
public SycloCalendar ( int year , int month , int dayOfMonth , int  
hourOfDay , int minute , int second )
```

Parameters

- **year** – The year
- **month** – The month
- **dayOfMonth** – The day of the month
- **hourOfDay** – The hours
- **minute** – The minutes
- **second** – The seconds

SycloCalendar(int, int, int, int, int) constructor

Constructs a new SycloCalendar object.

Syntax

```
public SycloCalendar ( int year , int month , int dayOfMonth , int
hourOfDay , int minute )
```

Parameters

- **year** – The year
- **month** – The month
- **dayOfMonth** – The day of the month
- **hourOfDay** – The hours
- **minute** – The minutes

SycloCalendar(int, int, int) constructor

Constructs a new SycloCalendar object.

Syntax

```
public SycloCalendar ( int year , int month , int dayOfMonth )
```

Parameters

- **year** – The year
- **month** – The month
- **dayOfMonth** – The day of the month

SycloCalendar(Locale) constructor

Constructs a new SycloCalendar object.

Syntax

```
public SycloCalendar ( Locale locale )
```

Parameters

- **locale** – The locale

SycloCalendar(TimeZone, Locale) constructor

Constructs a new SycloCalendar object.

Syntax

```
public SycloCalendar ( TimeZone zone , Locale locale )
```

Parameters

- **zone** – The time zone
- **locale** – The locale

SycloCalendar(TimeZone) constructor

Constructs a new SycloCalendar object.

Syntax

```
public SycloCalendar ( TimeZone zone )
```

Parameters

- **zone** – The time zone

getInvalidTimeAndDate() method

Returns the Agentry invalid date value.

Syntax

```
public static GregorianCalendar getInvalidTimeAndDate ()
```

Returns

the invalid date.

isInvalidTimeAndDate() method

Checks to see if this object contains Agentry's "invalid date" value.

Syntax

```
public boolean isInvalidTimeAndDate ()
```

Returns

true if the date is invalid, or false if not.

isInvalidTimeAndDate(GregorianCalendar) method

Returns whether the given date matches Agentry's "invalid date" value.

Syntax

```
public static boolean isInvalidTimeAndDate ( GregorianCalendar  
testDate )
```

Parameters

- **testDate** – the date to check

Returns

true if the date is invalid, or false if not.

TransactionSession class

The TransactionSession class encapsulates the processing related to transactions.

Syntax

```
public class TransactionSession extends Session
```

Members

All members of TransactionSession, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>TransactionSession(String, Server, SessionData, User)</i> on page 794	Construct a new session.

Methods

Modifier and Type	Method	Description
public void	<i>beginTransaction()</i> on page 795	This method is called by the Agentry Server prior to executing the steps for the transaction.
public void	<i>endTransaction()</i> on page 795	This method is called after the steps for the transaction have been successfully processed.

Inherited members from Session

Modifier and Type	Member	Description
public final void	<i>debug(String)</i> on page 777	Write the given message to a debug log, if debugging is enabled.
public String	<i>getName()</i> on page 777	Returns the name of the session, as configured in the Agentry application.
public Server	<i>getServer()</i> on page 777	Returns the Server singleton object that the Java system connection is currently using.

Modifier and Type	Member	Description
public SessionData	<i>getSessionData()</i> on page 778	Returns the session data for this session.
public User	<i>getUser()</i> on page 778	Returns the user for this session, if any.
protected	<i>Session(String, Server, SessionData, User)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , <i>getSessionData()</i> , and <i>getUser()</i> methods.
protected	<i>Session(String, Server, SessionData)</i> on page 776	Construct a new session, and save each of the arguments so that they can be retrieved later via the <i>getName()</i> , <i>getServer()</i> , and <i>getSessionData()</i> methods.
public void	<i>sessionAborted()</i> on page 778	This is called if the session is aborted (e.g., by an exception).

Usage

In brief, transactions are the application component that defines data modifications made by the user on the client application. These changes are then transmitted to the Agentry Server for processing. The processing for a transaction is defined by its steps. This class contains methods that allow the implementation of processing that may be required before or after the transaction steps are processed.

A designer can extend this class if special processing is required before or after a transaction is processed. If this class is extended, the *Server* class must also be extended and its *createTransactionSession* method must be overridden to return the designer implemented subclass of the *TransactionSession* class.

TransactionSession(String, Server, SessionData, User) constructor

Construct a new session.

Syntax

```
public TransactionSession ( String transactionName , Server server ,
SessionData sessionData , User user )
```

Parameters

- **transactionName** – The fetch name, as configured in the Agentry application.

- **server** – The Server object that the Java System connection was configured to use.
- **sessionData** – Session data for this fetch.
- **user** – The client user performing the fetch.

Usage

This constructor is called by the `Server.createTransactionSession` method. Subclasses should implement a constructor with the same signature.

beginTransaction() method

This method is called by the Agentry Server prior to executing the steps for the transaction.

Syntax

```
public void beginTransaction ()
```

Usage

Any processing that should take place at this point should be implemented in this method.

endTransaction() method

This method is called after the steps for the transaction have been successfully processed.

Syntax

```
public void endTransaction ()
```

Usage

Any processing that should take place at this point should be implemented in this method.

User class

This class represents an Agentry client user.

Syntax

```
public class User extends AgentryJavaBackEndManagedObject
```

Members

All members of User, including inherited members. **Variables**

Modifier and Type	Variable	Description
protected String	<i>_name</i> on page 805	User name.

Constructors

Modifier and Type	Constructor	Description
public	<i>User(String)</i> on page 799	This is the constructor method for objects of type User.

Methods

Modifier and Type	Method	Description
public final GregorianCalendar	<i>backendTimeAndDate()</i> on page 799	Deprecated. This method has been renamed to <code>getSystemConnectionTime()</code> . This method has been renamed to <code>getSystemConnectionTime()</code> .
public void	<i>beginChangePassword()</i> on page 800	This is the first method called when a user is attempting to change their password.
public ChangePasswordResult	<i>changePassword(String, String)</i> on page 800	This method is called when a user is attempting to change their password.
public void	<i>changePasswordFailed(StringBuffer)</i> on page 801	This method is called when the <code>changePassword(String, String)</code> method returns any value other than <code>ChangePassword_Success</code> or <code>ChangePassword_NotHandled</code> .
public void	<i>changePasswordSessionAborted()</i> on page 801	This method is called if the password change operation is aborted for any reason.
public final void	<i>debug(String)</i> on page 801	Writes a debugging message to the user's log file on the Agentry server.
public void	<i>endChangePassword()</i> on page 802	This method is called when the user's password has been successfully changed.
public String	<i>getName()</i> on page 802	Returns the user's name.
public GregorianCalendar	<i>getSystemConnectionTime()</i> on page 802	This is called by the Agentry server to find out what time the Java system connection thinks it is right now.
public final void	<i>getTimeZone(StringBuffer)</i> on page 803	Deprecated. This method has been moved to <code>Server#getTimeZone()</code> . This method is no longer supported.

Modifier and Type	Method	Description
public void	<i>loggedIn()</i> on page 803	This method is called after a user has been successfully logged in.
public void	<i>loggedOut()</i> on page 803	This method is called after the transmission has been completed and after the user is logged out of the system.
public void	<i>reLoggedIn()</i> on page 804	This method is called when a user logs into the Agentry Server and the server still has a previous login session for that user.
public void	<i>revalidate(String)</i> on page 804	This method authenticates a client user against the Java System Connection.
public void	<i>timedOut()</i> on page 805	This method is called in the event a user session times out.
public void	<i>update(GregorianCalendar)</i> on page 805	This method is called periodically (once every second or so) by the Agentry Server.

Usage

This class is created by the `Server#createUser(String)` factory method. It contains methods for notifying the application of successful login, logout, and other events. It also contains methods which can be overridden to allow Agentry to change a user's password on a remote system.

Applications can extend this class to implement their own behavior; however, if you do so you must also override the `Server#createUser(String)` factory method to return the new subclass. You must then change the `serverClass` setting in the `Agentry.ini` configuration file to tell the Agentry Java system connection to use your new `Server` subclass.

User.ChangePasswordResult enum

Outcomes for password changes, returned by the `User#changePassword(String, String)` method and its ilk.

Members

All members of `ChangePasswordResult`, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>ChangePassword_Blocked</i> on page 798	The user has been blocked; the password has not been changed.
public	<i>ChangePassword_Failure</i> on page 798	The attempt to change the user's password has failed.
public	<i>ChangePassword_NotHandled</i> on page 799	The user's password change is not handled by this System Connection.
public	<i>ChangePassword_Success</i> on page 799	The user's password has been successfully changed.

Methods

Modifier and Type	Method	Description
public int	<i>getValue()</i> on page 798	Called by Agentry to retrieve the integer value for the enum.

getValue() method

Called by Agentry to retrieve the integer value for the enum.

Syntax

```
public int getValue ()
```

Returns

the value

ChangePassword_Blocked variable

The user has been blocked; the password has not been changed.

Syntax

```
public    ChangePassword_Blocked
```

ChangePassword_Failure variable

The attempt to change the user's password has failed.

Syntax

```
public    ChangePassword_Failure
```

ChangePassword_NotHandled variable

The user's password change is not handled by this System Connection.

Syntax

```
public    ChangePassword_NotHandled
```

Usage

(Normally used in environments where multiple back-end systems are in place).

ChangePassword_Success variable

The user's password has been successfully changed.

Syntax

```
public    ChangePassword_Success
```

User(String) constructor

This is the constructor method for objects of type User.

Syntax

```
public    User ( String name )
```

Parameters

- **name** – The user ID as entered on the client application. This value is accessible by calling the name() member method.

Usage

The API class Server will call this method prior to attempting to log the user into the system and/or perform the data synchronization with the Java interface.

backendTimeAndDate() method [deprecated]

Deprecated. This method has been renamed to getSystemConnectionTime(). This method has been renamed to getSystemConnectionTime().

Syntax

```
public final GregorianCalendar backendTimeAndDate () throws  
AgentryException
```

Returns

nothing, throws UnsupportedOperationException.

Exceptions

- **AgentryException class** – not thrown

- **UnsupportedOperationException** – to indicate that the method is no longer supported and should not be called.

Usage

Override that method instead.

beginChangePassword() method

This is the first method called when a user is attempting to change their password.

Syntax

```
public void beginChangePassword ()
```

Usage

It is intended to allow the designer to implement any functionality or processing that may be necessary prior to changing the user's password. Typically this method will start a password change transaction with a remote server. The actual password change functionality should not be a part of this processing.

This method will only be called if authentication is enabled for the Agentry Java system connection.

changePassword(String, String) method

This method is called when a user is attempting to change their password.

Syntax

```
public ChangePasswordResult changePassword ( String oldPassword ,  
String newPassword )
```

Parameters

- **oldPassword** – The current password for the user.
- **newPassword** – The value that user's password should be changed to.

Returns

one of the constants from User.ChangePasswordResult

Usage

This method can be overridden to implement password changing against a remote system. The return value indicates the success or failure, and the reason for the failure, of the change password attempt.

This method will only be called if authentication is enabled for the Agentry Java system connection.

changePasswordFailed(StringBuffer) method

This method is called when the `changePassword(String, String)` method returns any value other than `ChangePassword_Success` or `ChangePassword_NotHandled`.

Syntax

```
public void changePasswordFailed ( StringBuffer errorString )
```

Parameters

- **errorString** – This is a string value that can be set to a text value. This value will be written out to the user's debug log. It should indicate the reason why the password change failed.

Usage

This method can be overridden to perform any necessary processing in the event a password change fails.

This method will only be called if authentication is enabled for the Agentry Java system connection.

changePasswordSessionAborted() method

This method is called if the password change operation is aborted for any reason.

Syntax

```
public void changePasswordSessionAborted ()
```

Usage

It should roll back anything that was started by the `beginChangePassword()` method.

This method will only be called if authentication is enabled for the Agentry Java system connection.

debug(String) method

Writes a debugging message to the user's log file on the Agentry server.

Syntax

```
public final void debug ( String userMessage )
```

Parameters

- **userMessage** – The message to log

Usage

This will only work if per-user logging is turned on in `AgentryLogging.ini`.

This method is a convenience method that calls into the Java Logging API to do the actual logging, and assumes that Agentry's default Java Logging configuration is in place (which will

route log messages back to the Agentry server). It will log to a logger named "com.syclo.agentry.Server", at the `FINE` level (which translates to log detail level 3 in Agentry).

When invoked outside of Agentry (e.g. in unit tests), this will log to the console, as that is Java's normal default logging configuration.

endChangePassword() method

This method is called when the user's password has been successfully changed.

Syntax

```
public void endChangePassword ()
```

Usage

Designers can override this method to perform any additional processing that may be required after the user's password has been modified. This method will still be called even if `changePasswordFailed()` is called before it.

This method will only be called if authentication is enabled for the Agentry Java system connection.

getName() method

Returns the user's name.

Syntax

```
public String getName ()
```

Returns

the user name

getSystemConnectionTime() method

This is called by the Agentry server to find out what time the Java system connection thinks it is right now.

Syntax

```
public GregorianCalendar getSystemConnectionTime () throws  
AgentryException
```

Returns

The current date and time from the remote system's perspective.

Exceptions

- **AgentryException class** – if the current date and time cannot be determined.

Usage

Implementations that are communicating with remote servers should override this method to return the time on the remote server, if possible, in order to help Agentry calculate the difference between the time on the client and the time on the remote system. If you override the `Server#getTimeZone` method or set the Java system connection's time zone explicitly in `Agentry.ini`, then you should override this method to return the current time in the same time zone that that method is reporting.

getTimeZone(StringBuffer) method [deprecated]

Deprecated. This method has been moved to `Server#getTimeZone()`. This method is no longer supported.

Syntax

```
public final void getTimeZone ( StringBuffer tz )
```

Parameters

- **tz** – not used

Exceptions

- **UnsupportedOperationException** – to indicate that the method is no longer supported and should not be called.

Usage

Override `Server#getTimeZone()` instead, as the time zone affects the entire Java system connection and not a specific user.

loggedIn() method

This method is called after a user has been successfully logged in.

Syntax

```
public void loggedIn ()
```

Usage

The default version of this method writes a message to the user's debug log, if debugging is active, and returns. This method can be overridden if additional processing is required after a user has successfully logged in and before the transmission is processed.

loggedOut() method

This method is called after the transmission has been completed and after the user is logged out of the system.

Syntax

```
public void loggedOut ()
```

Usage

This is the last method called prior to the destruction of a User object. The default version of this method performs only the single task of logging a message to the user's debug log, if debugging is enabled. This method can be overridden to disconnect a user from a remote system or perform other cleanup.

reLoggedIn() method

This method is called when a user logs into the Agentry Server and the server still has a previous login session for that user.

Syntax

```
public void reLoggedIn ()
```

Usage

This can occur if a user loses network connectivity in the middle of a transmission. This method can be overridden to perform any special processing that may be needed in this situation.

revalidate(String) method

This method authenticates a client user against the Java System Connection.

Syntax

```
public void revalidate ( String password ) throws LoginException
```

Parameters

- **password** – The password for the current user, as entered on the client application.

Exceptions

- **LoginException class** – if the login fails for any reason, or if the login succeeds but an exceptional condition exists (such as an expired or soon-to-be expired password).

Usage

This method is called when a user reconnects to the Agentry Server from a client application and the `enableAuthentication` option is set to `true` in the Java section of the `Agentry.ini` file. Override this method to implement logic to perform full validation of the user against a remote system.

This method should return normally if the authentication of the user succeeds. If authentication fails for any reason, the appropriate `LoginException` subclass should be thrown. An exception should also be thrown for other conditions such as expired or soon-to-be-expiring passwords. By default, this method throws `PasswordInvalidException`, which means that the password is not valid for the user.

timedOut() method

This method is called in the event a user session times out.

Syntax

```
public void timedOut ()
```

Usage

A time out occurs when a transmission is idle for longer than the configured maximum time limit. This method can be overridden to perform any tasks that may be needed in this event.

In the event of a timeout, the user will also be logged out of the Agentry Server, which will trigger calls to the `loggedOut()` method as well.

update(GregorianCalendar) method

This method is called periodically (once every second or so) by the Agentry Server.

Syntax

```
public void update ( GregorianCalendar update )
```

Parameters

- **update** – The time when the timer tick occurred, should be "now" more or less.

Usage

This method can be overridden if some sort of routine maintenance is required, such as sending a keep-alive to a remote server.

_name variable

User name.

Syntax

```
protected String _name
```

Usage

Can be obtained via the `getName()` method.

SessionData interface

The `SessionData` interface is used throughout the AJAPI classes.

Syntax

```
public interface SessionData
```

Members

All members of `SessionData`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public SessionData	<i>sessionData(String)</i> on page 807	Returns a new SessionData object, configured with the given SDML name prefix, that has access to the same session data as this object.

Methods

Modifier and Type	Method	Description
public String	<i>eval(String)</i> on page 808	Evaluates the given string as an Agentry SDML expression and returns the result.
public boolean	<i>getBoolean(String)</i> on page 808	Returns the specified property as a Boolean value.
public byte[]	<i>getBytes(String)</i> on page 808	Returns the specified property as an array of bytes.
public double	<i>getDouble(String)</i> on page 809	Returns the specified property as a double-precision floating point decimal value.
public float	<i>getFloat(String)</i> on page 809	Returns the specified property as a floating point decimal value.
public int	<i>getInteger(String)</i> on page 809	Returns the specified property as an integer value.
public long	<i>getLong(String)</i> on page 809	Returns the specified property as a long integer value.
public String	<i>getString(String)</i> on page 810	Returns the specified property as a string.
public GregorianCalendar	<i>getTimeAndDate(String)</i> on page 810	Returns the specified property as a date contained in a GregorianCalendar object.
public String	<i>getTimeAndDate(String, String)</i> on page 810	Returns the specified property as a date string using the given format, specified as an Agentry date format string (not a Java date format string!).

Usage

It can also be used within the designer-implemented extensions of those classes. This class encapsulates the Server Data Markup Language (SDML) functionality available in Agentry. Through a `SessionData` object, the designer can access the data specific to the current session.

This class contains several getter methods to return specified pieces of data. Each of these methods returns the data as a different data type, such as a string or integer. All of these methods take a single argument of type `String` that specifies the data to return. An example, to retrieve the data for a string property within a transaction named "Description", the following line of code would be used:

```
String desc = _sessionData.getString("Description");
```

When calling these methods, it is important to make sure that the appropriate method is called for the desired data type. No errors will be reported for mismatched data types. For example, if a property within a transaction is of type integer, and the value is retrieved by calling `getString`, the value will be returned as a `String` value. In some cases this may be desirable behavior, but in others it can cause undesirable results.

The primary implementation of this interface is a private class that can only be instantiated by the Agentry Server, since it calls back into the running Agentry server to obtain its data. For unit-testing purposes, you can also create a new subclass or mock implementation of this interface; one example of such a testing version is `TestSessionData`.

sessionData(String) constructor

Returns a new `SessionData` object, configured with the given SDML name prefix, that has access to the same session data as this object.

Syntax

```
public SessionData sessionData (String sessionData )
```

Parameters

- **sessionData** – The new SDML name prefix.

Returns

A new `SessionData` object that prefixes all property references with the prefix given by `sessionData`.

Usage

This can be used to create a session data object that is effectively restricted to only accessing data that starts with the given prefix.

eval(String) method

Evaluates the given string as an Agentry SDML expression and returns the result.

Syntax

```
public String eval ( String sdmlString )
```

Parameters

- **sdmlString** – The SDML string to evaluate (without the enclosing angle brackets).

Returns

The result as a string.

getBoolean(String) method

Returns the specified property as a Boolean value.

Syntax

```
public boolean getBoolean ( String property )
```

Parameters

- **property** – The property name

Returns

The property value as a Boolean.

getBytes(String) method

Returns the specified property as an array of bytes.

Syntax

```
public byte[] getBytes ( String property ) throws AgentryException
```

Parameters

- **property** – The property name

Returns

The property value as an array of bytes

Exceptions

- **AgentryException class** – if the property cannot be interpreted as bytes

getDouble(String) method

Returns the specified property as a double-precision floating point decimal value.

Syntax

```
public double getDouble ( String property )
```

Parameters

- **property** – The property name

Returns

The property value as a double.

getFloat(String) method

Returns the specified property as a floating point decimal value.

Syntax

```
public float getFloat ( String property )
```

Parameters

- **property** – The property name

Returns

The property value as a float.

getInteger(String) method

Returns the specified property as an integer value.

Syntax

```
public int getInteger ( String property )
```

Parameters

- **property** – The property name

Returns

The property value as an integer.

getLong(String) method

Returns the specified property as a long integer value.

Syntax

```
public long getLong ( String property )
```

Parameters

- **property** – The property name

Returns

The property value as a long integer.

getString(String) method

Returns the specified property as a string.

Syntax

```
public String getString ( String property )
```

Parameters

- **property** – The property name

Returns

The property value as a string.

getTimeAndDate(String) method

Returns the specified property as a date contained in a `GregorianCalendar` object.

Syntax

```
public GregorianCalendar getTimeAndDate ( String property ) throws  
AgentryException
```

Parameters

- **property** – The property name

Returns

The property value as a date. If the property was empty, then this will return the current date.

Exceptions

- **AgentryException class** – if the value cannot be parsed as a date.

getTimeAndDate(String, String) method

Returns the specified property as a date string using the given format, specified as an Agentry date format string (not a Java date format string!).

Syntax

```
public String getTimeAndDate ( String property , String format )
```

Parameters

- **property** – The property name
- **format** – The date format to use. Note that this is an Agentry Server date format, not the format used by e.g., SimpleDateFormat!

Returns

The property value as a formatted string.

Usage

If you prefer to use Java date format strings, then just call `getTimeAndDate(String)` to get a `Calendar` object and feed it to a `SimpleDateFormat` object.

Agentry SAP Framework

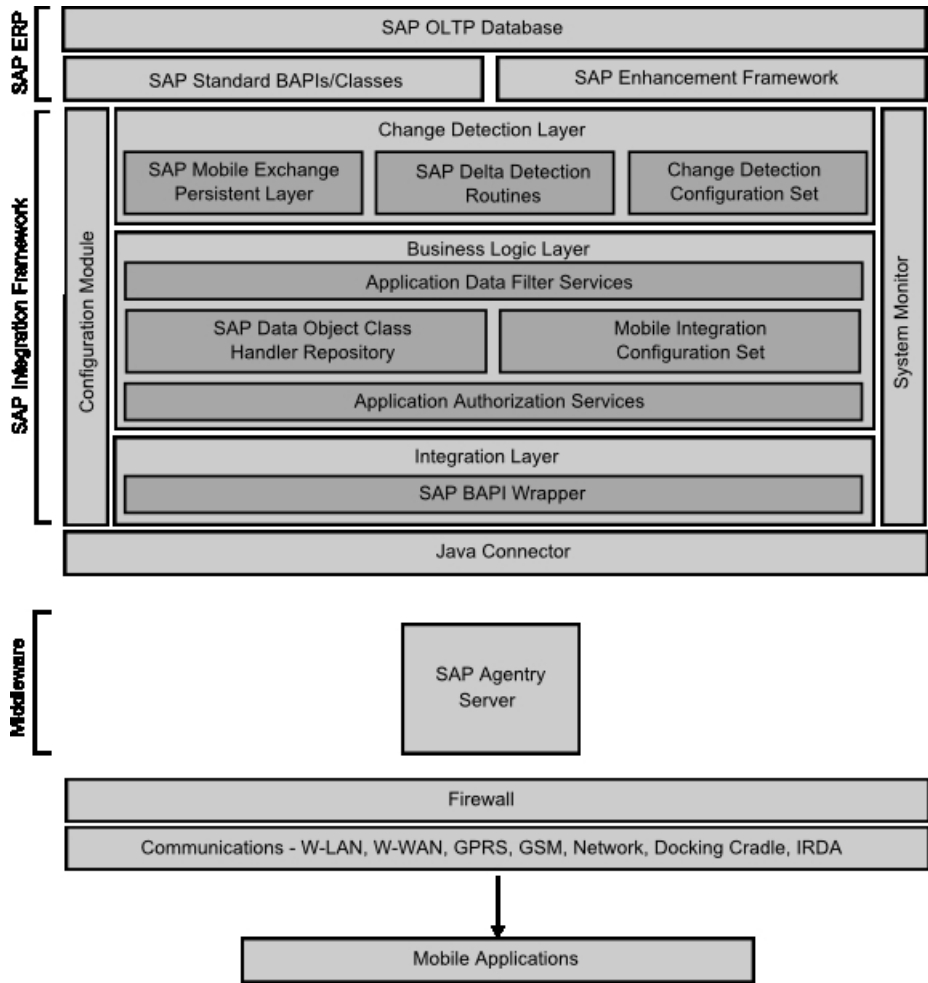
Agentry SAP Framework

The Agentry SAP Framework provides an efficient way to build mobile solutions for SAP® using SAP Mobile Platform. The framework was developed to address the following business needs:

- Establish a common mobile integration service layer and architecture for the mobile adaptation of enterprise business processes and business object data for enterprise mobile applications.
- Define a consistent integration pattern for mobile data object modeling, change detections, data distribution, delta sync calculation, and data pushes.
- Provide a framework for mobile application logging, tracing, administration, and monitoring.
- Uses a combined coding and configuration approach that enables a comprehensive and flexible application paradigm. Standard objects are configurable and extensible. Partners and customers are able to develop their own integration services using the same integration pattern.
- Uses the latest technology supported by SAP, to protect the return on investment.

The framework consists of several components shown in the following diagram.

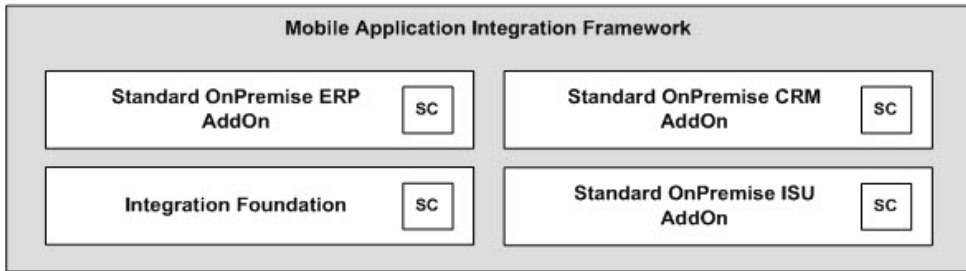
Figure 1: Agentry SAP Framework System Components



Software Component Layering

The Agentry SAP Framework uses a layered software component approach; dependency between each software component layer is by design and encouraged in order to provide maximum reusability. There are two categories of software components within the Agentry SAP Framework: the Foundation Add-On Component, and the Application Add-On Components, as shown in the following diagram.

Figure 2: Agentry SAP Framework System Components



The Foundation Add-On Component (SMFND) is designed for a Netweaver ABAP-based system. It can be deployed to any SAP systems based on Netweaver AS ABAPA 7.x. The foundation component defines the standard integration patterns supported by the framework. It provides class libraries, RFC module pools, system programs, utilities, application configuration tools, and system administration tools. Application integration services are developed using the foundation component.

Application add-on components are developed using the integration patterns, services, and tools provided by the foundation component. Application add-on components provide an application level integration support for mobile applications.

A layered approach is used when developing the application add-ons. A standard application add-on is developed for each SAP Business Suite system and supports all mobile applications for the suite system. There is no mobile application-specific add-on using this approach. Each standard application add-on is designed to provide mobile integration support for all standard business processes and business objects within a specific suite system. Mobile integration services are designed and developed to be shared and reused by multiple mobile applications requiring the same business process and business object.

Currently only onPremise SAP ERP systems and SAP CRM systems are supported by the framework. Add-On SMERP supports the SAP ERP system and Add-On SMCRM supports the SAP CRM system.

The two standard application add-ons supported by the framework are SMERP Add-On for SAP ERP Systems and SMCRM Add-On for SAP CRM Systems. SMERP supports ERP-based mobile applications such as SAP Work Manager, SAP Inventory Manager, and SAP Rounds Manager. SMCRM supports CRM-based mobile applications such as SAP CRM Service Manager and SAP Sales Manager.

SAP Industry Solutions provides specialized industry specific business processes and features. To support these processes, specialized application add-ons have been developed. SMISU Add-On for SAP ERP Systems with IS-Utility is currently supported by the framework. It provides additional support for meter management integration services for SAP IS-Utility systems, in addition to the standard plant maintenance processes supported by the SMERP add-on. SMISU add-on has a dependency of SMERP add-on.

SAP Framework

The framework consists of core components. These components are logically grouped in three main layers:

- Change detection layer
- Business logic layer
- Integration layer

Within the framework, there also exists the configuration module and system monitor. The configuration module provides the interface to the integration framework to allow for the configuration and administration of the various components within it. The system monitor provides an administrative interface to monitor processing related to users, pushes, and other synchronization tasks within the framework.

Change Detection

The change detection layer of the Agentry SAP Framework contains the exchange tables and triggers used to detect and track changes made to the data in the SAP system, and stores exchange information used during synchronization between the mobile application and SAP. Included in the change detection layer are the following:

- **Mobile Exchange Persistent Layer** - Contains the exchange tables used during synchronization to compare data on the Client to data in the SAP system, and to support the synchronization of only the differences.
- **Delta Detection Routines** - Contains the triggers, created in the SAP enhancement framework, to detect changes to production data of importance to the mobile application.
- **Change Detection Configuration Set** - Configuration tools, including the interface presented in the Configuration portal, to allow for the creation, configuration, and administration of the components of the change detection layer.
- **Exchange Object** - Presented in the Configuration Panel, the exchange object encapsulates the SAP tables, exchange table, class handler, and other components involved in the change detection process. Exchange objects are utilized by the fetch process to determine what data has changed since the last transmit of the mobile device.

Business Logic Layer

The business logic layer contains the logical components to work with the exchange data objects in the change detection layer for downstream synchronization, as well as the logical components to update data from transactions in the mobile application to the SAP system. Included in the business logic layer are the following:

- **Application Data Filter Services** - Specifies the filtering of data to be retrieved from the SAP system for transmission down to the mobile application. Not all fields from a given table will necessarily be sent to the mobile application. The application data filter services allow for the specification of the specific fields from a table to be sent to the mobile application.

- **Data Object Class Handler Repository** - Contains the logic used by the mobile data objects to synchronize data. A class handler is created to retrieve data from or update data to the SAP system, making calls to the standard SAP BAPIs.
- **Application Authorization Services** - Contains the security settings specific to the mobile user group. Settings for authorization services can be applicable to users of all mobile applications synchronizing data through the Agentry SAP Framework, for a specific application, or for a specific class handler within the business logic layer.
- **Mobile Integration Configuration Set** - Configuration tools, including the interface presented in the Configuration Panel, to allow for the creation, configuration, and administration of the components of the business logic layer.
- **Mobile Data Object** - Presented in the Configuration Panel, the mobile data object encapsulates the class handler, exchange object (in the change detection layer), the application data filtering services, and other components involved in the synchronization of data. The mobile data object references the exchange object in the change detection layer to support its synchronization. It in turn is referenced by the BAPI wrappers within the integration layer.

Integration

The integration layer of the Agentry SAP Framework contains the BAPI wrappers that present the integration point to the mobile application servers. The Java steplets, complex table, and data table classes within the synchronization definitions of the mobile application call into these BAPI wrappers, through the SAP Java Connector (SAP JCo). The BAPI wrappers, in turn, call into the mobile data objects, which include the business logic for synchronization.

The architecture and design of the BAPI wrappers is intended to exclude the business logic related to the synchronization of the production data. This structure minimizes the necessity of modifying Java code within the mobile application as the result of changes to the synchronization processing configured in the integration of the Agentry SAP Framework.

Configuration Module

The configuration module is the web interface for the Agentry SAP Framework Configuration Panel. This interface provides the tools to create, configure, and administer the components of the integration of the Agentry SAP Framework.

System Monitor

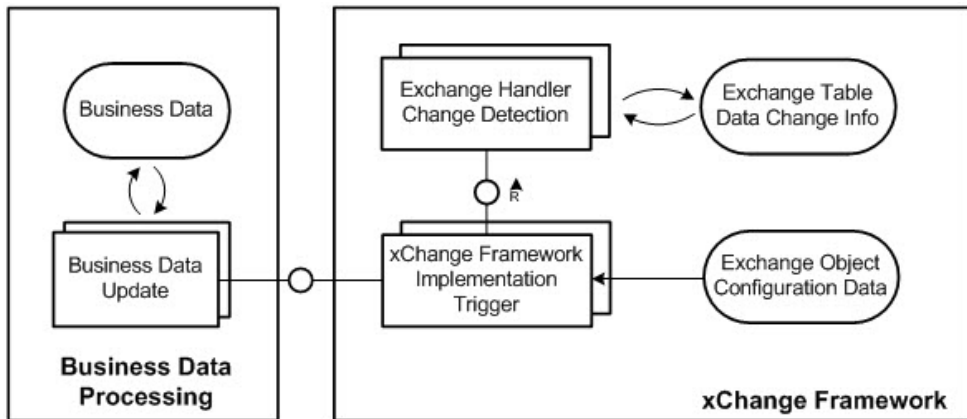
The system monitor is the web interface for the Agentry SAP Framework Administration and Monitoring Panel. This monitoring panel is provided to view mobile users' synchronization activities, including fetches, transactions, and pushes.

Mobile Exchange Persistent Layer

The mobile exchange persistent layer allows the transfer of data from SAP to the mobile client as well as the transfer of data from the mobile client to SAP. This exchange process provides a consistent way to manage and capture mobile client-related master and transaction data

changes within SAP. The mobile exchange persistent layer uses the SAP NetWeaver enhancement framework to implement change detection triggers.

Figure 3: System Diagram - xChange Framework



Delta Detection Routine

The delta detection routines are the triggers within the enhancement framework that detect modifications to data of concern to the mobile application. When changes are made to production data in the back end system, the delta detection routines will capture the change event. If a given change is one that will affect the data in the mobile application, then the routine will update the appropriate exchange table in the mobile exchange persistent layer. Intelligence is incorporated in the delta detection routines to determine if a change is one of concern to the mobile application. If it is not, then the change is ignored by the routine.

Change Detection Configuration Set

The change detection configuration set controls how the change detection should be carried out by the mobile exchange persistent layer. Customer-defined configurations are protected through a reserved customer namespace, such as “Y*” or “Z*”. The change detection configuration set is mobile application-independent. However, it is a set of mobile application-specific change detection rules. The rules are configurable through the Agentry SAP Framework Configuration Panel.

Data Object Handler Class Repository

The data object class handler repository links BAPI wrappers with mobile data objects and encapsulates all business logic related to mobile applications. Internally standard BAPIs or custom logic can be implemented to provide optimal mobile support. The object-oriented design provides benefits such as reliability, reusability, extensibility and maintainability.

All transaction updates are performed through standard BAPIs in order to ensure data integrity. The repository also supports streamlining multiple SAP transactions or multiple BAPIs for improved business processes. The data objects within the repository are application-dependent and delivered in the relevant package based on SAP's hierarchy, such as /SYCLO/MM or /SYCLO/PM.

The mobile data objects contained within the repository support result-set field selection, data filtering and security checks as defined by the integration rules. The mobile data objects also support monitoring and logging applications. Mobile data objects are configurable through the Agentry SAP Framework Configuration Panel. Different rules can be defined for every class handler for each mobile application.

Mobile Integration Configuration Set

The Mobile Integration Configuration Set within the business logic layer contains the user interface components presented in the Configuration Panel. The tools within this configuration set allow for the creation, configuration, and administration of the components within the business logic layer.

Application Authentication Services

Application authentication services allow you to perform additional mobile-related authentications that are not available using the standard SAP authentications and security profiles. These additional authorizations are configured using the Security Settings screen in the Configuration Panel.

BAPI Wrapper

The BAPI wrapper layer provides a consistent way to expose business logic and data from the SAP system to external mobile applications. The BAPI wrappers are decoupled from the business logic in SAP, making them easy to reconfigure or customize.

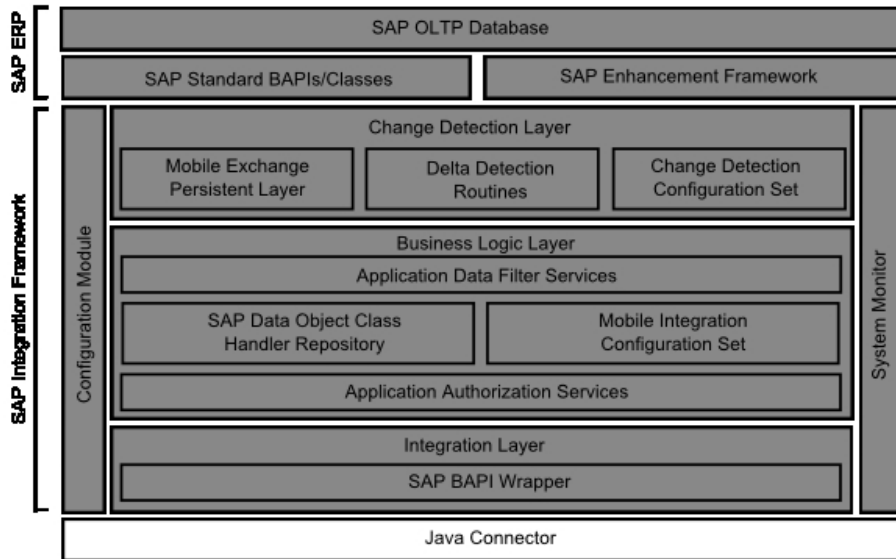
The standard BAPI wrapper library consists of a collection of custom developed BAPIs based on the SAP integration framework. BAPI wrappers improve interface consistency with standard naming conventions and development standards. The BAPI wrapper library is application-dependent and delivered in the relevant package based on SAP's hierarchy, such as /SYCLO/MM or SYCLO/PM. BAPI wrappers set parameters for mobile data objects to post and pull data in the SAP system. When the Server calls to SAP, these calls are encapsulated in a BAPI wrapper to ensure that SAP responds in a consistent manner and performs the actions that Agentry SAP Framework is asking for.

The Configuration Panel is used to configure BAPI wrappers by assigning them to mobile data objects and handler methods (GET, CREATE, UPDATE, or DELETE), function groups, and packages within SAP.

Java Connector

The Java connector is a standard SAP Java connector. See the appropriate SAP documentation for more details.

System Diagram - Java Connector



System Monitor

Administration and monitoring activities that take place behind the SAP Java connector for the Agentry SAP Framework are performed through the System Administration and Monitoring Panel. The Administration Panel is a problem-detection tool utilizing SAP's standard application log database. It allows administrators to trace who is logging into the mobile applications, what work they're doing, and the data being transferred between the mobile devices and the SAP system. The Administration Panel speeds problem resolution using real-time data and traceability.

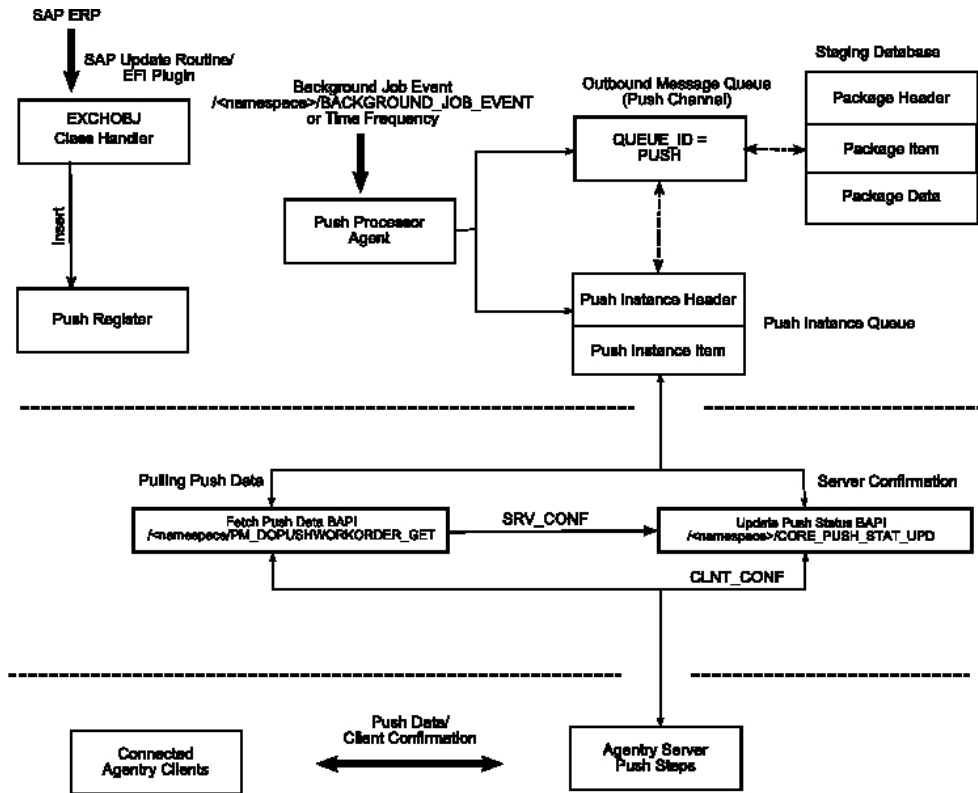
Configuration Module

The Configuration Module is the Agentry SAP Framework configuration web interface panel. The Configuration Panel simplifies modifications to existing mobile applications. It adds a layer of business logic for filtering at a system or user level. Data is mapped to the Agentry SAP Framework and pulled from SAP modules based on the user's credentials in the SAP system. Therefore, administrators can allow remote workers to only navigate and see data relevant to their job location or function.

Working with Push Scenarios

A push scenario pushes emergency work orders to the corresponding recipients. Use the following diagram and steps to follow a push instance from generation in SAP to reception on the Client.

Push Process Flow



1. The push exchange process initiates the push trigger based on the push conditions. Conditions are defined as filter rules in the push exchange object. For instance, `work order priority = 1` is considered an emergency work order in the base product release.
2. The work order that satisfies the push conditions inserts a record into the push register table `/SYCLO/PSH01` with an object key as the work order number and a push status of `NEW`.
3. The event `/SYCLO/BACKGROUND_JOB_EVENT` is raised after the work order is saved, which triggers the background job for the push processor agent.

4. The push processor job /SYCLO/CORE_PUSH_PROC_PROG is triggered, either by using the event or the time frequency. This trigger is based on specific customer processes.
5. The push processor determines the recipients for the push work order and builds the data for each recipient as a separate instance. The instance is stored in the outbound message queue /SYCLO/PSH02 with queue ID = PUSH, using the staging database.
6. The push instance displays one of the following statuses, viewable in the push monitor in the Administration Panel:
 - NEW
 - PROCESS
 - CANCEL
 - COMPLETED
 - SRV_COMP
7. The Agentry application within the SAP Mobile Server calls the push BAPI /SYCLO/PM_DOPUSHWORKORDER_GET for every predefined time interval and checks the push queue for new items.
8. The Agentry application within the SAP Mobile Server sends the push data to the respective Clients depending on the user credentials that match the push instance.
9. Once the Client receives the push message, it sends the Client confirmation back to the Server and the Server calls the BAPI /SYCLO/CORE_PUSH_STAT_UPD to update the confirmation with status CLNT_CONF back to SAP.

Outbound Trigger Overview

An outbound trigger allows a mobile application to interface with external systems such as the Agentry application within the SAP Mobile Server from SAP. Outbound triggers can be integrated into standard mobile application processes, such as push processing. Different types of outbound triggers can be defined, such as an HTTP trigger, file trigger, Web service trigger, etc. You define an outbound trigger definition in the IMG activity.

Requirements

Outbound triggers are configured for each mobile application. Therefore, the mobile application must be defined first. The Outbound trigger handler must be developed before it can be assigned to a trigger. An outbound trigger handler should implement interface /SYCLO/IF_CORE_OUTB_TRIGGER and should be a subclass of /SYCLO/CL_CORE_OTRIG_BASE.

Agentry SAP Framework Administration Functions in SAP

All components of the Agentry SAP Framework administration in SAP, such as BAPI wrappers and mobile data objects, support logging. Activity logs generated by the Agentry SAP Framework are integrated into the standard SAP application log database.

The following are administration and monitoring functions available in SAP:

- Agentry SAP Framework Push Instance Purge Utility

- Agentry SAP Framework Data Cache Purge Utility
- Agentry SAP Framework Generic Purge Utility
- Agentry SAP Framework Exchange Table Purge Utility
- Agentry SAP Framework Subscription Queue Purge Utility
- Agentry SAP Framework Log Deletion
- Agentry SAP Framework Log Display
- Agentry SAP Framework Statistics Log Purge Utility

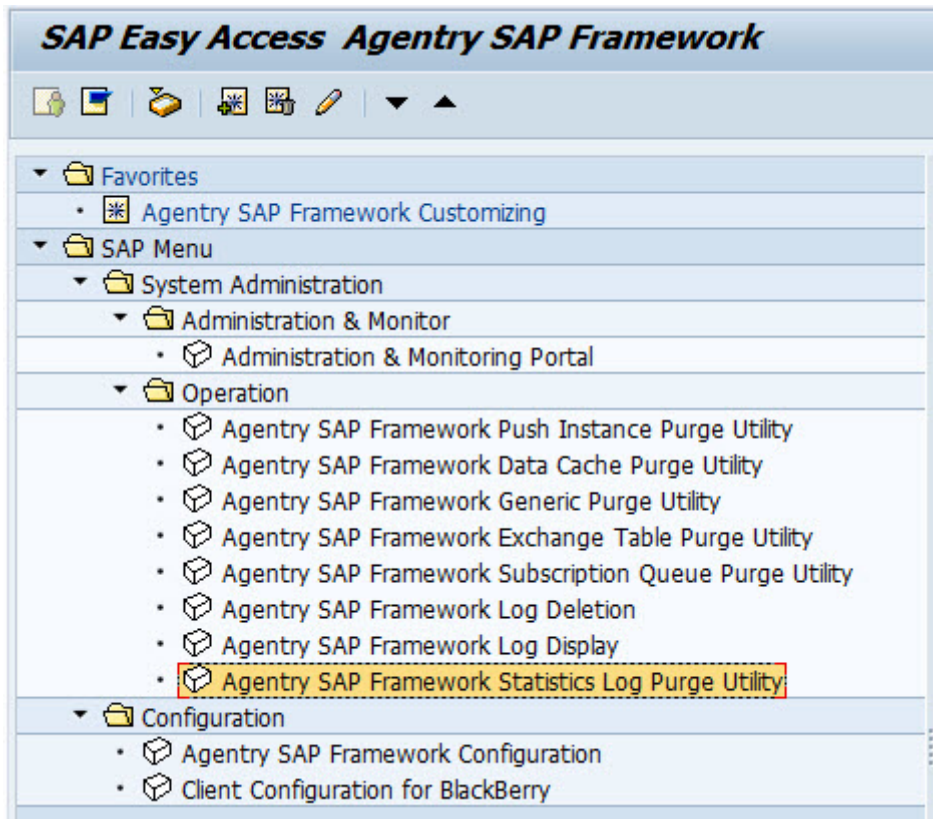
The manual process of purging data is important during configuration and modification in a development environment. Be sure to purge data after each test after reviewing it and starting the next test. This way, the data found in the logs is applicable to the newest test activities and will not cause confusion. Periodic purging of data also contributes to an optimum running environment.

Exchange table histories can be set to automatically delete after a set period of days through the Configuration Panel. All other data purges and log deletions must be done through SAP.

Accessing Administrative Functions in SAP

1. Log into SAP.
2. Type `/n/syclo/smart` into the command field and click the green checkmark to the left of the field, or press **Enter**.

The Agentry SAP Framework Administration window displays. Expand the SAP menu tree by clicking the arrows to the left of the menu items. Expand as follows: **SAP menu > System Administration > Operation**. The available administrative functions display:



3. Double-click on the desired administrative function to open the SAP window for that function.

Agentry SAP Framework Push Instance Purge Utility

The push instance purge utility is used to purge exchange table information that was used during the push processing. Once the push data is pushed to Client devices, the exchange information is no longer needed and can be purged.

Note: The purge utility in SAP performs a manual purge. Automatic purges of push instances are configured in the Push Scenario Definition panel of the Configuration Panel.

When desired fields are filled, click on the clock icon in the upper left to execute the purge.

SAP Administration - Push Instance Purge Utility Program

Syclo Push Instance Purge Utility Program

Runtime Settings

Mobile Application	SAP_CRM_SERVICE_MANAGER_40		
Push Instance GUID		to	
Push Status		to	
Scenario Id		to	
Record Id		to	
Expiration Date	10/08/2013		
Expiration Time	00:00:00		
Maximum No. of Instances			

☒ Test Run

Runtime Settings

- **Mobile Application:** Click the box icon to the right of the Mobile Application field to bring up a window displaying all mobile application choices.
- **Push Instance GUID:** Use the Push Instance GUID range fields to select GUIDs contained within the push instance table in SAP to purge.
- **Push Status:** Use the Push Status range fields to select statuses contained within the push instance table in SAP to purge. By default, all statuses are included if the fields are not filled in. The push statuses are as follows:
 - NEW
 - PROCESS
 - SRV_COMP
 - COMPLETED
 - CANCEL
- **Scenario ID:** Use the Scenario ID range fields to select scenario IDs contained within the push instance table to purge.
- **Record ID:** Use the Record ID range fields to select record IDs contained within the push instance table to purge.
- **Expiration Date:** Use the Expiration Date field to choose a date when the purge utility operation will expire. The expiration date is automatically filled with the current date.
- **Expiration Time:** Use the Expiration Time field to choose a time when the purge utility operation will expire.
- **Maximum No. of Instances:** Type in the maximum number of instances to purge from the instance table.

- **Test Run:** When this box is checked, records are not purged upon execution. Rather, a list of the records selected for purging displays, in order to determine that the purge parameters are correct. Once records are purged, they cannot be recovered.

Agentry SAP Framework Data Cache Purge Utility

Use the data cache purge utility to manage and purge packages associated with messages and the outbound message queue from SAP. The outbound message queue is the only way SAP communicates with the Client device. Packages are the data attached to outbound messages. In this way, an administrator can send a message to multiple users while referencing only one data package, rather than an individual data package for each outbound message.

SAP Administration - Data Cache Purge Utility Program

Syclo Data Cache Purge Utility Program

Runtime Settings

Mobile Application:

Expiration Date:

Expiration Time:

Data Handler Name: to

Data Handler Method: to

Config. Obj. Name: to

☐ Delete Packages

Package Deletion Settings

Storage Unit GUID: to

☒ Delete Outbound Messages

Message Deletion Settings

Message GUID: to

Message Status: to

Message Counter: to

Max Number of Messages Deleted:

☒ Test Run

Runtime Settings

- **Mobile Application:** Select the desired mobile application with which the outbound messages or packages are associated.

- **Expiration Date:** Use the Expiration Date field to choose a date when the purge utility operation will expire. When used with Expiration Time, this is the maximum expiration timestamp from which to purge messages from the cache (i.e.; purge messages that are set to expire 2013.10.06 at 3 p.m. or earlier). If an expiration date / time are provided, they are converted into a timestamp. The utility will only delete those messages whose expiration, or timestamp, is less than or equal to the provided timestamp. If this timestamp is not provided, deletion will occur without respect to the messages' expiration.
- **Expiration Time:** Use the Expiration Time field to choose a time when the purge utility operation will expire. When used with Expiration Date, this is the maximum expiration timestamp from which to purge messages from the cache (i.e.; purge messages that are set to expire 2013.10.06 at 3 p.m. or earlier). If an expiration date / time are provided, they are converted into a timestamp. The utility will only delete those messages whose expiration, or timestamp, is less than or equal to the provided timestamp. If this timestamp is not provided, deletion will occur without respect to the messages' expiration.
- **Data Handler Name:** Use the Data Handler Name range fields to select data handler names contained within the data cache to purge. Data handlers are responsible for messages within packages to be purged. It limits the purge to the class handler responsible for handling the message.
- **Data Handler Method:** Use the Data Handler Method range fields to select data handler methods contained within the data cache to purge. Data handlers are responsible for messages within packages to be purged. It limits the purge to the method of a class handler responsible for handling the message.
- **Configuration Object Name:** Use the Configuration Object Name range fields to select configuration object names contained within the data cache to purge. Limits the purge to the configuration object that owns, or is the source of, the message.

Package Deletion Settings

Select the Delete Packages option to purge packages stored in SAP.

Note: If a data package is associated with a message GUID, it cannot be deleted.

- **Delete Packages:** Select this radio button if you wish to delete the packages found through the Package Deletion Settings.
- **Storage Unit GUID:** Each package is associated with a unique storage unit GUID

Message Deletion Settings

Select the Delete Outbound Messages option to purge outbound messages stored in SAP.

Note: All packages associated with outbound messages will also be deleted if the Delete Outbound Messages option is chosen.

- **Message GUID:** Each outbound message is associated with a unique message GUID
- **Message Status:** Use the following available message statuses:
 - NEW

- SEND
- RECEIVED
- CANCEL
- CONFIRMED
- **Message Status:** Use the Message status range fields to select message statuses contained within the data cache to purge.
- **Message Counter:** Use the Message Counter range fields to select message counters contained within the data cache to purge.

Test Run

When this box is checked, packages or messages are not purged upon execution. Rather, a list of the packages or records selected for purging appears, in order to determine that the purge parameters are correct. Once packages or records are purged, they cannot be recovered.

Agency SAP Framework Generic Purge Utility

Use the generic purge utility to delete records in SAP pertaining to specific user IDs or middleware server records.

SAP Administration - Purge Utility Program

The screenshot shows the 'Syclo: Purge Utility Program' window. It has a title bar with a clock icon. Below the title bar is a section titled 'Middleware Data Selection'. This section contains a 'Mobile Application' field with a dropdown menu showing 'SAP_CRM_SERVICE_MANAGER_40'. To the right of this field is a 'User Guid' field with a 'to' label and another empty field. Below these fields are several checkboxes: 'User Registry', 'Session Registry', 'Object Registry', 'User Push History', 'Cross Reference', and 'Server'. At the bottom of the 'Middleware Data Selection' section is a 'Status Selections' section with a checkbox for 'Mobile Status'. At the very bottom of the window is a checkbox for 'Test Run' which is checked.

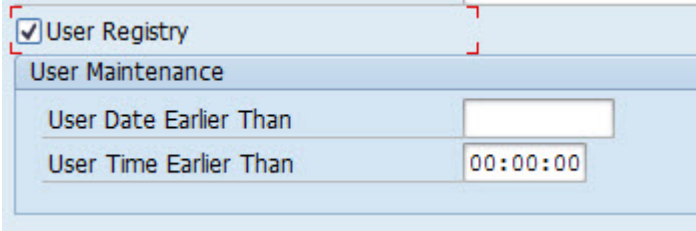
When a box in the MDW Selections field is checked, additional fields for the selection appear, allowing specific purging criteria to be set for the selection.

- **Mobile Application:** Select the desired mobile application with which the middleware server records are associated.
- **User GUID:** Use the User GUID range fields to select user GUIDs contained within the middleware server records to purge.

User Registry

When the User Registry box is checked, all records pertaining to that user ID, such as session and object records, are also purged. Each mobile user ID is a user GUID in SAP. Mobile users need a separate user GUID for each mobile application they use.

Generic Purge Utility - User Registry



The screenshot shows a dialog box titled 'Generic Purge Utility - User Registry'. At the top, there is a checkbox labeled 'User Registry' which is checked. Below this is a section titled 'User Maintenance'. It contains two input fields: 'User Date Earlier Than' and 'User Time Earlier Than'. The 'User Time Earlier Than' field is currently set to '00:00:00'.

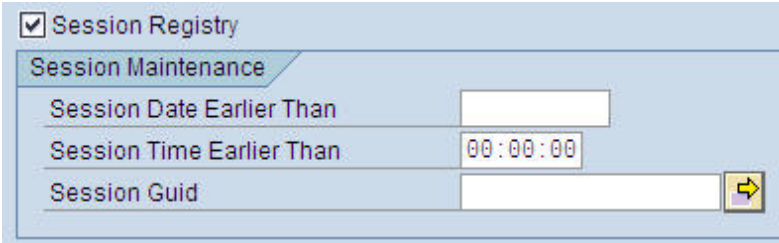
Check the User Registry box to purge user records according to the following criteria:

- **User Date Earlier Than:** Highlight the field and use the calendar to select a date to purge user records existing before the selected date.
- **User Time Earlier Than:** Highlight the field and use the time display window to select a time to purge user records existing before that time.

Session Registry

Every time a user ID connects to SAP, a session record is created. Each session record has a GUID, a start time stamp and an end time stamp.

Generic Purge Utility - Session Registry



The screenshot shows a dialog box titled 'Generic Purge Utility - Session Registry'. At the top, there is a checkbox labeled 'Session Registry' which is checked. Below this is a section titled 'Session Maintenance'. It contains three input fields: 'Session Date Earlier Than', 'Session Time Earlier Than' (set to '00:00:00'), and 'Session Guid'. There is a yellow arrow icon to the right of the 'Session Guid' field.

Check the Session Registry box to purge session records according to the following criteria:

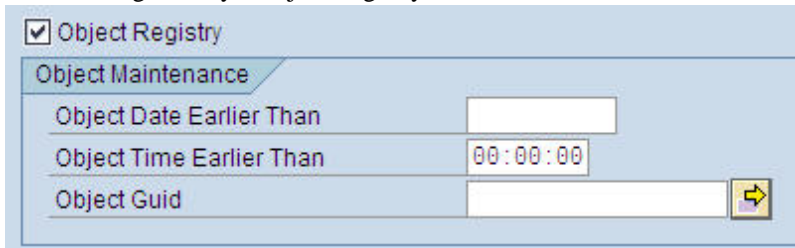
- **Session Date Earlier Than:** Highlight the field and use the calendar to select a date to purge session records created before the selected date.

- **Session Time Earlier Than:** Highlight the field and use the time display window to select a time to purge session records created before that time.
- **Session GUID:** Either type in the GUID or use the Multiple Selection icon to the right of the field to select a range of GUIDs.

Object Registry

Each object on each Client device has a unique GUID contained within the object registry table. The object registry is used by the fetch process to calculate what data is needed by each Client device based on change detections since last transmit.

Generic Purge Utility - Object Registry



Check the Object Registry box to purge object records according to the following criteria:

- **Object Date Earlier Than:** Highlight the field and use the calendar to select a date to purge object records created before the selected date.
- **Object Time Earlier Than:** Highlight the field and use the time display window to select a time to purge object records created before that time.
- **Object GUID:** Either type in the GUID or use the Multiple Selection icon to the right of the field to select a range of GUIDs.

User Push History

When push history is enabled, every time an object is pushed to a user it is recorded to a history table (/SYCLO/PSH05). The user push history purge utility removes old history records, or specific history records, by specifying a date/time or record GUIDs from which to purge.

Note: In order for history records to appear, the push (distribution) handler has to support the history function.

Generic Purge Utility - User Push History

Check the User Push History box to purge object records according to the following criteria:

- **Update Date Earlier Than:** Highlight the field and use the calendar to select a date to purge user push history created before the selected date.
- **Update Time Earlier Than:** Highlight the field and use the time display window to select a time to purge user push history created before that time.
- **Push History Record GUID:** Either type in the GUID or use the Multiple Selection icon to the right of the field to select a range of GUIDs.

Cross Reference

The reference tables contained within SAP facilitate the key mapping process for Agentry applications. When a user creates a local object with a temporary ID and transmits to the system, the object is assigned an object GUID. Agentry can also break down a document into smaller chunks of information, each with its own reference GUID mapped to the object GUID.

Generic Purge Utility - Cross Reference

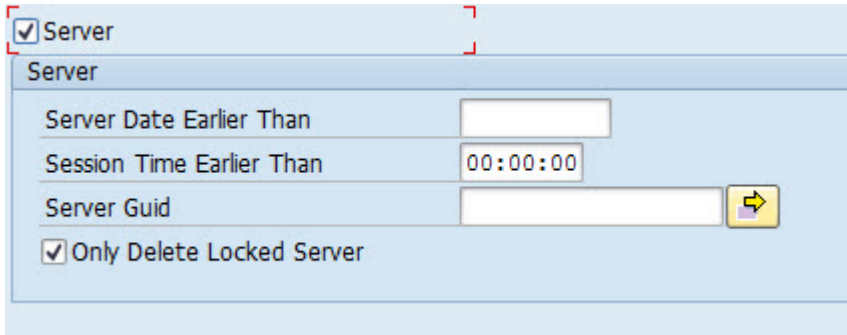
Check the Cross Reference box to purge cross references according to the following criteria:

- **Ref. Obj. Date Earlier Than:** Highlight the field and use the calendar to select a date to purge reference object records created before the selected date.
- **Ref. Obj. Time Earlier Than:** Highlight the field and use the time display window to select a time to purge reference object records created before that time.
- **Reference GUID:** Either type in the GUID or use the Multiple Selection icon to the right of the field to select a range of GUIDs.

Server

A system can contain multiple middleware servers within the system. Each server has a unique GUID associated with it.

Generic Purge Utility - Server




☒ Server

Server

Server Date Earlier Than

Session Time Earlier Than

Server Guid 

☒ Only Delete Locked Server

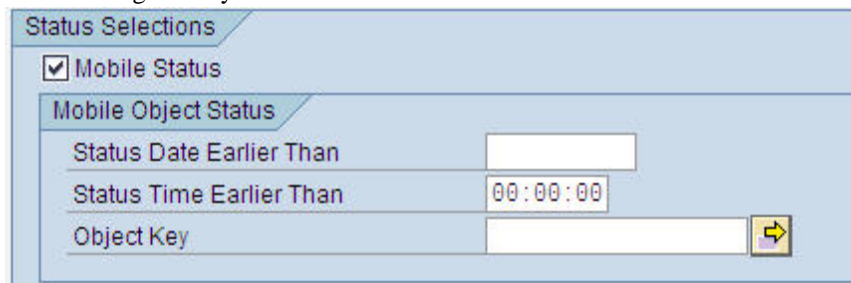
Check the Server box to purge server records according to the following criteria:

- **Server Date Earlier Than:** Highlight the field and use the calendar to select a date to purge server session records created before the selected date.
- **Session Time Earlier Than:** Highlight the field and use the time display window to select a time to purge server session records created before that time.
- **Server GUID:** Either type in the GUID or use the Multiple Selection icon to the right of the field to select a range of GUIDs.
- **Only Delete Locked Server:** Check this box if you only want to purge server records on the locked server.

Mobile Status

Whenever a technician changes the status of an object and transmits from the mobile device, the status management tables in SAP are updated. Each object has a unique object key associated with that object.

Generic Purge Utility - Status Selections




Status Selections

☒ Mobile Status

Mobile Object Status

Status Date Earlier Than

Status Time Earlier Than

Object Key 

Check the Mobile Status box to purge mobile object records according to the following criteria:

- **Status Date Earlier Than:** Highlight the field and use the calendar to select a date to purge mobile status records created before the selected date.
- **Status Time Earlier Than:** Highlight the field and use the time display window to select a time to purge mobile status records created before that time.
- **Object Key:** Either type in the object key or use the Multiple Selection icon to the right of the field to select a range of object keys.

Test Run

Check the Test Run box in order to have SAP display another window that shows in table format the records that will be deleted based on the criteria selected in the Purge Utility Program screen. Once records are purged from the system, they cannot be recovered.

Generic Purge Utility - Deleted Session Table Entries Table

Syco: Purge Utility Program

Deleted Session Table Entries:

Session GUID	User GUID	Server GUID	Time Stamp	Active	Active	Session Close Time	SoLose	Created By	Creation Time Stamp	Chap By
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0		X	20,130,619,215,315	X	EDWARDS	20,130,619,215,314	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,619,215,443	X	EDWARDS	20,130,619,215,443	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,620,171,903	X	EDWARDS	20,130,620,171,903	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,620,174,130	X	EDWARDS	20,130,620,173,626	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0	X		20,130,621,159,000	X	TALLUTO	20,130,621,154,822	TALLUTO
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,621,172,424	X	TALLUTO	20,130,621,168,912	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,621,171,424	X	EDWARDS	20,130,621,171,424	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,621,171,647	X	EDWARDS	20,130,621,171,627	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,621,172,393	X	EDWARDS	20,130,621,172,348	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,621,172,393	X	EDWARDS	20,130,621,172,393	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,170,630	X	EDWARDS	20,130,624,170,630	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,170,838	X	TALLUTO	20,130,624,170,782	TALLUTO
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,170,838	X	TALLUTO	20,130,624,170,817	TALLUTO
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,215,814	X	EDWARDS	20,130,624,215,793	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0	X		20,130,624,215,000	X	LARROS	20,130,624,215,007	LARROS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,215,814	X	EDWARDS	20,130,624,215,814	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,215,558	X	EDWARDS	20,130,624,215,519	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,215,826	X	EDWARDS	20,130,624,215,826	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,213,947	X	EDWARDS	20,130,624,213,942	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,213,947	X	EDWARDS	20,130,624,213,942	EDWARDS
0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0000000A78011E0206A330C90C86E190	0			20,130,624,213,224	X	LARROS	20,130,624,213,212	LARROS

Agentry SAP Framework Exchange Table Purge Utility Program

Use the purge utility for the exchange tables to purge obsolete exchange objects from one or more mobile applications in SAP. Obsolete records are determined based on the purge frequency configured in the Agentry SAP Framework Configuration Panel.

To determine the purge frequency in the Agentry SAP Framework Configuration Panel, navigate to the Exchange Object Configuration menu item under the Mobile Application Settings section. In the tab **Technical Settings**, set the Days to Keep History field to the desired number of days and click the **Save** button.

When field selection is complete, click on the Clock icon at the top left of the screen to execute the exchange table purge.

SAP Administration - Exchange Table Purge Utility Program

Syclo Exchange Table Purge Utility Program

Selection Criteria

Mobile Application	<input type="text"/>	<input type="text"/>	<input type="button" value="↗"/>
Exchange Object	<input type="text"/>	to <input type="text"/>	<input type="button" value="↗"/>
Last Changed By	<input type="text"/>	to <input type="text"/>	<input type="button" value="↗"/>
Object Key	<input type="text"/>	to <input type="text"/>	<input type="button" value="↗"/>
Exchange Action	<input type="text"/>	to <input type="text"/>	<input type="button" value="↗"/>
Record Status	<input type="text"/>	to <input type="text"/>	<input type="button" value="↗"/>

☐ Delete All Entries

Selection Criteria

- **Mobile Application:** Use the mobile application range fields to select one or more mobile applications from which to purge objects from the exchange tables.


Note: Selection choices may vary depending on which mobile applications are available on the system. If there is only one mobile application, this field does not apply.

- **Exchange Object:** Use the exchange object range fields to select one or more exchange objects from which to purge tables.
- **Last Changed By:** Use the range fields to select one or more user names who made changes from which to purge tables.
- **Object Key:** Key of the exchanged object. This ID is governed by how the field OBJKEY is populated in the relevant exchange table. The keys will vary from object to object.
- **Exchange Action:** Use I for Insert, U for Update, and/or D for Delete.
- **Record Status:** [blank] is used for the standard purge, D = Delete, and S = Skip.
- **Delete All Entries:** Select this checkbox to purge all obsolete records from the exchange tables, regardless of the application or exchange objects.








Agency SAP Framework Statistics Record Purge Utility

Use the statistics record purge utility to delete statistic records generated by the SAP integration framework. Statistic records for mobile applications and individual mobile users can be purged selectively.

System performance statistic records purge program



Runtime Settings

Mobile Application	<input type="text" value=""/>		
Statistic Record GUID	<input type="text" value=""/>	to	<input type="text" value=""/> 
Session GUID	<input type="text" value=""/>	to	<input type="text" value=""/> 
User GUID	<input type="text" value=""/>	to	<input type="text" value=""/> 
Record No.	<input type="text" value=""/>	to	<input type="text" value=""/> 
BAPI Wrapper Name	<input type="text" value=""/>	to	<input type="text" value=""/> 
Sort Field	<input type="text" value=""/>	to	<input type="text" value=""/> 
Delete Records Before Date	<input type="text" value="01/09/2012"/>		
Delete Records Before Time	<input type="text" value="23:59:59"/>		



☒ Test Run

Agency SAP Framework Log Deletion

Use the log deletion function to delete expired logs from SAP. When desired fields are filled, click on the Clock icon to execute the deletion.

SAP Administration - Delete Expired Logs

Application log: Delete Expired Logs

Delete logs









All logs are deleted which satisfy the following selection conditions, and for which:

- the expiry date is reached or passed
- the expiry date is not defined

Expiry date

☐ Only logs which have reached their expiry date
☒ and logs which can be deleted before the expiry date

Selection conditions

Object	<input type="text" value=""/>		<input type="text" value=""/>	
Subobject	<input type="text" value=""/>	to	<input type="text" value=""/>	
External ID	<input type="text" value=""/>	to	<input type="text" value=""/>	
Transaction code	<input type="text" value=""/>	to	<input type="text" value=""/>	
User	<input type="text" value=""/>	to	<input type="text" value=""/>	
Log number	<input type="text" value=""/>	to	<input type="text" value=""/>	
Problem class	<input type="text" value=""/>	to	<input type="text" value=""/>	
from (date/time)	<input type="text" value=""/>		<input type="text" value="00:00:00"/>	
to (date/time)	<input type="text" value="11.04.2010"/>		<input type="text" value="23:59:59"/>	

Options

☐ Only calculate how many
☐ Generate list
☒ Delete immediately

Delete by Number of Logs

COMMIT Counter

Expiry Date

Click the first radio button to only delete logs which have reached their expiry date. Click the second radio button to delete logs that have reached their expiry date as well as logs that can be deleted before their expiry date has passed.

Selection Conditions

- **Object:** Select the desired object by either typing it in the field or clicking on the selection icon. The object is usually set to **/syclo/**.
- **Subobject:** Select from the following subobjects if the main object is **/syclo/**:
 - **ADMIN** - Administration portal logs
 - **BAPI** - Client application logs
 - **CONFIG** - Configuration portal logs
 - **DEFAULT** - All logs not covered through the rest of the subobjects
 - **EXCHANGE** - Exchange and transaction process logs
- **External ID:** Each log entry has an external ID, depending on what the log is for.
- **Transaction Code:** Not used
- **User:** SAP user ID that created the logs
- **Log Number:** Each log entry has a log number associated with it. If the log number or range of numbers is known, enter them here.
- **Problem Class:** The problem class of the logs
- **from (date/time):** Select the beginning start date and time of the logs to delete by clicking in the fields and using the calendar and the time window display to choose the correct date and time.
- **to (date/time):** Select the end date and time of the logs to delete by clicking in the fields and using the calendar and the time window display to choose the correct date and time.

Options

- **Only calculate how many:** Click this radio button to generate a popup window with the number of table logs that fit the criteria for deletion. Selecting this option does not result in any log deletion.
- **Generate list:** Click this radio button to generate a list of which table logs fit the field criteria for deletion. This list opens a different SAP window with multiple options for working within the list, including individual selection of table logs to delete. Selecting this option does not result in any log deletion.
- **Delete immediately:** Click this radio button to delete all table logs immediately. The system will still confirm the deletion of logs in a popup window before permanently deleting them.

Delete by Number of Logs

- **COMMIT Counter:** Type in the number of logs desired to delete.

Agentry SAP Framework Log Display

Use the log display utility to view activity logs based on the criteria selected on the main screen. When all desired criterion are selected, click on the Clock icon at the top of the screen to execute the request and display the desired logs.

SAP Administration - Analyze Application Log

Analyze Application Log

Object	/SYCLO/	
Subobject		
External ID		

Time Restriction

From (Date/Time)	12.05.2010		00:00:00	
To (Date/Time)	12.05.2010		23:59:59	

Log Triggered By

User	*	
Transaction code	*	
Program	*	

Log Class

☐ Only very important logs
☐ Only important logs
☐ Also less important logs
☒ All logs

Log Creation

☒ Any
☐ Dialog
☐ In batch mode
☐ Batch Input

Log Source and Formatting

☒ Format Completely from Database
☐ Format Only Header Data from Archive
☐ Format Completely from Archive

- **Object:** Select the desired object by either typing it in the field or clicking on the selection icon to the right of the field. The object is usually set to /syclo/ to view logs.
- **Subobject:** Select from the following subobjects if the main object is /syclo/:
 - **ADMIN** - Administration portal logs
 - **BAPI** - Client application logs
 - **CONFIG** - Configuration portal logs

- **DEFAULT** - All logs not covered through the rest of the subobjects
- **EXCHANGE** - Exchange and transaction process logs
- **External ID:** Type the external ID into the field.

Time Restriction

- **From (Date/Time):** Click on the white square icons to the right of the date and time fields to select a start date and time of the beginning of the logs chosen for display. The date and time are automatically set for the current date at 00:00:00 hours.
- **To (Date/Time):** Click on the white square icons to the right of the date and time fields to select an end date and time of the final logs chosen for display. The date and time are automatically set for the current date at 23:59:59 hours.

Log Triggered By

- **User:** SAP user ID
- **Transaction Code:** Standard SAP transaction codes
- **Program:** Standard SAP programs

Log Class

Select the appropriate standard SAP log class. Log classifications are based on the implementation by the developer.

Log Creation

Select the appropriate log creation setting. These are standard SAP settings based on the implementation by the developer.

Log Source and Formatting

Select the appropriate log source and formatting setting. These are standard SAP settings based on the implementation by the developer.

Enable SAP Solution Manager to Diagnose Agentry Issues

You can use the SAP Solution Manager to diagnose issues with the Agentry SAP Framework.

To enable this **SAP Solution Manager 7 EhP 1** must be installed.

- OSS Note 1371097 Diagnostics Setup for Agentry Servers
- End-To-End Root Cause Analysis System Landscape Setup Guide available here:

<https://service.sap.com/~sapidb/011000358700000074392009E>

Reporting Issues Using SAP Service Marketplace

You can report issues with the Agentry SAP Framework using the SAP Service Marketplace. Customer issues entered into the SAP Service Marketplace are automatically sent to SAP's Technical Support team.

Java Development for SAP

In order to modify the communications between the SAP Agentry Server and the SAP system, it is necessary to implement a Java development environment. This section contains some information to aid the developer or implementor in creating this environment.

Set Up the SAP Java Project in Eclipse

Once the Java IDE has been installed, you must set up the development or build project that will allow you to modify, compile, and debug the Java portion of the SAP application.

The following items must be a part of your Java development project:

- The Java source files for the SAP application
- The jCo.jar file
- The Agentry-level *.class files, provided with the SAP Agentry Server in the sub-directory Java\Syclo\Agentry. All of the *.class files in this folder should be part of the project.
- Optionally, the junit.jar file, provided with the Eclipse Java IDE.
- Retrieve the Java source files for the SAP application by contacting SAP's Customer Support. Included in these files will be the Java source files for all of the Java classes used by the SAP. These include the steplet classes used in the Java step definitions, and the classes used by the data tables and complex tables of the application. When creating the project in your Java IDE, the Java source files from SAP should be imported to the development project in the IDE.
- The Java files provided by SAP are the source files for your project. The *.class files must also be a part of the project's build path. These are the Java files that make up the Agentry Java API. They are installed with the SAP and can be referenced from the Server's installation location, or they can be copied to a location convenient to the IDE.
- The jCo.jar file for the SAP Java Connector toolkit must be a part of the build path for the project. This can be referenced in the SAP Agentry Server location, or at some other location convenient to the Java IDE. If the Editor is installed to a separate host system from that of the SAP Agentry Server, the JCo.jar file must be extracted from the Java Connector ZIP file. It can be placed in any convenient location. Be sure to note this location as it will be needed in the Java project created in the IDE.
- SAP recommends that you include this junit.jar file in your build path if you are using the Eclipse IDE. The junit.jar file is provided with Eclipse and is a library of Java classes used for debugging purposes, several of which are implemented in the SAP Java classes. If this

is not included in your project, you will receive warnings related to these classes at build time. The junit.jar file can be found in the Eclipse install directory at:

```
C:\eclipse\plugins\org.junit_4.2.8
```

Java Architecture

The Java back end uses classes of type SAPObject to represent data objects sent to and from the client. Most of these classes are POJOS (Plain Old Java Objects), meaning they store data in fields, provide accessory/mutator methods, and know how to construct themselves when told to by other code.

SAPObjects

SAPObjects can be composed of other SAPObjects stored as arrays of SAPObjects (e.g., Notifications have NotificationItems, etc). Since fetch BAPIs will now bring down child objects as well, the logic will populate the children using a single BAPI call without the need for read steps.

StepHandler

Stephandler classes provide the calling interface to classes subclassing the Agentry Java API (steplets, data tables, complex tables). Methods in StepHandler classes should be static. StepHandler methods also provide an interface for JUnit test suites.

All of this metadata is encapsulated in the SAPObject class rather than in an external file.

BAPI

The BAPI class encapsulates all of the BAPI processing needed by the Java back end. It is abstract because there are specific kinds of BAPIs:

- **FetchBAPI:** FetchBAPIs are the type of BAPIs that create SAPObjects from the exchange process. The SAPObjects are then parsed by Agentry for use on the client.
- **DataTableBAPI**
- **ComplexTableBAPI**
- **TransactionBAPI:** TransactionBAPIs take an Agentry transaction on the client and turn it into one or more JCO.Tables, then run the BAPI passing the tables in order to update SAP.

The BAPI objects that the code will use will be subclasses of FetchBAPI, TransactionBAPI, DataTableBAPI or ComplexTableBAPI. Developers need to write these BAPI classes to call the BAPIs to do the things that the “application” needs to do. A developer-written BAPI is a specific ABAP function call for a particular application (i.e., fetch all work orders for a user, send up a locally added notification, etc.).

Note that there is not necessarily a one-to-one correspondence between BAPI classes and ABAP function names (e.g. /SYCLO/MM_DOPHYSINVDOK_GET).

BAPI classes do the following:

- Create themselves out of the JCO.Repository when necessary. This is just before they are about to be called, but a developer might make these poolable/cacheable.
- Set input values in JCO.Table parameters, such as search ranges in the case of FetchBAPIs or transaction data in the case of TransactionBAPIs.
- Are executed when called (Get the results or post the data to SAP).
- Report exceptions and errors back to calling code. This includes reading return tables and parsing for error messages when necessary.

Data Flow

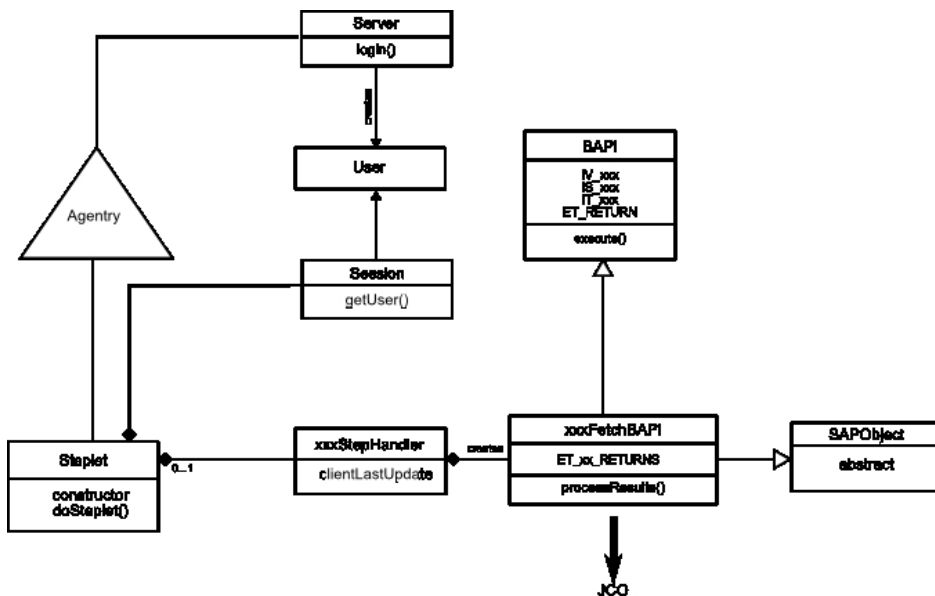
The synchronization processes described in the following sections illustrate the default processes as provided by SAP with the application. During implementation, these standard processes may be configured to support implementation-specific needs.

Data Flow - Fetch

A fetch defines how the SAP Agentry Server synchronizes data for a target object collection. This object collection must be a top-level collection within the module. A fetch is made up of steps that retrieve the data for the collection from the back end system. These steps are grouped into three categories within the Fetch definition: Client Exchange Steps, Server Exchange Steps, and Removal Steps. A fetch may also include properties to store data captured from the user and validation rules for those property values.

The following diagram and steps depict what happens when the Agentry Client must load or reload a fetch.

Figure 4: Data Flow - Fetch



1. A Server exchange steplet defined in a fetch calls the steplet `doSteplet()` method to fetch objects for the user.
2. The `doSteplet()` method calls the appropriate method in the appropriate `xxxStepHandler` class.
3. The `xxxStepHandler` method instantiates the necessary appropriate `xxxFetchBAPI` object, passing the `User` object and `clientLastUpdate` parameter to it.
4. The `xxxFetchBAPI` constructor retrieves the `JCo` function object from the repository, using the connection on `User`.
5. `xxxFetchBAPI` class sets `BAPI` import parameters `IV_XXX` and/or `IS_XXX`.
6. `xxxFetchBAPI` constructor adds `IT_XXX` records to `BAPI` import tables for search criteria or other input parameters.
7. `xxxStepHandler` method calls `processResults()` method from `xxxFetchBAPI`.
8. `xxxFetchBAPI` `processResults()` calls `execute()` method on `BAPI` and checks for exceptions.
9. `xxxFetchBAPI` `processResults()` reads appropriate `ET_xx_RETURNS` table for error messages.
10. `xxxFetchBAPI` `processResults()` iterates over `ET_xx_RETURNS` table and reads records from the table.

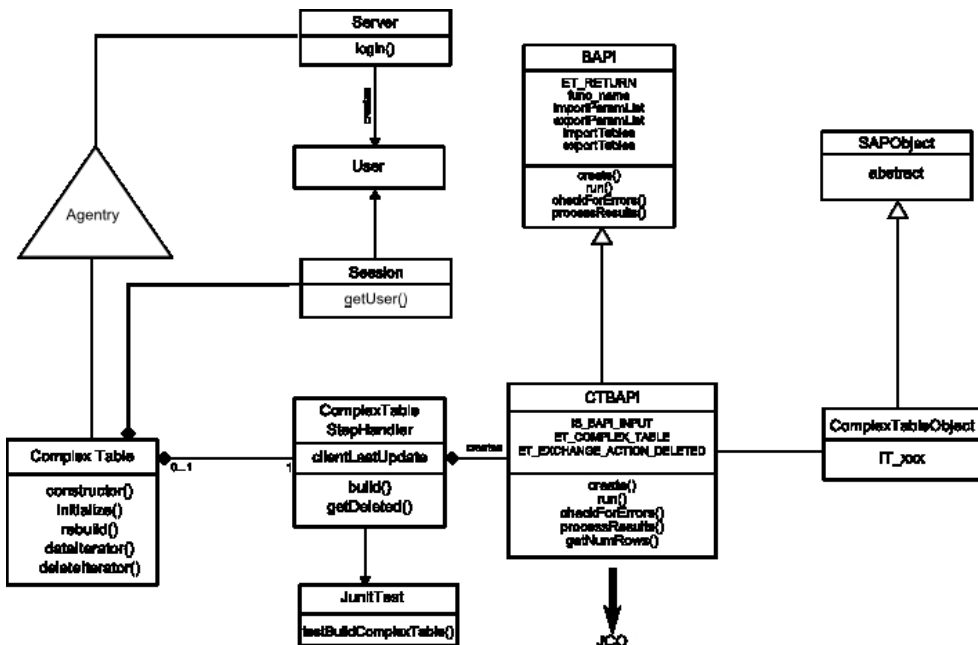
11. For each record, xxxFetchBAPI processResults () calls appropriate constructor in the appropriate SAPObjct subtype.
12. The SAPObjct subtype constructor maps JCo record column names to field names.
13. xxxFetchBAPI processResults () collects these SAPObjcts in an array and passes it back to xxxStepHandler.
14. xxxStephandler passes SAPObjcts array back to steplet doSteplet () .
15. Steplet doSteplet () stores SAPObjcts array in _returnData.
16. Agency application within the SAP Mobile Server parses _returnData and sends object collection up to Client.

Data Flow - Complex Table

The complex table definition defines a table of records containing multiple fields stored on the SAP Client in a structured and searchable format. A complex table can contain large amounts of data with records numbering in the thousands. Included in the complex table are the fields for its records and indexes on fields to provide search functionality and structure to the overall data in the table. The complex table definition also defines how its data is synchronized.

The following diagram and steps depict what happens when the Agency Client must load or reload a complex table.

Figure 5: Data Flow - Complex Table



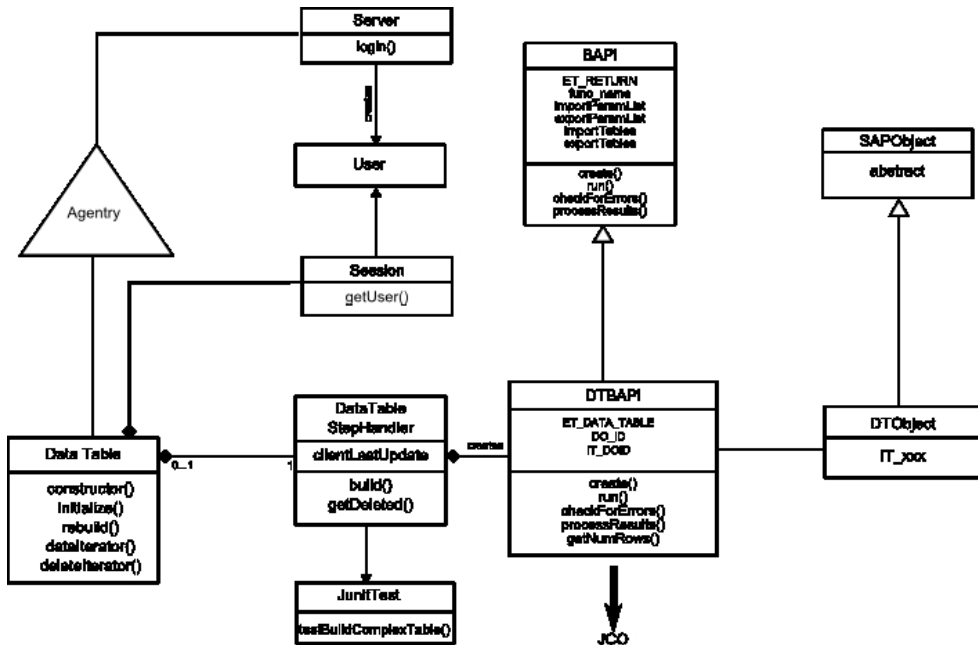
1. The complex table's `initialize()` method is called.
2. The `initialize()` method calls a `ComplexTableStepHandler build()` static method passing the User object.
3. The `ComplexTableStepHandler build()` method constructs the necessary CTBAPI class, passing the User object and `clientLastUpdate` parameter to it.
4. The CTBAPI constructor retrieves the JCo function object from the repository, using the connection on User.
5. CTBAPI sets BAPI import parameter `IS_BAPI_INPUT`.
6. CTBAPI adds `IT_XXX` record(s) to `ComplexTableObject` import tables for search criteria or other input parameters.
7. `ComplexTableStepHandler build()` method calls `getNumRows()` method in CTBAPI.
8. CTBAPI `getNumRows()` method calls the `execute()` function and checks for exceptions.
9. CTBAPI `getNumRows()` method reads `ET_RETURN` table in the BAPI class for error messages.
10. CTBAPI `getNumRows()` method iterates over `ET_COMPLEX_TABLE` and reads records from the table.
11. For each record, CTBAPI `getNumRows()` method calls the appropriate constructor in the appropriate SAPObjct subtype.
12. The SAPObjct subtype constructor maps the JCo record column names to field names.
13. CTBAPI `getNumRows()` method collects the SAPObjcts in `ComplexTableIterator` and passes them back to `ComplexTableStepHandler`.
14. `ET_EXCHANGE_ACTION_DELETED` is read and follows the same steps as 10 - 13. These SAPObjcts populate another `ComplexTableIterator`.
15. `ComplexTableHandler` passes the iterators back to `ComplexTable dataIterator()` and `deleteIterator()`.
16. `ComplexTable initialize()` stores the iterators in `ComplexTableIterator` and returns them to Agentry in `dataIterator()` and `deleteIterator`.
17. Agentry application defined in the SAP Mobile Server parses these iterators and sends table rows up to Client.

Data Flow - Standard Data Table

The complex table selection property type is used to store a selection made by the user from a complex table. The value stored in a complex table selection property is the key field of the selected record within the complex table. The data type of this value will be a string, integral number, or decimal number, based on the data type of the key field.

The following diagram and steps depict what happens when the Agency Client must load or reload a data table.

Figure 6: Data Flow - Standard Data Table



1. The data table's `initialize()` method is called.
2. The `initialize()` method calls a `DataTableStepHandler build()` static method passing the `User` object.
3. The `DataTableStepHandler build()` method constructs the necessary `DTBAPI` class, passing the `User` object, `clientLastUpdate` parameter, and table name to it.
4. The `DTBAPI` constructor retrieves the `JCO` function object for `/SYCLO/CORE_DT_GET` from the repository, using the connection on `User`.
5. `DTBAPI` sets the table name in the table `IT_DOID`, in the field `DO_ID` in the `BAPI` import parameters.
6. `DTBAPI` adds `IT_xxx` record(s) to `DTObject` import tables for search criteria or other input parameters.
7. `DataTableStepHandler build()` method calls `getNumRows()` method in `DTBAPI`.
8. `DTBAPI getNumRows()` calls the `execute()` function and checks for exceptions.
9. `DTBAPI getNumRows()` reads `ET_RETURN` table in the `BAPI` class for error messages.

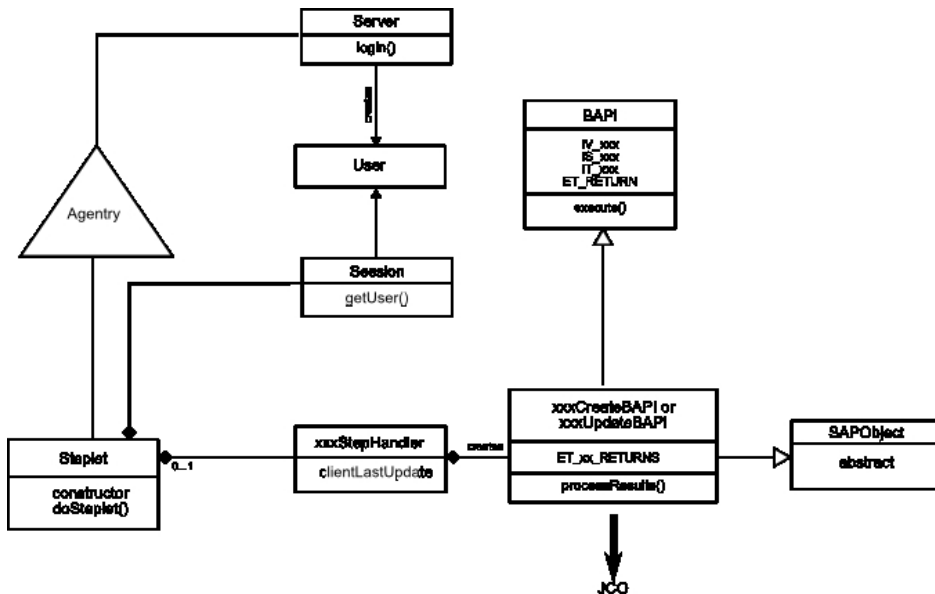
10. CTBAPI `getNumRows()` method iterates over `ET_DATA_TABLE` and reads records from the table.
11. For each record, DTBAPI `getRows()` method calls the appropriate constructor in the appropriate SAPObjct subtype.
12. The SAPObjct subtype constructor maps the JCo record column names to field names.
13. DTBAPI `getNumRows()` method collects the SAPObjcts in DataTable and passes them back to DataTableStepHandler.
14. DataTableStepHandler `build()` method passes the DataTableObject back to the `initialize()` method and is then stored in the appropriate field.
15. Agentry application within the SAP Mobile Server parses these iterators and sends the table rows to the client.

Data Flow - Transaction

The transaction definition defines data to be captured on the SAP Client. As a part of its definition, the transaction includes a target object type, data values to be captured, client-side data validation, and updating its data to the back end system by the SAP Agentry Server during synchronization. Transactions can add new object instances, edit an existing object, delete an object, or modify a complex table or data table record. Each of these behaviors is exhibited by a different transaction type, selected during the creation of the transaction.

The following diagram and steps depict what happens when the Agentry client must load or reload a transaction.

Figure 7: Data Flow - Transaction

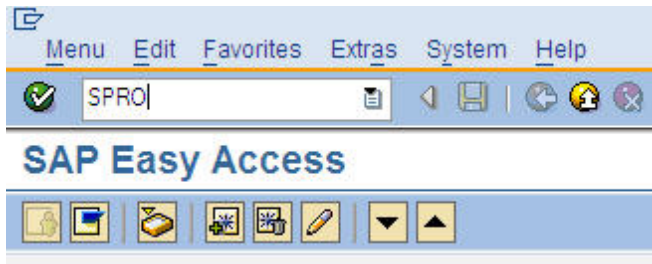


1. A Server exchange steplet defined in a Transaction calls Steplet doSteplet() method to begin the transaction.
2. Steplet's doSteplet() calls appropriate method in the xxxStepHandler class to create, update, or delete objects in SAP.
3. xxxStephandler method constructs the necessary SAPObjct subtype by passing the User to the SAPObjct's constructor.
4. xxxStepHandler method instantiates the necessary xxxCreateBAPI or xxxUpdateBAPI object, passing the SAPObjct to constructor.
5. The xxxCreateBAPI or xxxUpdateBAPI constructor maps the SAPObjct's fields to the necessary JCO record columns.
6. xxxCreateBAPI or xxxUpdateBAPI constructor retrieves JCO function object from the repository, using the connection on User.
7. xxxCreateBAPI or xxxUpdateBAPI constructor sets BAPI import parameters IV_XXX and/or IS_XXX.
8. xxxCreateBAPI or xxxUpdateBAPI constructor adds IT_XXX records to BAPI import tables.
9. xxxStepHandler method calls add(), update() or delete() method of xxxCreateBAPI or xxxUpdateBAPI.
10. xxxCreateBAPI or xxxUpdateBAPI calls the execute() method on BAPI and checks for exceptions.

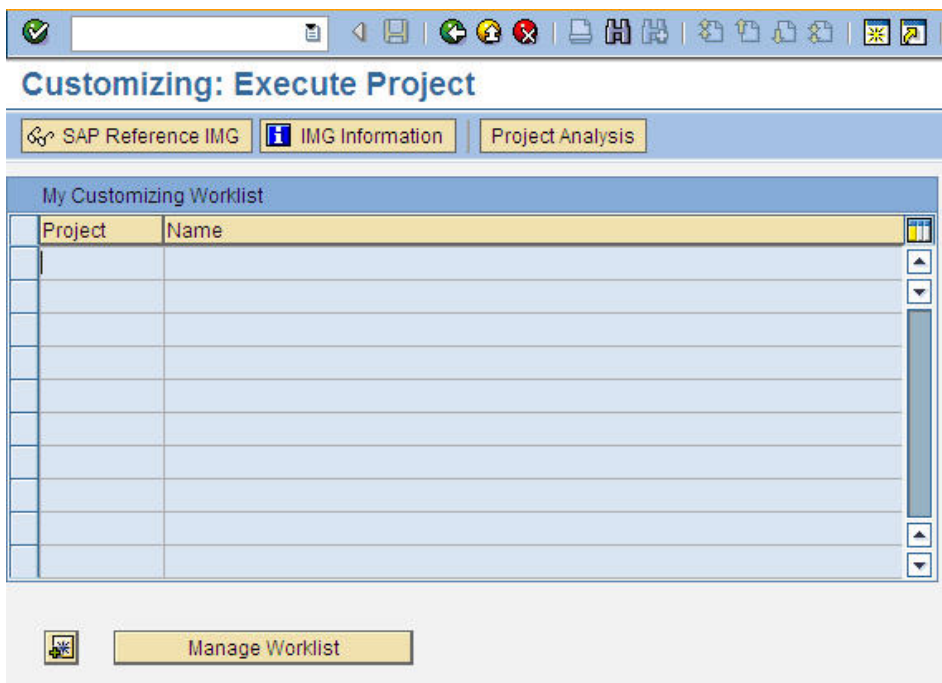
11. xxxCreateBAPI or xxxUpdateBAPI reads ET_RETURN table for error messages.
12. If successful, control passes back to xxxStepHandler, Steplet, and Agency.

Accessing the Agency SAP Framework Configuration Panel

1. Log into SAP.
2. Type the command `SPRO` into the command box and click the green check mark or click **Enter**.

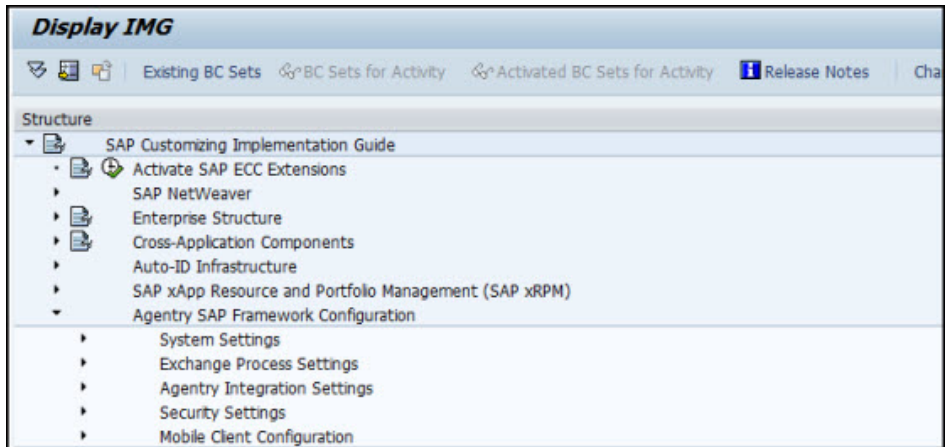


The Customizing: Execute Project screen displays.



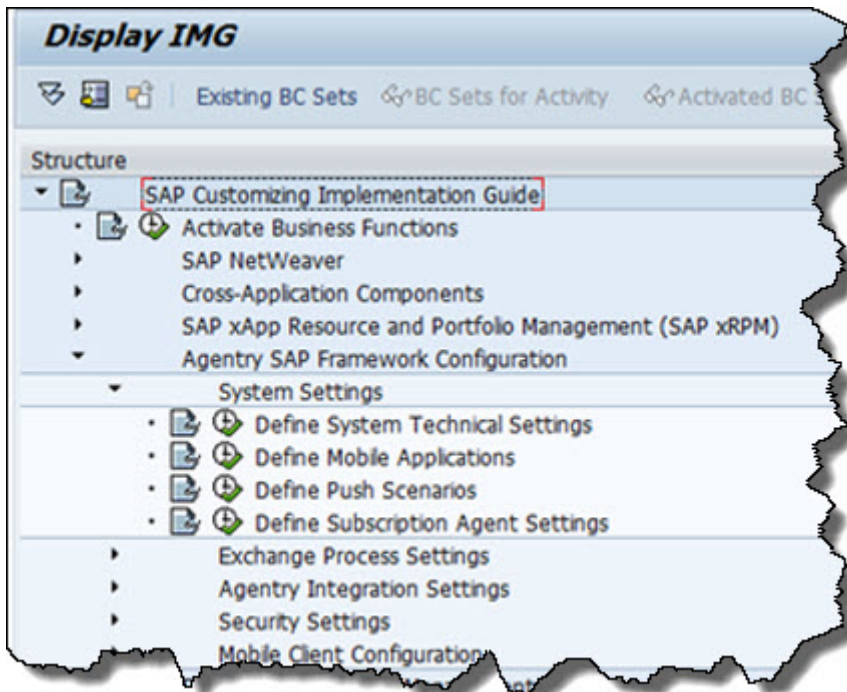
3. Click the **SAP Reference IMG** button.

The Display IMG screen displays.



4. Expand the Agentry SAP Framework Configuration line by clicking on the arrow to the left of the line.

Agentry SAP Framework submenus display. Expand any one of the submenus to display specific configuration functions and click the clock icon to open the Agentry SAP Framework ConfigPanel.



Note: Clicking the paper icon displays a screen with a brief description of the specific configuration function.

The SAP NetWeaver Web Application Server log on screen opens in a browser window.

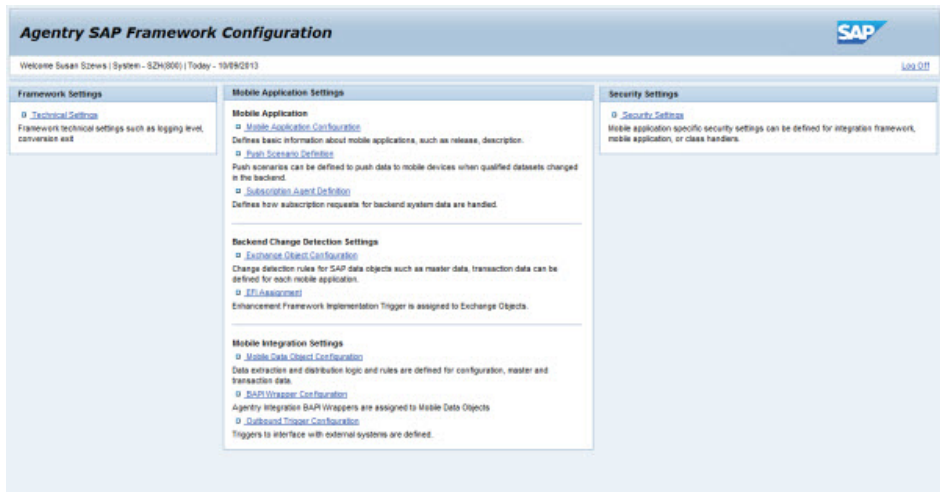
Note: Depending on your configuration, the log on screen may be different than what is shown.

5. Expand any one of the submenus to display specific configuration functions and click the clock icon to open the Agentry SAP Framework Configuration panel.
6. Fill in any necessary fields for your specific implementation and click **Log On**.

The Agentry SAP Framework Configuration portal opens in the browser window. The window that opens corresponds to the submenu line item chosen in SAP.

7. Click the ConfigPanel Home link at the top of the screen to navigate to the main configuration page.

The Agency SAP Framework Configuration Panel home page displays.



Standard Operations in the Configuration Panel

While each mobile application configuration is unique, there are certain standard buttons and options available to perform the configurations.

Filtering by Mobile Application

If more than one mobile application is available on the same system, you can use the filter function to only view items for a specific application. The filter option is found on the main portal home page, as well as any other page where multiple application items could be displayed.

To filter by application, click the arrow to the right of the Mobile Application Filter field and select the appropriate mobile application. To remove the selection and view all items for all mobile applications on the system, click in the field again and select the asterisk (*) symbol.

Creating, Copying, Deleting, and Changing Items

There are four standard actions available to configure different components and items within your mobile application setup.

- **Create:** Creates a new item. All modifiable fields are empty.
- **Copy:** Copies the item that was highlighted and creates a new item. All modifiable fields are filled in with the information from the existing item and are available for changes prior to saving.
- **Delete:** Deletes the highlighted item.
- **Change:** Allows changes to be made to the highlighted item in the modifiable fields.

Saving or Cancelling Changes to an Item

Once the Create, Copy, or Change button is clicked, the Save and Cancel buttons appear. After making any changes to the configuration, click **Save** to save the changes or **Cancel** to discard the changes.

Note: If the Save and Cancel buttons are active, the ConfigPanel Home main menu link is not available. You must either save your changes or cancel out of the changes in order to return to the main Configuration portal page.

Message List

Certain actions can generate system messages. These messages can be error messages or informational messages. If you perform an action that prompts a system message, a message bar appears above the main panel with a brief description of the message.

Click the **Show List** button to display the detailed view of the message list.

Figure 8: Configuration Panel - Detailed Message Display

Current Messages		Log
	Type	Message Text
	All	
		Choose the key from the allowed namespace
Row 1 of 1		

Agentry SAP Framework Configuration Panel Overview

All configuration activities for Agentry SAP Framework are performed using the Configuration screen.

Configuration changes made through the Configuration portal have significant impact to the behavior of the framework component and mobile applications. Changes should be made in the development environment and fully tested before they are migrated to the rest of the SAP system landscape.

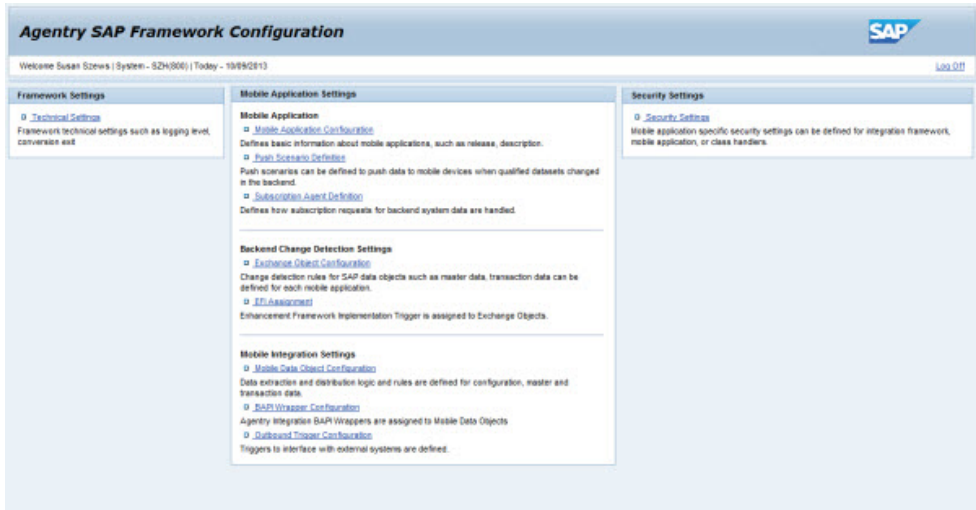
The following user authorizations are required in order to work with the configuration portal:

- *Authorization object - S_ICF*

- *Authorization field* - ICF_FIELD - SERVICE
- *Authorization field* - ICF_VALUE - SYCLOADM
- *Authorization object* - S_TCODE
- *Authorization field* - TCD - /SYCLO/CONFIGPANEL

If you create additional security roles through the Security Settings panel in the ConfigPanel, you must incorporate them into the system as well.

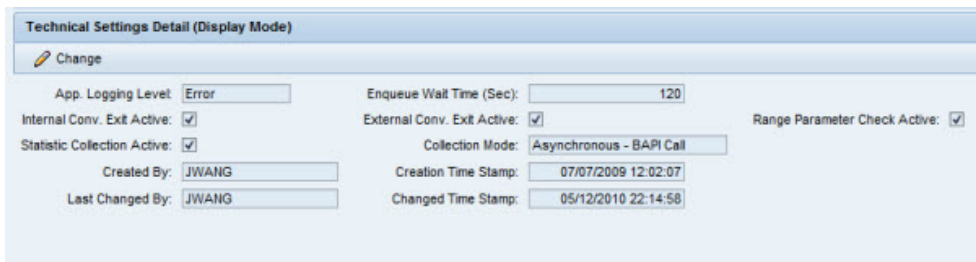
Figure 9: Agentry SAP Framework Configuration Panel Main Screen



Configuration Panel - Technical Settings

Framework technical settings affect all components of the framework, including mobile data objects, BAPI wrappers, and exchange objects.

Figure 10: Agentry SAP Framework Configuration Panel Technical Settings Screen



Configuration Panel - Mobile Application Settings

Mobile application settings are used to define and configure how the mobile application, such as SAP, functions. There are three areas used to configure the mobile application:

- **Mobile Application Configuration** - Defines basic information about mobile applications, such as release and descriptions
- **Push Scenario Definition** - Defines Push Scenarios to push data to mobile devices when qualified data sets change in the back end
- **Subscription Agent Definition** - Defines how subscription requests for back end system data are handled

See the applicable sections in this manual for further information.

Configuration Panel - Back End Change Detection Settings

Back end change detection settings are used to define and configure how the mobile application, such as SAP, communicates with SAP and the object tables contained within SAP. There are two areas used to configure the back end change detection:

- **Exchange Object Configuration** - Change detection rules for SAP data objects, such as master data and transaction data, defined for each mobile application
- **EFI Assignment** - Enhancement Framework Implementation trigger assigned to Exchange Objects

Note: You must create tables and objects in SAP and Agentry before you can create or configure information in the Configuration Panel.

See the applicable sections in this manual for further information.

Configuration Panel - Mobile Integration Settings

Mobile integration settings are used to link BAPI wrappers with mobile data objects and encapsulate the business logic related to the mobile application.

There are three areas used to configure mobile integration:

- **Mobile Data Object Configuration** - Data extraction and distribution logic and rules defined for configuration of master data and transaction data
- **BAPI Wrapper Configuration** - Agentry Integration BAPI Wrappers assigned to Mobile Data Objects
- **Outbound Trigger Configuration** - Triggers to interface with external systems

See these sections in the manual for further information.

Configuration Panel - Security Settings

Use the Security Settings page in the Configuration Panel to set mobile application security parameters at the following levels:

- **System**- Security at this level is application-independent and applies to all components built on the SAP integration framework
- **Product** - Security at this level is at the SAP application level
- **Class Handler** - Security at this level is specific to a data object class handler

All security checks are carried out by the SAP integration framework at runtime.

Technical Settings

Use the Technical Settings Detail screen in the Agentry SAP Framework Configuration Panel to change the settings for the application logs viewable in SAP. Here, you can change framework technical settings such as the logging level and conversion exit.

Technical Settings Detail (Display Mode)

Change

App. Logging Level: Enqueue Wait Time (Sec):

Internal Conv. Exit Active: ☒ External Conv. Exit Active: ☒ Range Parameter Check Active: ☒

Statistic Collection Active: ☒ Collection Mode:

Created By: Creation Time Stamp:

Last Changed By: Changed Time Stamp:

Application Logging Level

Defines the logging level for all framework components. Logging entries are recorded in the SAP application log database under the object **/sycolo/**. The logging levels are:

- No logging
- Abort
- Error
- Warning
- Info
- Debug
- Trace

Enqueue Wait Time (Sec)

If an SAP object is locked and inaccessible during an update by a mobile device, this parameter controls the number of seconds the underlying component should continue to attempt to access the locked object in intervals of 1 second. If accessing the locked object is still unsuccessful after the wait time, the update process is aborted.

Internal Conversion Exit Active

When checked, the framework runtime data manager performs standard SAP external-to-internal format conversion exit for all inbound BAPI parameters. This option is enabled by default. This setting should only be changed by the application developer, as it will have direct impact to the result of the mobile application.

External Conversion Exit Active

When enabled, the framework runtime data manager performs standard SAP internal-to-external format conversion exit for all outbound BAPI parameters. This option is enabled by default. This setting should only be changed by the application developer, as it will have direct impact to the result of the mobile application.

Range Parameter Check Active

When enabled, the framework runtime data manager will perform checks on all SAP range parameters of inbound BAPI parameters. The SAP range parameter has the structure of SIGN, OPTION, LOW and HIGH. Check routine will set the SIGN value to 'I' and the OPTION value to 'EQ' if not specified. This option is enabled by default. This setting should only be changed by the application developer, as it will have direct impact to the result of the mobile application.

Collection Mode

Collection mode determines how system statistic records are written to the database. Two modes are supported currently: synchronously and asynchronously. When Synchronously is selected, the statistics record is written to the database in real-time during BAPI calls. However, selecting this option incurs a performance penalty. Selecting Asynchronously means that statistics are collected in-memory and written asynchronously to the database at the end of the BAPI call.

Statistic Collection Active

When enabled, the framework records all runtime statistics associated with the BAPI calls between the middleware server and SAP. This collection provides data for the KPI statistics collections found in the Administration portal. This setting should only be changed by the application developer, as it will have direct impact to the result of the mobile application.

Created By, Creation Time Stamp, Last Changed By, Changed Time Stamp

The user ID and time stamps are automatically logged when a record is created or changed.

Mobile Application Configuration

Use the Mobile Application Configuration page to set general settings for the entire SAP mobile application.

Mobile Application - General

Use the General tab to create or change basic information about a mobile application.

Mobile Application - General Settings

Basic Data

- **Mobile Application:** The name of the mobile application, limited to 40 characters. This is a required field.
- **Description:** A brief, easy to understand description of the mobile application, limited to 60 characters. This is a required field.
- **Release:** The release number of the mobile application

User Management Setting

Disable Automatic User Creation: When checked, a new user GUID is not automatically created when a new mobile client is detected in the system. The system administrator must manually create and maintain mobile users through the Administration portal.

Server Management Setting

Disable Automatic Server Registration: When checked, a new server GUID is not automatically created when a new server is detected in the system. The system administrator must manually create and maintain servers through the Administration portal.

Multi Back End Setting

- **Multi Back End Enabled:** When checked, enables a specific mobile application to connect to multiple SAP systems, consisting of one host server and one or more satellite servers
- **System Role:** Drop-down menu listing Host or Satellite -
 - A host system is the connection between SAP and the Agentry application in the SAP Mobile Server. The host server provides the logic to the Agentry application and

functions as a bridge to the satellite server(s). There can only be one host server per system.

- Satellite servers communicate with SAP through the host server.

In order to complete multi-back end configuration, the host and satellite servers must be configured in the System Components tab. See the *Mobile Application - System Components* section for more details.

Administrative Info

- **Created By:** SAP user ID of the person who created the mobile data object
- **Creation Time Stamp:** Date and time of the creation of the mobile data object
- **Last Changed By:** SAP user ID of the person who last changed the mobile data object
- **Changed Time Stamp:** Date and time of the change to the mobile data object

Mobile Application - Mobile Status Setting

Use the Mobile Status Setting tab to map the available mobile statuses that a mobile data object (MDO) supports on the client side. If a user status also exists for the same MDO, you can link it to the mobile status and the system status through this tab.

Mobile Application - Mobile Status Setting

Mobile Application (Display Mode)

Create

General | **Mobile Status Setting** | Conversion Exit Setting | System Components | Parameters | Client Globals

Mobile Application Info

Mobile Application: Release:

Mobile App. Desc.:

Mobile Status Mapping

Mobile Status List

	Object Type	Mobile Status	System Status	User Status

Row 0 of 0

Mobile Status Detail

Object Type:

Mobile Status: Label On Mobile:

System Status: User Status:

Mobile Application Info

- **Mobile Application:** (Read Only) The name of the mobile application
- **Mobile Application Description:** (Read Only) A brief, easy to understand description of the mobile application
- **Release:** (Read Only) The release number of the mobile application

Mobile Status Mapping

- **Create button:** Click **Create** to create a new mobile status detail. Fill in the fields in the Mobile Status Detail section to automatically fill in the fields in this table.
- **Delete button:** Click **Delete** to delete an existing mobile status detail. To delete a mobile status detail, click the rectangle to the left of the Object Type column in the row you want to delete and click **Delete**.
- **Object Type:** (Read Only) Object type from the Mobile Status Detail section
- **Mobile Status:** (Read Only) Mobile status from the Mobile Status Detail section

- **System Status:** (Read Only) System status from the Mobile Status Detail section
- **User Status:** (Read Only) User status from the Mobile Status Detail section

Mobile Status Detail

- **Object Type:** The specific object in a mobile application, i.e., NOTIFICATION
- **Mobile Status:** Status defined by the mobile application
- **Label on Mobile:** Not used
- **System Status:** Standard SAP status code
- **User Status:** SAP user status code as defined in SAP

Mobile Application - Conversion Exit Setting

Use the Conversion Exit Setting tab to list the SAP conversion exits to exclude during runtime by the framework.

Mobile Application - Conversion Exit Setting

Mobile Application (Display Mode)

Create Copy Delete Change

General Mobile Status Setting **Conversion Exit Setting** System Components Parameters

Mobile Application Info

Mobile Application: SAP_WORK_MANAGER_60 Release: 6.0.0

Mobile App. Desc.: SAP Work Manager 6.0.0 with LAM

Conversion Exit Exclusion

Conversion Exit List

Conversion Exit	Active Flag	Skip Conversion	Skip On Initial	Setting Scope
AOBAR	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	All Conversion Exit
EXCRT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	All Conversion Exit
TSTPS	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	All Conversion Exit

Conversion Exit Detail

Conversion Exit: AOBAR

Setting Scope: All Conversion Exit Skip Conversion: ☐ Skip On Initial: ☒

Setting Enabled: ☒

Mobile Application Info

- **Mobile Application:** (Read Only) The name of the mobile application
- **Mobile App(lication) Desc(ription):** (Read Only) A brief, easy to understand description of the mobile application
- **Release:** (Read Only) The release number of the mobile application

Conversion Exit List

- **Add button:** Click **Add** to create a new conversion exit detail. Fill in the fields in the Conversion Exit Detail section to automatically fill in the fields in this table.
- **Delete button:** Click **Delete** to delete an existing conversion exit detail. To delete a conversion exit detail, click the rectangle to the left of the Conversion Exit column in the row you want to delete and click **Delete**.
- **Conversion Exit:** (Read Only) Conversion exit from the Conversion Exit Detail section.
- **Active Flag (column 1):** (Read Only) When checked, the Skip Conversion box is checked in the Conversion Exit Detail section
- **Skip Conversion (column 2):** (Read Only) When checked, the Skip on Initial box is checked in the Conversion Exit Detail section
- **Skip On Initial (column 3):** (Read Only) When checked, the Setting Scope box is checked in the Conversion Exit Detail section
- **Setting Scope:** (Read Only) Conversion exit scope from the Conversion Exit Detail section

Conversion Exit Detail

- **Conversion Exit:** Name of the conversion exit as found in SAP
- **Setting Scope:** Choose from the drop down menu choices:
 - All Conversion Exit - Both input and output conversion exit routines are excluded
 - Input Conversion Exit - Conversion routines are exited when data is sent to SAP
 - Output Conversion Exit - Conversion routines are exited when data is pulled out of SAP
- **Skip Conversion:** When checked, the conversion routine is always exited. When you check this box, it automatically checks the box in the first Active Flag column in the table above.
- **Skip on Initial:** When checked, the conversion routine is only excluded when the initial field does not contain a value. If the initial field contains any value, the conversion routine runs. When you check this box, it automatically checks the box in the second Active Flag column in the table above.
- **Setting Enabled:** When checked, the settings configured for the conversion exit are enabled and the exit is active. When you check this box, it automatically checks the box in the third Active Flag column in the table above.

Mobile Application - System Components

Use this tab to define system components in a multi-back end system. Configuration in this tab is not necessary if the application does not require a multi-back end system.

Note: You must check the Multi Back End Enabled box in the General tab of the Mobile Application Configuration pane in order for system component configuration to function.

Mobile Application - System Components

The screenshot shows the 'Mobile Application (Display Mode)' window with the 'System Components' tab selected. The window has a top bar with 'Create', 'Copy', 'Delete', and 'Change' buttons. Below the tabs, the 'Mobile Application Info' section shows 'Mobile Application: SAP_SALES_MANAGER_25', 'Release: 2.5', 'Mobile App. Desc.: SAP CRM Sales 2.5', and 'Multi Backend Enabled: ☒'. The 'Mutil Backend System Component' section contains a 'System Component List' table. The table has columns: System Component, System Role, RFC Destination, Active Flag, and Component Mobile App. It lists two components: 'ERP_SYSTEM' (Satellite, SSLM25_ERP_SD_HOST, Active Flag checked, Component Mobile App: SAP_SALES_MANAGER_25) and 'LOCAL_HOST' (Host, Active Flag checked). Below the table is the 'System Component Detail' section with fields for System Component (ERP_SYSTEM), RFC Destination (SSLM25_ERP_SD_HOST), Host, Client, System Role (Satellite), Component Mobile App (SAP_SALES_MANAGER_25), Active Flag (checked), Created By (JWANG), Last Changed By (TEJ), Creation Time Stamp (31.10.2012 15:17:33), and Changed Time Stamp (28.01.2013 21:02:54).

System Component	System Role	RFC Destination	Active Flag	Component Mobile App.
ERP_SYSTEM	Satellite	SSLM25_ERP_SD_HOST	<input checked="" type="checkbox"/>	SAP_SALES_MANAGER_25
LOCAL_HOST	Host		<input checked="" type="checkbox"/>	

Mobile Application Info

- **Mobile Application:** (Read Only) The name of the mobile application
- **Release:** (Read Only) The release number of the mobile application
- **Mobile App(lication) Desc(ription):** (Read Only) A brief, easy to understand description of the mobile application
- **Multi Back End Enabled:** (Read Only) When checked, the multi-backend was activated in the General tab

System Component List

- **Add button:** Click **Add** to create a new system component detail. Fill in the fields in the System Component Detail section to automatically fill in the fields in this table.

- **Delete button:** Click **Delete** to delete an existing system component detail. To delete a system component detail, click the rectangle to the left of the System Component column in the row you wish to delete and click **Delete**.
- **System Component:** (Read Only) System component from the System Component Detail section
- **System Role:** (Read Only) System role from the System Component Detail section
- **RFC Destination:** (Read Only) RFC destination from the System Component Detail section
- **Active Flag:** (Read Only) When checked, the Active Flag box is checked in the System Component Detail section
- **Component Mobile App:** In multi-back end scenarios, when different back end names are used, this is the application name that can virtually tie all applications together

System Component Detail

- **System Component:** Descriptive name of the component. This is a required field.
- **RFC Destination:** Must be defined in SAP prior to configuration in Agentry SAP Framework. Use transaction code SM59 in SAP to create or change the RFC destination.
- **Host:** (Read Only) Identifying host name, defined in SAP
- **System Number:** (Read Only) Identifying server number, defined in SAP
- **Client:** (Read Only) Number of the client that the system component connects to, defined in SAP
- **System Role:** Determines if the system component is a host or a satellite. There can only be one host per multi-back end system.
- **Active Flag:** When checked, the system component is activated in the multi-back end system
- **Component Mobile App:** Common application name for multi-back end systems
- **Created By:** SAP user ID of the person who created the mobile data object
- **Creation Time Stamp:** Date and time of the creation of the mobile data object
- **Last Changed By:** SAP user ID of the person who last changed the mobile data object
- **Changed Time Stamp:** Date and time of the change to the mobile data object

Mobile Application - Parameters

Use this tab to define system parameters.

Mobile Application - Parameters

Mobile Application (Display Mode)

Create Copy Delete Change

General Mobile Status Setting Conversion Exit Setting System Components Parameters Client Globals

Mobile Application Info

Mobile Application: SAP_SALES_MANAGER_25 Release: 2.5

Mobile App. Desc.: SAP CRM Sales 2.5

Application Parameters

Parameter List

Import/Export Search

Parameter Group	Param. Name	Param. Value	Par	Act	No I	Cor
TABLE_CHECK	CTDuration	168		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	TABL
STEPHANDLER	RelatedObjectNavigationStepHandler	com.syclo.sap.salesmanager.stephandler.RelatedObjectNavigatio		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SalesOrderAddStepHandler	com.syclo.sap.component.salesorder.stephandler.SalesOrderAdi		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SalesOrderEditStepHandler	com.syclo.sap.component.salesorder.stephandler.SalesOrderEdi		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SalesOrderFetchStepHandler	com.syclo.sap.component.salesorder.stephandler.SalesOrderFet		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SalesOrderRemoveStepHandler	com.syclo.sap.component.salesorder.stephandler.SalesOrderRer		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SalesOrderSearchStepHandler	com.syclo.sap.component.salesorder.stephandler.SalesOrderSe		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SalesOrderUpdateStatusStepHandler	com.syclo.sap.component.salesorder.stephandler.SalesOrderUp		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SettingsCalendarActivityFetchStepHandler	com.syclo.sap.salesmanager.stephandler.SettingsCalendarActivi		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP
STEPHANDLER	SurveyAddStepHandler	com.syclo.sap.component.survey.stephandler.SurveyAddStepHa		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	STEP

Parameter Detail

Parameter Group: TABLE_CHECK

Param. Name: CTDuration Param. Scope: Application

Param. Value: 168

Rule Id: Use Rule: ☐

Rule Input Param: Comment: TABLE_CHECK define Frequency of how often to check a complex table. Param. Name should be the name of the complex table.

Active Flag: ☒ No Runtime Change: ☒

Mobile Application Info

- **Mobile Application:** (Read Only) The name of the mobile application
- **Release:** (Read Only) The release number of the mobile application
- **Mobile Application Description:** (Read Only) A brief, easy to understand description of the mobile application

Parameter List

Note: The columns in the Parameter List table are read-only. Use the Parameter Detail section to make any additions or edits to the table.

- **Add button:** Click **Add** to create a new parameter detail. Fill in the fields in the Parameter Detail section to automatically fill in the fields in this table.
- **Delete button:** Click **Delete** to delete an existing parameter detail. To delete a parameter detail, click the rectangle to the left of the Parameter Group column in the row you want to delete and click **Delete**.
- **Parameter Group:** Parameter group from the Parameter Detail section

- **Param. Name:** Parameter name from the Parameter Detail section
- **Param. Value:** Parameter detail from the Parameter Detail section
- **Param. Scope:** Parameter scope from the Parameter Detail section
- **Active Flag:** When checked, the Active Flag box is checked in the Parameter Detail section.
- **No Runtime Change:** When checked, the No Runtime Change box is checked in the Parameter Detail section
- **Comment:** Comments from the Parameter Detail section

Parameter Detail

- **Parameter Group:** The group to which the parameter belongs. Groups are a means of organizing parameters. References to a parameter include both the group name and the parameter name.
- **Param. Name:** The unique name of the parameter
- **Param. Scope:** The scope of the parameter value. There are two options:
 - Mobile Application: Value for all users of the application
 - Mobile User: Value that can be overridden for individual users. To override a user's parameter value, see the Administration & Monitoring portal information on parameters.
- **Param. Value:** The currently configured value of the parameter. References to this parameter will return this value
- **Rule Id:** If enabled, this is the rule to be used at run time
- **Use Rule:** When checked, you can define a rule to be used at run time
- **Rule Input Param:** If the specified rule has optional parameters, define them here
- **Active Flag:** When checked, the parameter is used by the mobile application. Inactive parameters are not available to the mobile application.
- **No Runtime Change:** When checked, the value of the parameter cannot be overridden. The configured value is always the value. If not checked, parameter values can be overridden at runtime through synchronization processing.
- **Comment:** Any comments applicable to the parameter that describe its purpose or value. This has no effect on the parameter's behavior and is provided for reference purposes only.

Mobile Application - Client Globals

Use this tab to define client globals.

Mobile Application - Client Globals

Mobile Application (Display Mode)

Create Copy Delete Change

General Mobile Status Setting Conversion Exit Setting System Components Parameters **Client Globals**

Mobile Application Info

Mobile Application: SAP_SALES_MANAGER_25 Release: 2.5

Mobile App. Desc.: SAP CRM Sales 2.5

Client Globals

Client Global List

Import/Export Search

Global Group	Global Name	Global Value	Global Scope	Active Flag	No Runtime Change	Comment
ACTIVITY_IMG	5	Phone	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	6	Calendar	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	CFU	FollowUp	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	CP1	Calendar	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	CP2	Calendar	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	CX1	Calendar	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	CX2	Tasks	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACTIVITY_IMG	006	Calendar	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ACTIVITY_IMG defines
ACCOUNT_NAVIGATION	1	Sales Area	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	NAVIGATION defines
ACCOUNT_NAVIGATION	2	Relationships	Application	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	NAVIGATION defines

Client Global Detail

Client Global Group: ACTIVITY_IMG

Client Global Name: 5 Global Scope: Application

Client Global Value: Phone

Rule Id: Use Rule: ☐

Rule Input Param:

Comment: ACTIVITY_IMG defines the image Name to use for each Activity Category in CTActivity complex table.

Active Flag: ☒

No Runtime Change: ☒

Mobile Application Info

- **Mobile Application:** (Read Only) The name of the mobile application
- **Release:** (Read Only) The release number of the mobile application
- **Mobile Application Description:** (Read Only) A brief, easy to understand description of the mobile application

Client Global List

Note: The columns in the Client Global List table are read only. Use the Client Global Detail section to make any additions or edits to the table.

- **Add button:** Click **Add** to create a new global. Fill in the fields in the Client Global Detail section to automatically fill in the fields in this table.
- **Delete button:** Click **Delete** to delete an existing global. To delete a global, press the rectangle to the left of the Global Group column in the row you wish to delete and click **Delete**.
- **Global Group:** Global group from the Client Global Detail section

- **Global Name:** Client global group name from the Client Global Detail section
- **Global Value:** Client global value from the Client Global Detail section
- **Global Scope:** Global scope from the Client Global Detail section
- **Active Flag:** When checked, the Active Flag box is checked in the Client Global Detail section
- **No Runtime Change:** When checked, the No Runtime Change box is checked in the Client Global Detail section
- **Comment:** Comments from the Client Global Detail section

Client Global Detail

- **Client Global Group:** The group to which the global belongs. Groups are a means of organizing globals. References to a global include both the group name and the global name.
- **Client Global Name:** The unique name for the global
- **Global Scope:** The scope of the global value. There are two options:
 - Mobile Application: Value for all users of the application
 - Mobile User: Value that can be overridden for individual users.
- **Client Global Value:** The currently configured value of the global. References to the global return this value.
- **Rule ID:** Name, or ID, of the ABAP or class
- **Use Rule:** When checked, the rule listed in the Rule ID field is active. If this value is active, then the Client Global Value field is not used.
- **Rule Input Param:** Parameters to use with the rule. Examples include a key value pair, a user parameter, or a table.
- **Comment:** Displays any comments added to the global to describe its purpose or current value. This has no effect on the global's behavior and is provided for reference purposes only.
- **Active Flag:** When checked, the client global is activated in the system. Inactive globals are not available to the mobile application.
- **No Runtime Change:** When checked, the value of the global cannot be overridden. The configured value in the Configuration portal will always be the value. Globals without this setting can be overridden at runtime through synchronization processing.

Push Scenario Definition

A push scenario is the data that the SAP Agentry Server can push to mobile application users. For example, an Emergency Service order is pushed down to the SAP Client of a single user or multiple users. The Push Scenario Definition panel provides an interface to configure what data is pushed and the triggers that initiate a push.

Pushing data from SAP to the mobile client using the ABAP Add-On push framework consists of two key steps:

1. Push relevant data change is detected in SAP and a push instance is registered with the push registry.
2. The new push instance entry in the push registry is processed by a system program (push processor). The data content to be pushed and the recipients for the push instance are determined. For each recipient, a message is generated in his/her out box of the outbound message queue, and waits for pick up.

Push Scenario Definition - General Data

Use the General Data tab of the Push Scenario Detail screen to modify data for a push scenario. You can define source, subscriber, notification, and activation settings.

Push Scenario Definition - General Data Tab

Push Scenario Detail (Display Mode)

Create Copy Delete Change

General Data Event Setting Outbound Trigger Subscription Settings

Basic Data

Scenario Id: SWM60_MRS_WORKORDER_PUSH Alias: SWM60_EMERGENCY_WORKORDER_OPERAT

Mobile Application: SAP Work Manager 6.0.0 with LAM

Source Setting

Source Type: Exchange Object

Source Object: SWM60_WORK_ORDER_OPERATION_PL

Source Handler: /SYCLO/CL_PM_AUFPL_EX_HNDLR

Distribution Setting

Distribution Type: Mobile data object

Distribution Object: SWM60_WORKORDER_PUSH

Distribution Handler: /SMERP/CL_PM_PUSHWORKORDER_DO

Subscriber Setting

Subscriber Type: All mobile users Validity (Hr): 8

Priority (0 - Highest, 9 - Lowest): 0

Disable Owner Originated Push: ☐ Check Mobile Transaction History: ☐ History Interval (Seconds): 0

Notification Setting

Email Notification: ☐ No Data Package: ☒

Email Subject: Syclo emergency order assignment notification

Email Message (140 chars): An emergency order &OBJKEY_REF& has been assigned to you. Please update your mobile device.

Activation

Active Flag: ☒ Enable Push History: ☒ Require Metadata: ☐ Enable Fetch Callback: ☒

Administrative Info

Created By: YALAMATIS Creation Time Stamp: 09/14/2013 00:15:11

Last Changed By: POPEM Changed Time Stamp: 10/10/2013 16:58:58

Basic Data

- **Scenario ID:** Name of the push scenario
- **Alias:** Alias of the push scenario
- **Mobile Application:** Application to which the push scenario belongs

Source Setting

- **Source Type:** Type of source object associated with the push scenario
- **Source Object:** Drop-down list containing the available source objects for the push scenario

Note: In order for an exchange object to be listed in the drop-down menu, the Push Relevant box must be checked in the Push Settings tab of the Push Scenario Definition screen.

- **Source Handler:** Class handler associated with the source object for the push scenario. This is a non-editable field.

Distribution Setting

- **Distribution Type:** Type of distributed object associated with the push scenario
- **Distribution Object:** Name of the specific object associated with the push scenario, chosen by a drop-down list
- **Distribution Handler:** Name of the class handler from the class repository that is responsible for updating the exchange table. This field is automatically filled when choosing the source object and is not editable.

Subscriber Setting

- **Subscriber Type:** Drop-down list to choose if the push is sent to all users, users with active connections, or users defined in a scenario subscriber list.
- **Validity (Hr):** Amount of time, in hours, of the validity of the data to be pushed to clients. When the time limit has passed, the data within the push scenario is no longer valid and will not be pushed to any more mobile applications.
- **Priority:** The priority assigned to the push, with the default set to 0. The higher the priority setting, the higher the push is in the push queue. For instance, a push priority set to 0 is processed before a different push with a priority set to 5. For push instances with the same set priority or default priority, the pushes are processed in the order in which they were created.

Notification Setting

- **Email Notification:** When checked, sends an email to all users affected by the push scenario.

Note: User set up for email notification must be performed in the System Administration and Monitoring portal in the Administration - User Management panel before email notification is performed.

- **No Data Package:** When checked, the data package, or push information, is not sent to the mobile device when the email notification is sent. A user must connect to the system and perform a regular fetch in order to retrieve the push information. In this way, users do not receive outdated push information if they are seldom actively connected to the system.
- **Email Subject:** Subject, or header, of the email message
- **Email Message (140 chars):** Body of the email message, limited to 140 characters. The limit is to support short messages to websites such as Twitter.
A special variable, **&OBJKEY&**, is available for use in the body of the email message. This variable is substituted for the actual object identifier content during runtime and presented on the mobile Client.

Activation

- **Active Flag:** When checked, the push scenario is in an active state. If unchecked, the push scenario is not performed.
- **Enable Push History:** When checked, the push history table in SAP is populated. The push history table contains a list of users and the object keys pushed to those users.
- **Require Metadata:** The metadata table is only populated for the push transaction when this option is checked. When unchecked, the default, the metadata table is not populated, saving Server resources.
- **Enable Fetch Callback:** Fetch callback is used to augment data, in order to make it a two-step process. If implemented, an extra callback routine is invoked when Agentry is retrieving push messages for SAP. Note that you must enable the logic on the back end as well, before fetch callback will be possible. When information changes in SAP, a push is triggered and the distribution agent writes persisting information into the queue. This information is written to the database, which historically has created overhead to the system. The fetch callback avoids the overhead by not performing calculations in the queue; rather, it waits for Agentry to get the push. At that point, calculations for the fetch occur. In this way, the overhead of writing to the database is eliminated. An advantage to implementing fetch callback is that the copy is always fresh, as it is on demand. This eliminates obsolete copies. However, the disadvantage of using fetch callback is that calculations occur every time Agentry demands it.

Administrative Info

- **Created By:** SAP user ID of the person who created the push scenario
- **Creation Time Stamp:** Date and time of the creation of the push scenario
- **Last Changed By:** SAP user ID of the person who last changed the push scenario
- **Changed Time Stamp:** Date and time of the change to the push scenario

Example

The following sample screen shows that a push is enabled for all the transaction types defined in the list. For example, if a sales order is created and/or updated in the SAP back end, the changes for the transaction are pushed to the mobile client to the appropriate user without any transmit or trigger from the mobile client.

The screenshot displays the 'Push Scenario Detail (Display Mode)' configuration screen. On the left, a tree view under 'Push Scenarios By Mobile App' shows 'SAP_SALES_MANAGER_25' expanded, with 'SSLM25_SALES_ORDER_PUSH' selected. The main area shows the configuration for this scenario. The 'General Data' tab is active, displaying fields for 'Scenario ID' (SSLM25_SALES_ORDER_PUSH), 'Alias' (empty), 'Mobile Application' (SAP CRM Sales 2.5), 'Source Setting' (Source Type: Exchange Object, Source Object: SSLM25_SALES_ORDER_PUSH, Source Handler: /SYCLOI_CRMCTX_BUS115_EX_H), 'Distribution Setting' (Distribution Type: Mobile data object, Distribution Object: SSLM25_SALES_ORDER_PUSH, Distribution Handler: /SYCLOI_CRMCTX_PUSHSAORD_DO), 'Subscriber Setting' (Subscriber Type: All mobile users, Priority: 6, Validity: 8, Disable Owner Originated Push: unchecked, Check Mobile Transaction History: unchecked, History Interval: 0), 'Notification Setting' (Email Notification: unchecked, No Data Package: checked, Email Subject: Cycle Lead assignment notification, Email Message: Sales Order &OBJKEYS has been assigned to you. Please update your mobile device.), 'Activation' (Active Flag: checked, Enable Push History: unchecked, Require Metadata: unchecked, Enable Fetch Callback: checked), and 'Administrative Info' (Created By: JIWANG, Last Changed By: SYALABA, Creation Time Stamp: 31.10.2012 15:17:58, Changed Time Stamp: 14.02.2013 17:57:45).

Push Scenario Definition - Event Setting

Use the Event Setting tab of the Push Scenario Detail screen to control how the push processing is triggered for specific events when data is changed in SAP and triggers a push. Event settings can be set using variables or by assigning specific events or queues to initiate the push process.

You can initiate push processing either via an SAP background event or via an SAP qRFC. Each of these options is described in the appropriate section following the “Push Scenario Definition - Event Setting Tab” screen shot.

Push Scenario Definition - Event Setting Tab

Push Scenario Detail (Display Mode)

Create Copy Delete Change

General Data **Event Setting** Outbound Trigger Subscription Settings

Basic Data

Scenario Id: SWM60_MRS_WORKORDER_PUSH

Mobile Application: SAP Work Manager 6.0.0 with LAM

Background Event Setting Detail

Disable Background Event Trigger: ☒

Standard Event Setting

Event Id:

Event Parameter:

Rule Based Event Setting

Push Event Rule:

qRFC Setting Detail

Enable qRFC Processing: ☒

Queue Setting

Queue Name: SWM60_EM_&OBJKEY_REF&

qRFC Rule: /SYCLO/CL_CORE_QRFC_ROUTINE : Standard routine for qRFC queue determination

Runtime Parameters

Allow Instance Merge: ☒ Exclude Status SRV_COMP: ☒

Maximum Select Delay (Seconds): 2 Select Retry: 1

Basic Data

- **Scenario ID:** Name of the push scenario
- **Mobile Application:** Application to which the push scenario belongs

Background Event Setting Detail

Complete this section to enable push processing via an SAP background event.

If you enable this option, an SAP background event is raised after a push instance is registered with the push registry. The event can subsequently start an SAP background job that is subscribed to the background event. The started SAP background job can process new push instances in the push registry.

- **Disable Background Event Trigger:** Uncheck this box when using the Background Event Setting Detail fields. When checked, the push process is triggered by the

information configured here, rather than by the background event and no background event is raised after the push instance is registered with the push registry.

- **Event ID:** Either a static or variable event ID to be raised. For information on variables, hover over the field to read the tool tip that appears. Event IDs are used to determine when to raise the event.
- **Event Parameter:** A free-text field to configure parameters associated with the Event ID when raised. Parameters can be static or use variables. For information on variables, hover over the field to read the tool tip that appears. Typical parameters can be push scenario IDs, etc.
- **Push Event Rule:** This is a special ABAP class routine that returns the Event ID and event parameters programmatically. A standard routine is provided, but a custom routine can be developed by the customer if there is a business need.

Special variables can be used when defining the Event ID. Special variables are substituted at runtime. If special variables are used, the push event rule must be defined in order for the substitution to occur. Special variables include the following:

- &OBJKEY&
- &OBJKEY_REF&
- &MOBILE_APP&
- &INSTANCE_GUID&
- &SCENARIO_ID&
- &SOURCE_OBJECT&
- &SOURCE_TYPE&
- &DATUM&
- &UZEIT&
- &UNAME&
- &HOST&

Monitoring SAP background event trigger by push instances

To monitor the SAP push, use the following tools:

- Go to the **Administration & Monitoring Portal -> Monitoring -> Push Instance Monitor** and search for push instances using proper search filters. Verify the Event ID and the event parameter of the push instance.
- Use the transaction code **SM37** to search and verify that SAP background jobs are being triggered properly by the Event ID and the event parameters raised during the push process.

qRFC Setting Detail

Complete this section to enable push processing via SAP qRFC.

If you enable this option, after a push instance is registered with the push registry, a new entry is added to the SAP qRFC queue for the specific push instance. The qRFC queue entry is

processed automatically by the SAP system. When the qRFC queue is processed, the specific push instance is processed by the push processor.

- **Enable qRFC Processing:** When checked, enables initiating the push process through the qRFC queue.
- **Queue Name:** Either a static or a variable qRFC queue name.
- **qRFC Rule:** This is a special ABAP class routine that returns the qRFC queue name programmatically. A standard routine is provided, but a custom routine can be developed by the customer if there is a business need.
- **Allow Instance Merge:** When checked, if multiple processes are triggered on the same SAP object, the instances are merged to save processing time.
- **Exclude Status SRV_COMP:** When checked, push instances with SRV_COMP status are not reprocessed.
- **Maximum Select Delay:** This is the number of seconds to wait before reading from the push registry DB table. This is only needed for slow systems. This is typically set to 1 or 0. If the system takes longer to write to the database, set to higher than 1.
- **Select Retry:** Number of times to retry the select from DB table is nothing is found. This is only needed for slow systems.

Special variables can be used when defining the queue name. Special variables are substituted at runtime. If special variables are used, the push event rule must be defined in order for the substitution to occur. Special variables include the following:

- &OBJKEY&
- &OBJKEY_REF&
- &MOBILE_APP&
- &INSTANCE_GUID&
- &SCENARIO_ID&
- &SOURCE_OBJECT&
- &SOURCE_TYPE&
- &DATUM&
- &UZEIT&
- &UNAME&
- &HOST&

Monitoring push instance processing via SAP qRFC

To monitor the SAP push, use the following tools:

- Go to the **Administration & Monitoring Portal -> Monitoring -> Push Instance Monitor** and search for push instances using proper search filters. Verify the qRFC queue name of the push instance.

- Use the transaction code **SMQ1** to search and verify that there are not outstanding entries waiting in the qRFC queue.

Example

The following sample screen shows that a push is enabled via the SAP qRFC queue.

Push Scenario Detail (Display Mode)

Create Copy Delete Change

General Data **Event Setting** Outbound Trigger Subscription Settings

Basic Data

Scenario Id: SWM60_EMERGENCY_WORKORDER_PUSH
Mobile Application: SAP Work Manager 6.0.0 with LAM

Background Event Setting Detail

Disable Background Event Trigger: ☒

Standard Event Setting

Event Id:
Event Parameter:

Rule Based Event Setting

Push Event Rule:

qRFC Setting Detail

Enable qRFC Processing: ☒

Queue Setting

Queue Name: SWM60_EM_&OBJKEY&
qRFC Rule: /SYCLO/CL_CORE_QRFC_ROUTINE : Standard routine for qRFC queue determination

Runtime Parameters

Allow Instance Merge: ☒ Exclude Status SRV_COMP: ☒
Maximum Select Delay (Seconds): Select Retry:

Push Scenario Definition - Outbound Trigger

Use the Outbound Trigger tab of the Push Scenario Detail screen to assign one or more outbound triggers to the push scenario. After push generation, the outbound trigger should be enabled to notify the Agentry application in the SAP Mobile Server.

Push Scenario Definition - Outbound Trigger Tab

Push Scenario Detail (Display Mode)

Create Copy Delete Change

General Data Event Setting **Outbound Trigger** Subscription Settings

Basic Data

Scenario Id: SWM60_EMERGENCY_WORKORDER_PUSH

Mobile Application: SAP Work Manager 6.0.0 with LAM

Outbound Trigger Setting Detail

Enable Outbound Trigger: ☒

Use Single Instance Processing: ☒

Data Fetch Retry Wait (Seconds): 60

Outbound Triggers Assigned

Seq. No.	Outbound Trigger Id	Active
00001	HTTP SWM60_HTTP_PUSH_TRIGGER : /SYCLO/CL_CORE_OTRIG_STD_PUSH	<input checked="" type="checkbox"/>

Basic Data

- **Scenario ID:** Name of the push scenario
- **Mobile Application:** Application to which the push scenario belongs

Outbound Trigger Setting Detail

- **Enable Outbound Trigger:** When checked, allows all “active” outbound triggers to process.
- **Use Single Instance Processing:** When checked, the system sends each outbound trigger action as a separate XML file. This is usually reserved for test mode. In most instances, you will want to leave this option unchecked to initiate batch processing rather than single instance processing.
- **Data Fetch Retry Wait (Seconds):** Setting a data fetch retry wait reduces the hits to the Server in cases of Server miscommunication.

Outbound Trigger Setting Detail

This table shows all outbound triggers assigned to this push scenario with the following three fields:

- **Seq. No.:** When there are multiple triggers for the push scenario, the sequence number defines the order in which the triggers are processed.
- **Outbound Trigger ID:** The identification number of the outbound trigger
- **Active:** Indicates whether or not the outbound trigger is active. Note: you must use the **Enable Outbound Trigger** checkbox to enable processing for active triggers.

Push Scenario Definition - Outbound Trigger Tab in Change Mode

Push Scenario Detail (Change Mode)

Save Cancel

General Data Event Setting **Outbound Trigger**

Basic Data

Scenario Id: * SWM51_EMERGENCY_WORKORDER_OPERAT

Mobile Application: * Syclo SMART Mobile Suite - Work Manager 5.1

Outbound Trigger Setting Detail

Enable Outbound Trigger: ☐

Use Single Instance Processing: ☐

Data Fetch Retry Wait (Seconds): 0

Outbound Triggers Assigned

Add Trigger Remove Trigger Move Up Move Down

Seq. No.	Outbound Trigger Id	Active
00001		<input type="checkbox"/>

Row 1 of 1

When in Change Mode, use the action buttons that display to add or remove triggers to the push scenario or to change the sequence of the selected triggers.

Example

The following sample screen shows that an outbound trigger is enabled for Sales Order. For example, if a sales order is created and/or updated in the SAP back end, notifications of the changes for the transaction are sent to the Agentry application.

The screenshot displays the 'Push Scenario Detail (Display Mode)' window. On the left, a tree view shows 'Push Scenarios By Mobile App' with 'SAP_SALES_MANAGER_25' expanded, listing several scenarios including 'SSLM25_SALES_ORDER_PUSH'. The main panel has tabs for 'General Data', 'Event Setting', 'Outbound Trigger', and 'Subscription Settings'. The 'Outbound Trigger' tab is selected, showing 'Basic Data' with 'Scenario Id' as 'SSLM25_SALES_ORDER_PUSH' and 'Mobile Application' as 'SAP CRM Sales 2.5'. Below this, 'Outbound Trigger Setting Detail' shows 'Enable Outbound Trigger' and 'Use Single Instance Processing' both checked, and 'Data Fetch Retry Wait (Seconds)' set to 60. At the bottom, 'Outbound Triggers Assigned' contains a table with one entry.

Seq. No.	Outbound Trigger Id	Active
00001	HTTP SSLM25_HTTP_PUSH_TRIGGER : /SYCLO/CL_CORE_OTRIG_STD_PUSH	<input checked="" type="checkbox"/>

Push Scenario Definitions - Subscription Settings

Use this tab to define the push scenarios allowed through subscription. Enable the subscription settings when the push is based on a subscription-based push or receives OnDemand requests from the mobile client.

Subscription Settings

The screenshot shows the 'Push Scenario Detail (Display Mode)' window. At the top, there are buttons for 'Create', 'Copy', and 'Delete'. Below these are four tabs: 'General Data', 'Event Setting', 'Outbound Trigger', and 'Subscription Settings'. The 'Subscription Settings' tab is currently selected. Inside this tab, there are two main sections: 'Basic Data' and 'Subscription Agent Settings'. The 'Basic Data' section contains four input fields: 'Scenario Id', 'Mobile Application', 'Source Type', and 'Source Object'. The 'Source Handler' field is located to the right of the 'Source Object' field. The 'Subscription Agent Settings' section contains two fields: 'Allow Subscription' (a checkbox) and 'Subscription Agent Id' (a text input field).

Once you select one of the available push scenarios from the list in the left panel, the Basic Data section is automatically populated.

Subscription Agent Settings

To allow subscriptions to the push scenario, check the **Allow Subscription** check box and enter a **Subscription Agent ID**.

Note: For subscriptions to work, you must also select the Source Type **Client On Demand Request**, which appropriately changes the Subscriber Type. You must also complete the information on the Subscription Agent Definition screen.

The subscription agent is a simple Yes/No gatekeeper. It either accepts or does not accept the push request based on logic in the back end, or based on quota requests. If the subscription agent approves the push request, it generates a push instance. The subscription agent then puts a subscription request into the Subscription Request table. The table dictates which user requests which request key. After this, an instance is generated as an OnDemand request, with a reference to the Subscription Request table. The push is then processed as a normal push, through the push channel, rather than through the fetch channel.

OnDemand Push

With an OnDemand push, the client receives information through the push channel, rather than the fetch channel. In this way, the user can still work on the device, even while the push is adding data to the device. As an example, a user can request a PDF document to be added to the device, and requests it. An OnDemand push is used to retrieve that PDF from the back end and add it to the device through the push channel. Use the **Source Type** field to configure which type of push is required for the chosen subscription setting.

Subscription-Based Push

A subscription-based push uses the same concept as an OnDemand push. When there is a change in the back end, the push will figure out who subscribes to the data being pushed and then push that data out to the users, to their client devices. Use the **Source Type** field to configure which type of push is required for the chosen subscription setting.

Example

The following sample screen shows that OnDemand push is enabled for FactSheet generation push. To allow subscriptions to the push scenario, check the **Allow Subscriptions** check box and enter a Subscription Agent ID.

Subscription Agent Definition

The Subscription Agent Definition page allows you to define how subscription requests for back end System data are handled. On Demand subscriptions allow you to define push options. For instance, some accounts or activities have attachments which are not automatically pushed down. If subscribed, however, a Sales Manager can get those attachments On Demand, based on the settings defined on this page.

The screenshot displays the 'Subscription Agent Detail (Display Mode)' window in the SAP Mobile Platform configuration tool. The interface is divided into several sections: 'General Data', 'Agent Handler Info', 'Authorization Setting', 'Activation', and 'Administrative Info'. The 'Subscription Agent Id' is 'SSLM25_CLIENT_ONDEMAND_PUSHREQ', and the 'Mobile Application' is 'SAP CRM Sales 2.5'. The 'Agent Handler Info' section includes fields for 'Subscription Agent Handler', 'Maximum Delivery', 'Wait Interval (Second)', 'Open Subscription Quota', and 'Default Validity (Hr)'. The 'Authorization Setting' section has fields for 'Required User Role for Requestor' and 'Required User Role for Subscriber'. The 'Activation' section has an 'Active Flag' checkbox. The 'Administrative Info' section shows 'Created By', 'Last Changed By', 'Creation Time Stamp', and 'Changed Time Stamp'.

Section	Field	Value
General Data	Subscription Agent Id	SSLM25_CLIENT_ONDEMAND_PUSHREQ
	Subscription Agent Desc.	SSLM 2.5 - Client Ondemand Push Request
	Mobile Application	SAP CRM Sales 2.5
Agent Handler Info	Subscription Agent Handler	/SYCLO/CL_CORE_SUB_AGENT_BASE : Subscription agent base class
	Maximum Delivery	0
	Wait Interval (Second)	10
	Open Subscription Quota	100
	Default Validity (Hr)	1
	No Push Request	<input type="checkbox"/>
Authorization Setting	Required User Role for Requestor	
	Required User Role for Subscriber	
Activation	Active Flag	<input checked="" type="checkbox"/>
Administrative Info	Created By	JWANG
	Creation Time Stamp	17.12.2012 18:27:09
	Last Changed By	JWANG
	Changed Time Stamp	17.12.2012 18:27:09

Basic Data

- Subscription Agent ID: the name of the on demand push requirement subscription
- Subscription Agent Description:** text description of the agent ID
- Mobile Application:** the most current valid application version

Agent Handler Info

- Subscription Agent Handler:** the handler name and location for this subscription agent
- Maximum Delivery:** maximum number of items that will be delivered through the On Demand request
- Wait Interval (Second):** this is the time, in seconds, to wait between pushes
- Open Subscription Quota:** Enter a non-zero value to limit the number of open subscription requests in the system. If this field has a value, no new subscription requests are accepted once this quota is reached.
- Default Validity (Hr):** the amount of time, in hours, that the On Demand request is valid
- No Push Request:** check this box to indicate Client On Demand Push Requests are not allowed
- Keep Subscription Request History:** check this box to keep a history log of all subscription requests

Authorization Setting

- **Required User Role for Requestor:** To require requestors to have a specific role, enter the role here
- **Required User Role for Subscribers:** To require subscribers to have a specific role, enter the role here

Activation

- **Active Flag:** check this box to indicate that the On Demand Push Request is active

Next Steps

In order for the Subscription Agent ID to work, you must do the following:

1. Go to the Push Scenario Detail screen and on the General Data tab, select the Source Type **Client On Demand Request**. This changes the subscriber type.
2. On the Push Scenario Detail screen, go to the Subscription Settings tab and select **Allow Subscription**.

Exchange Object Configuration

The exchange object defines what in the exchange table is to be updated in the exchange persistent layer, what class handler should be called to update the exchange table, and what fields are related to the change detection. Use the Configuration Panel to specify which changes are relevant to the mobile application and what conditions must be satisfied for an update action to be triggered.

Exchange Object - Technical Settings

Use the Technical Settings tab to configure basic settings for an exchange object.

Exchange Object - Technical Settings Tab

The screenshot shows the 'Exchange Object Detail (Display Mode)' form with the 'Technical Settings' tab selected. The form contains the following fields and sections:

- Buttons:** 'Create' (disabled), 'Technical Settings' (selected), 'Change Detection Field Selection', 'Change Detection Condition Filter', 'Linkage Settings', 'Push Settings'.
- Fields:**
 - Exchange Object: [Text Box]
 - Mobile Application: [Text Box]
 - Application Area: [Text Box]
 - Reference Business Object: [Text Box]
 - Exchange Table Name: [Text Box]
 - Exchange Lock Object: [Text Box]
 - ExchObject Handler: [Text Box]
 - Active Flag: ☐
 - Exch. Object Desc.: [Text Box]
 - Exch. Table Desc.: [Text Box]
 - No Exchange Table Update: ☐
 - Days To Keep History: [Text Box with value '000']
- Administrative Info Section:**
 - Created By: [Text Box]
 - Creation Time Stamp: [Text Box with value '*00/00/0000 00:00:00']
 - Last Changed By: [Text Box]
 - Changed Time Stamp: [Text Box with value '*00/00/0000 00:00:00']

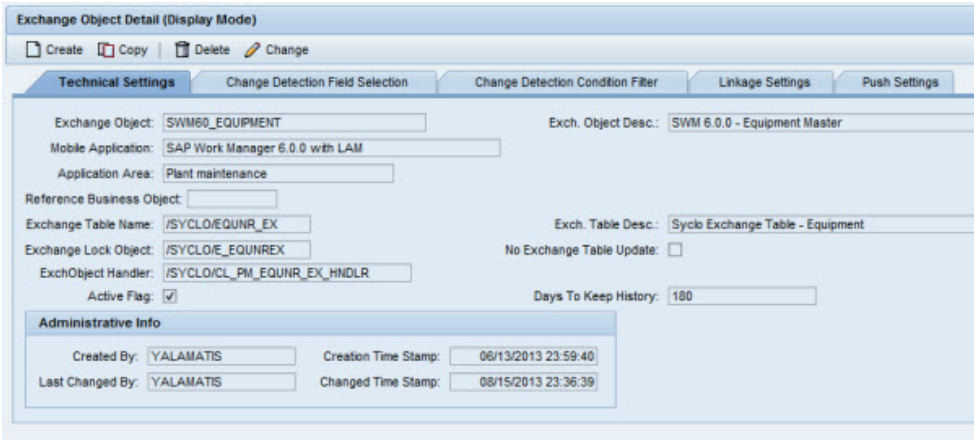
- **Exchange Object:** ID of the exchange object, limited to 40 characters
- **Exchange Object Description:** Brief description of the exchange object, limited to 60 characters
- **Mobile Application:** Specific mobile application to which the exchange object belongs using a drop-down selection field
- **Application Area:** Classifies the exchange object based on standard SAP application areas using a drop-down selection field
- **Reference Business Object:** Standard SAP business object
- **Exchange Table Name:** Name of the table stored in SAP that contains the technical data
- **Exchange Table Description:** Brief description of the exchange table
- **Exchange Lock Object:** SAP lock object used when updating the exchange table
- **No Exchange Table Update:** When checked, the record is not written to the exchange table in SAP when the record is changed.
- **Exchange Object Handler:** Name of class handler from the repository that is responsible for updating the exchange table
- **Active Flag:** When checked the exchange object is in an active state. If unchecked, the exchange object performs no actions.
- **Days to Keep History:** Number of days the historical data should be kept in the exchange table.

Administrative Info

- **Created By:** SAP user ID of the person who created the exchange object
- **Creation Time Stamp:** Date and time of the creation of the exchange object
- **Last Changed By:** SAP user ID of the person who last changed the exchange object
- **Changed Time Stamp:** Date and time of the change to the exchange object

Example

The following screen sample shows that the exchange process is enabled for Equipment. Any changes for Equipment master data will be recorded in the exchange table and transmitted to the Client during the next transmit.



Exchange Object - Change Detection Field Selection

The Change Detection Field Selection tab provides the ability to optimize the change detection process for mobile applications. If a value change is detected for any fields within the group, the object identifier is written to the exchange table, indicating that a change has been made. If the Active Flag is not checked for a field, any value changes made to that field will not be detected and recorded to SAP during the exchange process. By default, all fields are initially checked.

Figure 11: Exchange Object - Change Detection Field Selection Tab

Exchange Object Detail (Display Mode)

Create | Copy | Delete | Change

Technical Settings | **Change Detection Field Selection** | Change Detection Condition Filter | Lin

Exchange Object Info

Exchange Object: SWM60_EQUIPMENT | Exch. Object Desc.: SWM 6.0.0 - Equipment Master

ExchObject Handler: /SYCLO/CL_PM_EQUNR_EX_HNDLR

Exchange Object Field Selector

Search:

Field Catalog	Active Flag	Short Description
▶ Table - EFHM*	<input type="checkbox"/>	PRT fields in equipment master
▶ Table - EQBS*	<input type="checkbox"/>	Serial Number Stock Segment
▶ Table - FLEET	<input type="checkbox"/>	Fleet Attributes
▶ Table - ITOB*	<input type="checkbox"/>	Generated Table for View
▼ Table - JEST*	<input checked="" type="checkbox"/>	Individual Object Status
▪ Field - CHGNR	<input checked="" type="checkbox"/>	Change number
▪ Field - INACT	<input checked="" type="checkbox"/>	Status inactive
▪ Field - OBJNR	<input checked="" type="checkbox"/>	Object number
▪ Field - STAT	<input checked="" type="checkbox"/>	Status

Selection Proposal

Sort Options

☒ By Field Name
☐ By Field Description
☐ By DDIC Sequence

Exchange Object by Application Area

The Exchange Object by Application tree lists all application areas and the exchange objects linked to each application area. Expand the tree by clicking on the arrows to the left of the application area to display the exchange objects associated with it.

Exchange Object Info

- **Exchange Object:** ID of the exchange object. This is a non-editable field.
- **Exchange Object Description:** Brief description of the exchange object. This is a non-editable field.
- **Exchange Object Handler:** Name of class handler from the repository that is responsible for updating the exchange table. This is a non-editable field.

Exchange Object Field Selector

- **Field Catalog:** All fields that can be detected by the class handler when changes are made, grouped by the technical table name of the SAP business object. This is a non-editable field.
- **Active Flag:** When checked, either the table or a field within a table is active. Any value change to the selected field will be detected by the class handler.

Note: Checking the Active Flag box on a Table row selects all fields within the table.

- **Short Description:** Brief description of the table or the field within the table. This is a non-editable field.

Selection Proposal

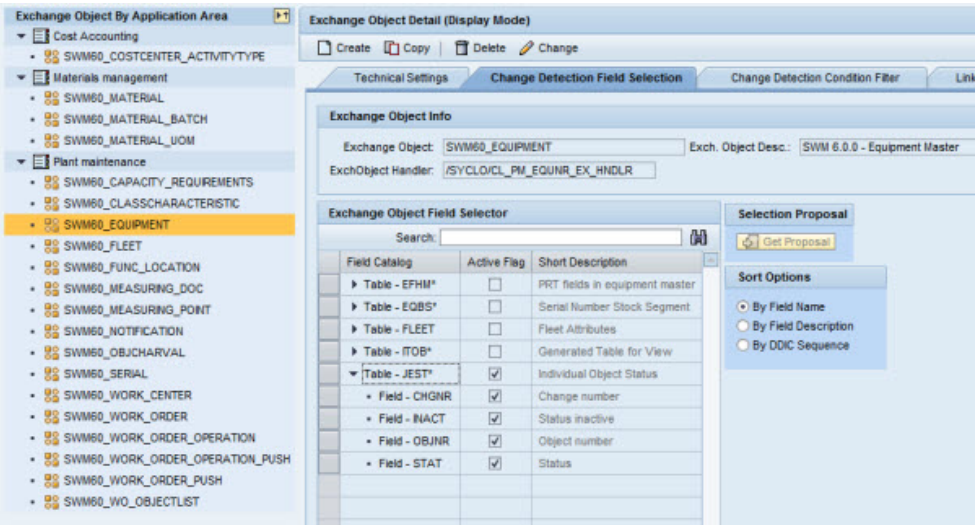
In a typical installation, it is not desirable to have all fields in all exchange tables checked as active for change detection. Rather, only the fields that are active on the mobile data object that are brought down to the mobile device should also be active in the exchange object.

Based on mobile data object usage in the mobile application, the Selection Proposal will examine the active flags that are checked for an exchange object’s table fields and provide recommendations to the administrator on which fields in the exchange object should be checked or unchecked.

Example

The properties for the enabled exchange object Equipment that should be captured and recorded in the exchange table are defined on the Change Detection Condition Filter tab.

The properties for account general data that triggers the exchange are defined on this tab, as shown in the following sample.



Exchange Object - Change Detection Condition Filter

The Change Detection Condition Filter tab provides the ability to restrict change detection based on data content. For exchange handlers to support this feature, you must define data filter conditions for which the underlying SAP business object must qualify before the change detection process is triggered. The condition is defined at the table field level and is in the SAP range table format.

Exchange Object Info

Exchange Object: SWM60_COSTCENTER_ACTIVITYTYPE Exch. Object Desc.: SWM 6.0.0 - Cost Center / Activity Type Maintenance
 ExchObject Handler: /SYCLO/CL_CO_LSTAR_EX_HNDLR

Exception Settings

Ignore Data Creation: ☐ Ignore Data Deletion: ☐ Ignore Data Update: ☐

Exchange Object Filter Rule Definition

Defined Filters

- Filter - ACTTYPE
- Filter - COSTCENTER
- Filter - CO_AREA
- Filter - FISC_YEAR

Rule Editor

Filter Name: ACTTYPE
 Reference Table Name: CSSL Reference Field Name: LSTAR
 Data Filter Rule Key:

Enter Range Value

Sign: Option:
 Low Value:
 High Value:
 Active Flag: ☐

Rule List

Rule No.	Rule Type	Rule Value	Active Flag

Exchange Table Object Info

- **Exchange Object:** ID of the exchange object. This is a non-editable field.
- **Exchange Object Description:** Brief description of the exchange object. This is a non-editable field.
- **Exchange Object Handler:** Name of class handler from the repository that is responsible for updating the exchange table. This is a non-editable field.

Exception Settings

- **Ignore Data Creation:** When checked, new records/data created are not processed to the exchange table
- **Ignore Data Deletion:** When checked, deleted records/data are not processed to the exchange table
- **Ignore Data Update:** When checked, updated records/data are not processed to the exchange table

Defined Filters

The Defined Filters box lists all data filters supported by the class handlers.

Rule Editor

- **Filter Name:** Name of the filter as defined in the class handler method. This information is defined by the class handler developer and is not editable.
- **Reference Table Name:** Technical name of the SAP database table where this filter is applied. This information is defined by the class handler developer and is not editable.
- **Reference Field Name:** Technical name of the SAP database table field where this filter is applied. This information is defined by the class handler developer and is not editable.
- **Data Filter Rule Key:** Internal technical key used by the framework at runtime

Enter Range Value

- **Sign:** Value for the SAP Range table column SIGN
- **Option:** Value for the SAP Range table column OPTION
- **Low Value:** Value for the SAP Range table column LOW
- **High Value:** Value for the SAP Range table column HIGH
- **Active Flag:** When checked, the rule is active

Rule List

The Rule List table displays a list of rules that have been defined.

- **Rule No.:** Number of the rule that is defined, in chronological order
- **Rule Type:** Rule type, automatically assigned by the rule type selected in the DOF Rule Type field
- **Rule Value:** Internal rule value saved by SAP
- **Active Flag:** When checked, the rule is active

Example

The following sample screen shows that any exchange detected for the Account will be considered only if the Account is maintained in one of the roles defined in the ROLE_TYPE filter criteria.

ConfigPanel Home

Exchange Object By Application Area

- Sales
 - SSLM25_ACCOUNT
 - SSLM25_ACTIVITY
 - SSLM25_ACTIVITY_PUSH
 - SSLM25_CONTACT
 - SSLM25_EMPLOYEE
 - SSLM25_LEAD
 - SSLM25_LEAD_PUSH
 - SSLM25_MARKETING_ATTRIBUTE
 - SSLM25_MARKETING_ATTRIBUTE_SET
 - SSLM25_MARKETING_ATTRIBUTE_VALUE
 - SSLM25_OPPORTUNITY
 - SSLM25_OPPORTUNITY_PUSH
 - SSLM25_PRODUCT
 - SSLM25_QUOTATION
 - SSLM25_QUOTATION_PUSH
 - SSLM25_SALES_ORDER
 - SSLM25_SALES_ORDER_PUSH
 - SSLM25_SURVEY
 - SSLM25_TASK
 - SSLM25_TASK_PUSH

Exchange Object Detail (Display Mode)

Create Copy Delete Change

Technical Settings Change Detection Field Selection Change Detection Condition Filter Linkage Settings

Exchange Object Info

Exchange Object: SSLM25_ACCOUNT Exch. Object Desc.: SSLM 2.5 - CRM Account

ExchObject Handler: /SSYCLO/CL_CRMMD_BUPA_EX_HDL

Exception Settings

Ignore Data Creation: ☐ Ignore Data Deletion: ☐ Ignore Data Update: ☐

Exchange Object Filter Rule Definition

Defined Filters

- Filter - BKIND
- Filter - BU_GROUP
- Filter - BU_TYPE
- Filter - ROLE_TYPE
- Filter - SOURCE

Rule Editor

Filter Name: ROLE_TYPE

Reference Table Name: BUT100 Reference Field Name: RLTYT

Data Filter Rule Key: SSLM25_ACCOUNT/SSYCLO/CL_CRMMD_BUPA_EX_HDL_ROLE_TYPE

Enter Range Value

Sign: Inclusive Option: =

Low Value: CRM000 - Sold-To Party

High Value:

Active Flag: ☒

Rule List

Rule No.	Rule Type	Rule Value	Active Flag
00001	RANGE	EOCRM000	<input checked="" type="checkbox"/>
00002	RANGE	EOBUP002	<input checked="" type="checkbox"/>
00003	RANGE	EEQBUP003	<input type="checkbox"/>

The properties for the enabled exchange object Account that should be captured and recorded in the exchange table are listed in this tab.

The properties for account general data that triggers the exchange are defined on the Change Detection Field Selection tab.

Exchange Object - Linkage Settings

The Linkage Settings tab allows the exchange objects that are linked together to communicate with each other. The communication is one-directional, with the exchange object sending information to the object(s) listed in the Linked Exchange Objects List. When there is a value change to the exchange object, that value change information is passed on to the linked exchange objects. The linked exchange objects then go through additional processes related to the value change.

Exchange Object - Linkage Settings

Exchange Object Info

- **Exchange Object:** ID of the exchange object. This is a non-editable field.
- **Exchange Object Description:** Brief description of the exchange object. This is a non-editable field.
- **Exchange Object Handler:** Name of class handler from the repository that is responsible for updating the exchange table. This is a non-editable field.

Linkage Settings

With the Linkage Hierarchy, you have the ability to go ‘n’ levels deep with linked objects. Any node changes triggers changes to the lower-level nodes linked to the parent node. These relationships are defined in the Linked Exchange Objects list.

For example:

```
Measuring Point
  Functional Location
    Work Order
```

If the Measuring Point data changes, then the Functional Location and the Work Order will change as well.

- **Maximum Linkage Hierarchy Level:** the maximum number of levels allowed to be set for linkage in the hierarchy

Linked Exchange Objects List

- **Add Linkage button:** Use this button to add a new linked exchange object. Click the **Add Linkage** button and use the fields in the Linkage Detail section to add information.
- **Delete Linkage button:** Use this button to delete a linkage. Highlight the row you wish to delete by clicking on the rectangle to the left of the Target Exchange Object cell and click the **Delete Linkage** button.
- **Target Exchange Object:** Displays the target exchange object selected in the Linkage Detail section.
- **Linkage Type:** Displays either an 'A' for asynchronous or an 'S' for synchronous, selected in the Linkage Detail section.
- **Active Flag:** When checked in the Linkage Detail section, the linkage between exchange objects is active.

Linkage Detail

- **Target Exchange Object:** The exchange objects that are linked to the exchange object listed in the Exchange Object Info section.
- **Linkage Type:** Currently, Synchronous is the only option available. When a value change occurs to the exchange object, notification to the linked exchange object is performed in real-time.
- **Exclude Data Creation / Update / Deletion:** The linkage for a source exchange object to a target exchange may be limited by the action done on the source object. The possible actions are Create, Update, and Delete. By checking any of these three 'exclude' boxes, the linkage is not triggered for that action.
- **Active Flag:** When checked, the linkage between the exchange object and the target exchange object is active.

Exchange Object - Push Settings

If the exchange object will be part of a push instance, it must be configured in the Push Settings tab before the object can appear in the Push Scenario definition Source Object drop-down menu.

' checkbox."/>

Exchange Object Info

- **Exchange Object:** ID of the exchange object. This is a non-editable field.
- **Exchange Object Description:** Brief description of the exchange object. This is a non-editable field.
- **Exchange Object Handler:** Name of class handler from the repository that is responsible for updating the exchange table. This is a non-editable field.

Push Settings

Push Relevant: When checked, the exchange object is listed as a selection in the Source Object drop-down list in the Push Scenario Definition screen.

EFI Assignment

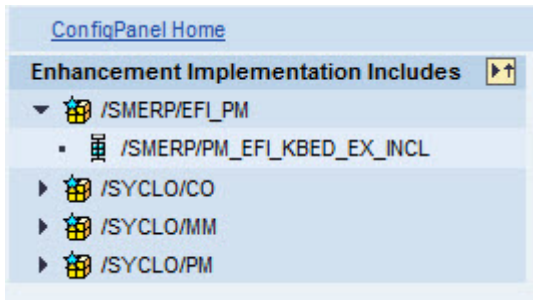
Enhancement Framework Implementation (EFI) source code plug-ins are implemented by the Agentry SAP Framework for each business object where change detection must be implemented. The source code plug-in is provided as an ABAP include file. Each exchange object is assigned to a plug-in to handle the actual change detection process. EFIs are typically available across multiple mobile applications running on the same system.

EFIs collect before and after images of data in an SAP object that has been created, modified, or deleted. The EFI then hands those images to the exchange object, which continues with the data processing. Therefore, the EFIs must be linked to the appropriate exchange objects.

Enhancement Implementation Includes

The Enhancement Implementation Includes list is a tree of the include file list in the package. Click on the arrow to the left of the first item to expand the list.

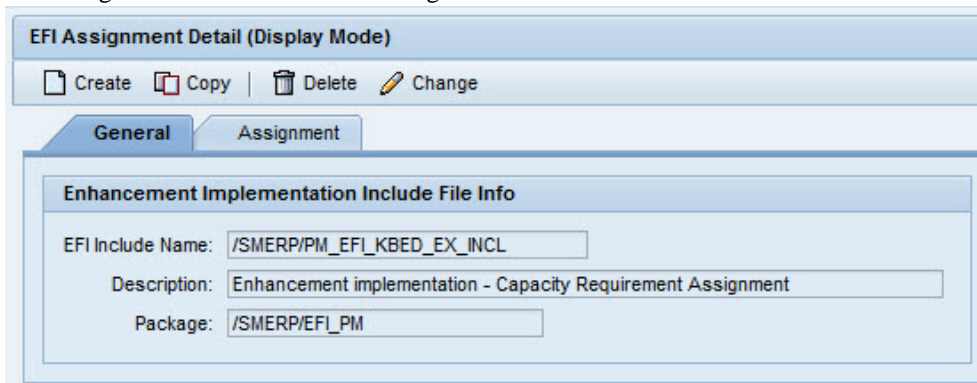
Enhancement Implementation Includes Tree



EFI Assignment - General Settings

Use the General tab to modify the general settings for a chosen EFI file.

EFI Assignment Detail - General Settings



- **EFI Include Name:** Source code plug-in file name.
- **Description:** Short description of the EFI. This is not an editable field and is automatically filled in when the EFI Include Name is selected.
- **Package:** Package where the EFI is located. This is not an editable field and is automatically filled in when the EFI Include Name is selected.

EFI Assignment - Assignment Settings

Use the Assignment tab to modify the EFI assignments.

EFI Assignment - Assignment Settings

EFI Assignment Detail (Display Mode)

Create Copy Delete Change

General **Assignment**

EFI Include Name: /SMERP/PM_EFI_KBED_EX_INCL
Description: Enhancement implementation - Capacity Requirement Assignment
Package: /SMERP/EFI_PM

EFI Assignment List

EFI Include Name	Mobile Application	Exchange Object	Exch. Object Desc.	Active Flag
/SMERP/PM_EFI_KBED_EX_INCL	SAP_WORK_MANAGER_60	SWM60_CAPACITY_REQUIREMENTS	SWM 6.0 - Capacity Requirements	<input checked="" type="checkbox"/>

Assignment Detail

Mobile Application: SAP Work Manager 6.0.0 with LAM
Exchange Object: SWM60_CAPACITY_REQUIREMENTS - SWM 6.0 - C
Exch. Object Desc.: SWM 6.0 - Capacity Requirements
Active Flag: ☒

Administrative Info

Created By: YALAMATIS Creation Time Stamp: 09/05/2013 00:30:38
Last Changed By: YALAMATIS Changed Time Stamp: 09/19/2013 01:09:36

EFI Info

The fields in this section are taken from information in the General tab and are not editable.

- **EFI Include Name:** Source code plug-in include file name
- **Description:** Description of the EFI include file name
- **Package:** SAP package to which the include file belongs

EFI Assignment List

EFI Assignment table: Table that displays which plug-ins are assigned to a specific include file. All column information is replicated in the Assignment Detail section directly below the table.

To highlight an individual row, click on the grey square to the left of the EFI Include Name column in that row.

Assignment Detail

Information in this section will change depending on which row is highlighted in the EFI Assignment List section table.

- **Mobile Application:** The specific mobile application and its release number. This field is non-editable.
- **Exchange Object:** Exchange object to which the EFI include file is assigned.

- **Exch. Object Desc:** A brief, easy to understand description of the exchange object, limited to 60 characters.
- **Active Flag:** When checked, the exchange object is in an active state. If unchecked, the EFI is not linked to the assigned mobile data object.

Administrative Info

- **Created By:** SAP user ID of the person who created the EFI assignments
- **Creation Time Stamp:** Date and time of the creation of the EFI assignments
- **Last Changed By:** SAP user ID of the person who last changed the EFI assignments
- **Changed Time Stamp:** Date and time of the change to the EFI assignments

Mobile Data Object Configuration

A mobile data object represents a mobile semantic view of data and activity combination for an SAP business object. Mobile data objects are data repositories in the namespace that can get, create, update, and delete information in SAP. They encapsulate the business logic of mobile applications by defining transactions, data structures, and business rules.

There are three types of mobile data objects:

- **DT - Data Table:** A simple representation of SAP business objects KEY and VALUE.
- **CT - Complex Table:** A two-dimensional representation of a business object with a single table of multiple columns.
- **DO - Standard Data Object:** A multi-dimensional representation of a business object with multiple tables representing different subsets of the business object.

Warning! Mobile data objects should be created, copied, or changed only by the mobile application developer or system administrator.

The configuration portal is used to easily modify mobile data object properties such as object type, class handlers, data filters, and other settings. For example, instead of modifying BAPIs to change what information is retrieved from SAP and pushed out to mobile devices, administrators can use the Configuration portal to modify mobile data objects and set up data filter rules.

Mobile Data Object - General Settings

Use the General Setting tab to modify the general settings for a chosen mobile data object.

Mobile Data Object - General Setting Tab

Data Object Detail (Display Mode)

Create
 Copy
 Delete

General Setting
 ResultSet Field Selection
 Data Filter
 Data Staging
 Proxy Setting
 Composite Settings

Basic Data

Mobile Data Object Id:

Description:

Data Object Type: Mobile Application:

Reference Business Object:

Data Object Handler Settings

Data Object Handler: Skip Exception Processing: ☐

Get Method:

Create Method:

Update Method:

Delete Method:

Exchange Object Settings

Exchange Object:

Conversion Exit Setting

Enable Conv. Exit Overwrite: ☐

Middleware Reference Info

Reference Middleware Object Type:

Activation

Data Object Active: ☐

Administrative Info

Created By: Creation Time Stamp:

Last Changed By: Changed Time Stamp:

Basic Data

The Basic Data section provides general information about the specific mobile data object. This information is used in multiple panels in the Configuration portal.

- **Mobile Data Object ID:** The name of the mobile data object, limited to 40 characters. This is a required field.
- **Description:** A brief, easy to understand description of the mobile data object, limited to 60 characters. This is a required field.

- **Data Object Type:** A drop-down list of the three mobile data object types:
 - Data Table - A simple representation of the SAP business objects KEY and VALUE
 - Complex Table - A two-dimensional representation of the business object with a single table of multiple columns
 - Standard Data Object - A multi-dimensional representation of a complete business object. It can have multiple tables representing different subsets of the business objects.
- **Mobile Application:** The name of the mobile application. Choose the mobile application from the drop-down list.
- **Reference Business Object:** The SAP business object for which the mobile data object is being created

Data Object Handler Settings

The Data Object Handler Settings section is used to configure the methods of the mobile data object.

- **Data Object Handler:** Name of the ABAP OO class handler from SAP's class repository. The ABAP OO class handler is developed by the application developer with predefined business logic and scope to perform fetch, create, delete, or update activities for an SAP business object.
- **Get Method:** Method defined in the class handler that fetches data for the underlying SAP business object. This is an optional field.
- **Create Method:** Method defined in the class handler that creates data for the underlying SAP business object. This is an optional field.
- **Update Method:** Method defined in the class handler that updates data for the underlying SAP business object. This is an optional field.
- **Delete Method:** Method defined in the class handler that deletes data for the underlying SAP business object. This is an optional field.
- **Skip Exception Processing:** When checked, if an exception occurs during mobile data object handling, the exception processing step is not invoked.

Exchange Object Settings

The Exchange Object Settings section allows the mobile data object to be associated with an exchange object. The exchange object is configured through the Exchange Object Configuration panel in the Configuration portal.

- **Exchange Object:** The name of the exchange object as defined in the SAP mobile exchange persistent layer. Specify this field by choosing an exchange object from the drop-down menu if the selected class handler should use the mobile exchange persistent layer to determine data exchanges to the mobile application.
- **Enable Conv. Exit Overwrite:** When checked, you will be able to define specific internal and external conversion settings in the Framework Technical Settings panel.

Middleware Reference Info

If the middleware is specified, the mobile data object will perform the standard exchange process as well as perform a lookup in the client object register table to determine what information the client contains. If data has been removed from the client that still exists in the SAP table, the data is re-added to the client during the transmit.

- **Reference Middleware Object Type:** Middleware objects are set through the Administration portal. This is an optional field.

Activation

Use this checkbox to enable or disable a mobile data object in the application without deleting the mobile data object.

Data Object Active: When checked, the mobile data object is in an active state. If unchecked, the mobile data object performs no actions.

Administrative Info

This section is used to easily determine which SAP user created or modified a specific mobile data object in the system.

- **Created By:** SAP user ID of the person who created the mobile data object
- **Creation Time Stamp:** Date and time of the creation of the mobile data object
- **Last Changed By:** SAP user ID of the person who last changed the mobile data object
- **Changed Time Stamp:** Date and time of the change to the mobile data object

Mobile Data Object - ResultSet Field Selection

When a field selector function is enabled for a class handler, the option to select fields for the GET method to populate is available. The class handler is designed to be mobile application-neutral. It can typically supply more data than the mobile application needs. Therefore, in order to preserve system performance, you can customize field usage settings to only retrieve required data for the mobile application. This ability prevents the need to develop a new class handler for each mobile application.

Mobile Data Object - ResultSet Field Selection Tab

Handler Info

- **Mobile Data Object ID:** Name of the mobile data object
- **Description:** Description of the mobile data object, limited to 60 characters. This is a required field.
- **Data Object Handler:** Name of the ABAP OO class handler from SAP's class repository. The ABAP OO class handler is developed by the application developer with predefined business logic and scope to perform fetch, create, delete, or update activities for an SAP business object.
- **Get Method:** Name of the GET method set in the General Setting tab.

Field Selection Detail

- **Field Catalog column:** Lists all the fields that can be returned by the class handler method, grouped in the order of the name of the class handler method, SAP table name, and field name. To display all information in this column, click on the arrows to the left of a name in order to expand the row.
- **Field Active column:** When checked, the data for the selected field is returned by the class handler method.
- **Field Description column:** Description of the specific field in the SAP table.
- **Data Format column:** How data is presented

- **Sort Options:** When there is a large amount of information presented, use the sort options to find the information required easily. When a different radio button is selected, the rows are collapsed and must be expanded again to display the new field sorting.

Mobile Data Object - Data Filter

When a data filter function is enabled for a class handler, the option exists to define various types of filter rules to control what data can be viewed by the mobile application based on a customer's business process. In an SAP environment, each user is assigned a role-based profile with authorization restrictions for what data is viewed and which activities performed.

For example, a user who works for a specific plant should not be able to view data for another plant. Data filter rules allow you to restrict data access for mobile applications. Data filters can be user-dependent or applied to the entire mobile application.

Mobile Data Object - Data Filter Tab

Data Object Detail (Display Mode)

Create Copy Delete Change

General Setting ResultSet Field Selection **Data Filter** Data Staging Proxy Setting Composite Settings

Handler Info

Mobile Data Object ID: SWM60_EQUIPMENT_COMPOSITE Description: SWM 6.0.0 - Composite Equipment fetch

Data Object Handler: /SMERPCL_PM_EQUIPMENT_DO

Data Object Filter Rule Definition

Defined Filters

- Get Method - GET
- Standard Filter
- Filter - CHAR_NAME
- Filter - CHECK_EQBS
- Filter - CHECK_FLEET_FLAG
- Filter - CHECK_ISU_FLAG
- Filter - CLASSTYP*
- Filter - CLASS_KEY
- Filter - COMP_CODE
- Filter - COSTCENTER
- Filter - CO_AREA
- Filter - DISTR_CHAN
- Filter - DIVISION
- Filter - DOC_DMS_ACTIVE*
- Filter - DOC_LINK_OBJ*
- Filter - DOC_TYPE
- Filter - EQUICATGRY
- Filter - EQUIPMENT
- Filter - EQUITYPE
- Filter - EQUIEXCL_SYST_STAT
- Filter - EQUIEXCL_USER_STAT
- Filter - EQUI_INCL_SYST_STAT

Rule Editor

Method Name: GET Filter Name: CHAR_NAME

Reference Table Name: CABN Reference Field Name: ATNAM

Data Filter Rule Key:

DOF Rule Type: Static Value in Range Format

Enter Range Value

Sign: Option:

Low Value:

High Value:

Active Flag: ☐

Rule List

Rule No.	Rule Type	Rule Value	Active Flag

Handler Info

- **Mobile Data Object ID:** Name of the mobile data object
- **Description:** Description of the mobile data object, limited to 60 characters. This is a required field.
- **Data Object Handler:** Name of the ABAP OO class handler from SAP's class repository. The ABAP OO class handler is developed by the application developer with predefined

business logic and scope to perform fetch, create, delete, or update activities for an SAP business object.

Defined Filters

The Defined Filters tree lists all data filters supported by the class handlers defined in the Data Object Handler Settings field in the General Setting tab. To expand the tree, click on the arrows to the left of the class handler methods to display the filters associated with the methods.

Rule Editor

- **Method Name:** Name of the class handler method where the data filter is defined. The data filter function is only supported for the GET method.
- **Filter Name:** Name of the filter as defined in the class handler method. This information is defined by the class handler developer and is not editable.
- **Reference Table Name:** Technical name of the SAP database table where this filter is applied. This information is defined by the class handler developer and is not editable.
- **Reference Field Name:** Technical name of the SAP database table field where this filter is applied. This information is defined by the class handler developer and is not editable.
- **Data Filter Rule Key:** Internal technical key used by the framework at runtime
- **DOF Rule Type:** Type of rule

There are three different types of rules:

- User Profile Parameter
- Static Value in Range Format
- Syclo Filter Class Handler
- Runtime Session Data

The settings are dependent on the specific type of rule set for the filter.

Data Filter - User Profile Parameter Rule

The screenshot shows the 'Rule Editor' window with the following fields and values:

- Method Name: GET
- Filter Name: CLASSTYPE
- Reference Table Name: KSSK
- Reference Field Name: KLART
- Data Filter Rule Key: SWM60_FUNC_LOCATION_COMPOSITE.GET.CLASSTYPE
- DOF Rule Type: User Profile Parameter (selected from a dropdown)

Below these fields is a section titled 'Select Parameter' containing:

- Parameter ID: [empty text box]
- Description: [empty text box]
- Active Flag: ☒

- **Parameter ID:** Memory parameter ID as defined in SAP and specified in the user profile. Click on the icon to the right of the field box to perform a search on all available parameter IDs.
- **Description:** Description of the memory parameter ID. This is not an editable field and is automatically filled in when the parameter ID is selected.
- **Active Flag:** When checked, the rule is active.

Data Filter - Static Value in Range Format

The screenshot shows the 'Rule Editor' window. At the top, there are fields for 'Method Name' (GET), 'Filter Name' (CLASSTYPE), 'Reference Table Name' (KSSK), and 'Reference Field Name' (KLART). Below these is the 'Data Filter Rule Key' (SWM60_FUNC_LOCATION_COMPOSITE.GET.CLASSTYPE) and a dropdown for 'DOF Rule Type' set to 'Static Value In Range Format'. A section titled 'Enter Range Value' contains a 'Sign' dropdown (Inclusive), an 'Option' dropdown (=), 'Low Value' (003), 'High Value' (empty), and an 'Active Flag' checkbox (checked).

- **Sign:** Value for the SAP Range table column SIGN
- **Option:** Value for the SAP Range table column OPTION
- **Low Value:** Value for the SAP Range table column LOW
- **High Value:** Value for the SAP Range table column HIGH
- **Active Flag:** When checked, the rule is active

Data Filter - Syclo Filter Class Handler

The screenshot shows the 'Rule Editor' window with the same top fields as the previous one. The 'DOF Rule Type' dropdown is now set to 'Syclo Filter Class handler'. Below this is a section titled 'Select Filter Class Handler' which includes a 'Syclo Data Filter Handler' dropdown, a 'Parameter' text field, and an 'Active Flag' checkbox (checked).

- **Syclo Data Filter Handler:** Name of the handler class as defined in the system.

- **Parameter:** Additional processing information that is passed to the class handler. The parameters are entered as free-text, and the syntax of the parameter stream is defined by the developer.
- **Active Flag:** When checked, the rule is active.

Data Filter - Runtime Session Data

The screenshot shows the 'Rule Editor' window. It contains several input fields and a dropdown menu. The 'Method Name' is 'GET', 'Filter Name' is 'CLASSTYPE', 'Reference Table Name' is 'KSSK', and 'Reference Field Name' is 'KLART'. The 'Data Filter Rule Key' is 'SWM60_FUNC_LOCATION_COMPOSITE.GET.CLASSTYPE'. The 'DOF Rule Type' is set to 'Runtime Session Data'. Below this, there is a section titled 'Specify Runtime Session Data Info' which includes fields for 'Runtime Session Data Name', 'Runtime Session Data Group', and an 'Active Flag' checkbox which is checked.

- **Runtime Session Data Name:**
- **Runtime Session Data Group:**
- **Active Flag:** When checked, the rule is active.

Rule List

The Rule List table displays a list of rules that have been defined.

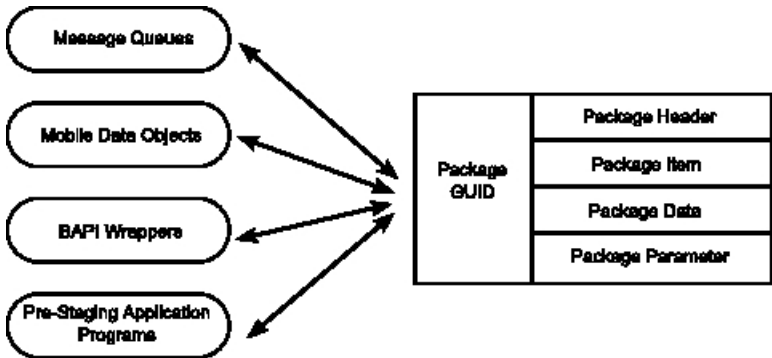
- **Rule No.:** Number of the rule that is defined, in chronological order
- **Rule Type:** Rule type, automatically assigned by the rule type selected in the DOF Rule Type field
- **Rule Value:** Internal rule value saved by SAP

Mobile Data Object - Data Staging

If an application processes a large amount of objects, data staging of the objects can assist with processing times. If an object is configured for data staging, the data within the object is stored as a package and is split into packets. The data can contain metadata and tagging for easy lifecycle management and data lookup. Standard APIs are provided for package management.

The following figure illustrates a general-purpose data staging model.

Figure 12: Mobile Data Object - Data Staging Model



Mobile Data Object - Data Staging Tab

Data Object Detail (Display Mode)

☐ Create ☐ Copy ☐ Delete ☐ Change

General Setting ResultSet Field Selection Data Filter **Data Staging** Proxy Setting Composite Settings

Handler Info

Mobile Data Object Id: Description:
Data Object Handler:

Data Staging Setting

Common Setting

Disable Conversion Exit for Package Read: ☐
Require Meta Data: ☐

Get Method Setting

Data Staging Supported: ☐

Create Method Setting

Data Staging Supported: ☐

Update Method Setting

Data Staging Supported: ☐

Delete Method Setting

Data Staging Supported: ☐

Handler Info

- **Mobile Data Object ID:** Name of the mobile data object

- **Description:** Description of the mobile data object, limited to 60 characters. This is a required field.
- **Data Object Handler:** Name of the ABAP OO class handler from SAP's class repository. The ABAP OO class handler is developed by the application developer with predefined business logic and scope to perform fetch, create, delete, or update activities for an SAP business object.

Data Staging Setting

- **Disable Conversion Exit for Package Read:**
- **Require Metadata:**
- **Get / Create / Update / Delete Method Setting:** When a checkbox is marked for a specific method, data staging is active for that method.

Mobile Data Object - Proxy Setting

Figure 13: Mobile Data Object - Proxy Setting Tab

Data Object Detail (Display Mode)

Create Copy Delete Change

General Setting ResultSet Field Selection Data Filter Data Staging **Proxy Setting** Composite Setting

Handler Info

Mobile Data Object Id: Description: Data Object Handler:

Proxy Settings

GET Method Setting

System Component: Proxy Type: BAPI Proxy Proxy Name: Proxy Active:

CREATE Method Setting

System Component: Proxy Type: BAPI Proxy Proxy Name: Proxy Active:

UPDATE Method Setting

System Component: Proxy Type: BAPI Proxy Proxy Name: Proxy Active:

DELETE Method Setting

System Component: Proxy Type: BAPI Proxy Proxy Name: Proxy Active:

Handler Info

- **Mobile Data Object ID:** Name of the mobile data object
- **Description:** Description of the mobile data object, limited to 60 characters. This is a required field.
- **Data Object Handler:** Name of the ABAP OO class handler from SAP's class repository. The ABAP OO class handler is developed by the application developer with predefined business logic and scope to perform fetch, create, delete, or update activities for an SAP business object.

Proxy Settings

The following four settings are available for each of the methods.

- **System Component:** Identifies the remote system

- **Proxy Type:** BAPI Proxy is the only supported proxy type
- **Proxy Name:** BAPI name of the remote system
- **Proxy Active:** When checked, proxy settings are active for the specific method.

Mobile Data Object - Composite Settings Tab

Use to Composite Settings tab to define the data object type Composite Settings. This allows you to combine and use multiple MDOs.

Data Object Detail (Display Mode)

Create Copy Delete

General Setting ResultSet Field Selection Data Filter Data Staging Proxy Setting **Composite Settings**

Handler Info

Mobile Data Object Id: Description:

Data Object Handler:

Composite Settings

Origin Method Type	Assigned Mobile Data Object	Assigned Method Type	Active Flag	Input Cascade	Output FIFO	Output Append

Composite Assignment Detail

Origin Method Type:

Assigned Mobile Data Object Id: Assigned Method Type:

Active Flag: ☐

Input Parameter Cascading: ☐

Output First In First Out: ☐ Output Allow Appending: ☐

Handler Info

- **Mobile Data Object ID:** the ID of the MDO to be used
- **Description:** description of the MDO being used
- **Data Object Handler:** name of the handler for the data object

Composite Settings

- **Origin Method Type:** by default this is GET method
- **Assigned Mobile Data Object:** the MDO ID
- **Assigned Method Type:** the defined method type
- **Active Flag:** indicates if this setting is active or not
- **Input Cascade:** indicates whether cascading has been allowed
- **Output FIFO:** output settings indicate first in, first out

- **Output Append:** indicates if results will be aggregated

Composite Assignment Detail

- **Origin Method Type:** by default, this is GET method
- **Assigned Mobile Data Object ID:** enter the MDO ID
- **Assigned Method Type:** enter the method type
- **Active Flag:** check this to indicate active status
- **Input Parameter Cascading:** when the BAPI is called, this allows a chain of MDOs to execute
- **Output First In First Out:** check this to define the output as first in, first out
- **Output Allow Appending:** check this to aggregate results

BAPI Wrapper Configuration

A BAPI wrapper is created by the application developer to expose SAP data and business logic to the mobile application. By design, the BAPI wrapper does not contain any business logic. Each BAPI wrapper must be assigned to a specific method type (GET, CREATE, UPDATE, or DELETE) of a mobile data object to perform the required business logic. By decoupling the business logic from the BAPI wrapper, it is possible to switch mobile data objects without affecting the underlying mobile application definition.

BAPI Wrapper - General Settings

Use the General tab to modify the general settings for a chosen BAPI wrapper.

BAPI Wrapper - General Settings

BAPI Wrapper Detail (Display Mode)

Create Copy Delete Change

General Assignment

BAPI Wrapper Info

BAPI Wrapper Name: /SMERP/PM_CTPLANTLOCATION_GET
 Description: Fetch PM Plant Location
 Function Group: /SMERP/BAPI_PM_PLANT_LOC Function Group Description: PM: PM Plant Location Fetch
 Package: /SMERP/BAPI_PM

Technical Info

BAPI Parameter List

- IMPORT
 - IS_BAPI_INPUT
- EXPORT
 - ES_BAPI_OUTPUT
- TABLE
 - ET_COMPLEX_TABLE
 - ET_EXCHANGE_ACTION_DELETED
 - ET_RETURN
 - IT_PLANT_RA

Field List

Field Name	Field Label	Data Format	Conversion Routine

BAPI Wrapper Info

- **BAPI Wrapper Name:** Technical name of the Remote Function Call (RFC) function module defined in the system
- **Description:** Description of the BAPI wrapper, limited to 60 characters. This is a required field.
- **Function Group:** Function group to which the BAPI belongs
- **Function Group Description:** Description of the function group, limited to 60 characters.
- **Package:** SAP group to which the function group and the BAPI wrapper both belong

BAPI Wrapper List

The BAPI wrapper list provides an expandable tree of the available BAPI wrapper function groups and the BAPI wrapper name(s) associated with each function group. Use the arrows to the left of the function name to display all BAPI wrapper names under the function group.

Technical Info

- **Field Name column:** Field name of the structure
- **Field Label column:** Text describing the structure

- **Data Format column:** Standard SAP data type
- **Conversion Routine column:** SAP-defined conversion routine

BAPI Wrapper - Assignment Settings

Use the Assignment tab to hook up BAPI wrapper assignments to mobile data objects. In this tab, you can change a BAPI wrapper's assignment to specific mobile data objects, assign a new mobile data object to the BAPI wrapper, or delete a mobile data object assignment from the BAPI wrapper.

BAPI Wrapper - Assignment Settings

BAPI Wrapper Detail (Display Mode)

Create Copy Delete Change

General **Assignment**

BAPI Wrapper Info

BAPI Wrapper Name: /SHERPIM_CTPLANTLOCATION_GE
Description: Fetch PM Plant Location

Mobile Data Object Assignment List

Mobile Application	Mobile Data Object Id	Description	Method Type	Active Flag	Default Assignment
SAP_WORK_MANAGER_60	SWM60_PLANTLOCATION	SWM 6.0.0 - Plant Location	GET	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Assignment Detail

Mobile Application: SAP Work Manager 6.0.0 with LAM
Mobile Data Object Id: SWM60_PLANTLOCATION:CT - SWM 6.0.0 - Plant I
Description: SWM 6.0.0 - Plant Location
Method Type: Get method
Active Flag: ☒ Default Assignment: ☒

Administrative Info

Created By: JONESCHR1 Creation Time Stamp: 06/18/2013 20:45:19
Last Changed By: JONESCHR1 Changed Time Stamp: 06/18/2013 20:45:19

BAPI Wrapper Info

- **BAPI Wrapper Name:** Technical name of the Remote Function Call (RFC) function module defined in the system
- **Description:** Description of the BAPI wrapper, limited to 60 characters. This is a required field.

Mobile Data Object Assignment List

Mobile Data Object Assignment table: Table that displays which mobile data objects are assigned to each BAPI wrapper. All column information is replicated in the Assignment Detail section directly below the table.

To highlight an individual row, click on the grey square to the left of the Mobile Application column in that row.

Assignment Detail

Information in this section will change depending on which row is highlighted in the Mobile Data Assignment List section table.

- **Mobile Application:** The specific mobile application and its release number. This field is non-editable.
- **Mobile Data Object ID:** The name of the mobile data object in a drop-down field.
- **Description:** A brief, easy to understand description of the mobile data object, limited to 60 characters. This field is non-editable.
- **Method Type:** Mobile data object method (GET, CREATE, UPDATE, or DELETE) that is assigned to the BAPI wrapper.
- **Active Flag:** When checked, the mobile data object is in an active state. If unchecked, the mobile data object performs no actions.
- **Default Assignment:** When checked, the specific mobile data object assigned to that BAPI wrapper is primary. If no mobile data object ID is specified in the standard BAPI wrapper input parameter 'IS_BAPI_INPUT-DO_ID', then the primary mobile data object is used during runtime.

It is possible to assign multiple MDOs to the same BAPI in a single mobile application. To override the default MDO assignment at runtime and to specify the desired MDO, the DO_ID field must be defined in IS_BAPI_INPUT.

Administrative Info

- **Created By:** SAP user ID of the person who created the BAPI wrapper assignments
- **Creation Time Stamp:** Date and time of the creation of the BAPI wrapper assignments
- **Last Changed By:** SAP user ID of the person who last changed the BAPI wrapper assignments
- **Changed Time Stamp:** Date and time of the change to the BAPI wrapper assignments

Security Settings

Security settings are used to provide additional rules and roles on top of the standard SAP-provided rules and roles.

Security Settings - System Security

Use this tab to configure system security settings that are mobile application-independent. System security settings apply to all applications running on the framework.

Security Settings - System Security

[illegible]

Security Check Rule List

- **Add Rule button:** Press the **Add Rule** button to add a new system security rule. Fill in the fields in the Rule Detail section to automatically fill in the fields in this table.
- **Delete Rule button:** Press the **Delete Rule** button to delete an system security rule. To delete a system security rule, press the rectangle to the left of the Rule Type column in the row you wish to delete and press the **Delete Rule** button.
- **Rule Type:** Rule Type from the Rule Detail section. This is a non-editable field.
- **Object Name:** Taken from the Profile field in the Rule Detail section if rule type Authorization Object is chosen. This is a non-editable field.

- **Authorization Field Name:** Standard SAP authorization object name. This is a non-editable field.
- **Authorization Field Value:** Free-text field. Text entered depends on developer implementation in SAP. This is a non-editable field.
- **System Admin Indicator:** When Rule Type: User Role is selected, taken from the System Admin Indicator field in the Rule Detail section. This is a non-editable field.

Rule Detail - Rule Type: User Role

- **Rule Type: User Role:** In addition to the standard SAP user profile rules, the user role can add restrictions on what a user can or cannot see in menus or other mobile application screens.
- **Role:** User role in SAP. To search for a user role, click on the white box icon to the right of the Role field to display the Role Selection search window.
- **Name:** Brief description of the role. This is a non-editable field.
- **System Admin Indicator:** Drop-down menu with four choices:
 - **System administrator:** User role can view system activity and make changes to system administration setup in the Administration portal.
 - **System administration - View only:** User role can view the system activity in the Administration portal, but cannot make changes to the setup.
 - **System configurator:** User role can view system configuration and make changes to the setup in the Configuration portal.
 - **System configuration - View only:** User role can view the system configuration in the Configuration portal, but cannot make changes to the setup.

Once the System Admin Indicator roles have been configured, the configurations and roles are available in both the Configuration and the Administration & Monitoring portals.

Rule Detail - Rule Type: Authorization Profile

- **Rule Type: Authorization Profile:** A collection of objects, or roles, such as Technician or Supervisor.
- **Profile:** Authorization profile in SAP. To search for an authorization profile, click on the white box icon to the right of the Profile field. The Profile Selection window displays.
- **Text:** Brief description of the authorization profile. This is a non-editable field.

Rule Detail - Rule Type: Authorization Object

- **Rule Type: Authorization Object:** Baseline object used across mobile applications
- **Authorization Object:** Authorization object in SAP.
- **Authorization Field:** Standard SAP authorization object name
- **Authorization Field:** Free-text field. Text entered depends on developer implementation in SAP.

Security Settings - Product Security

Use this tab to configure security settings for a specific mobile application.

Security Settings - Product Security

Security Rule Settings

System Security

Product Security

Syclo Class Handler Security

Security Check Rule List

Add Rule

Delete Rule

Product	Rule Type	Object Name	Authorization Field	Authorization Field Value
SAP_WORK_MANAGER_60	User Role			

Rule Detail

Product Info

Product: SAP Work Manager 6.0.0 with LAM

Security Rule Type

Rule Type: User Role

Select A User Role

Role:

Name:

Security Check Rule List

- **Add Rule button:** Press the **Add Rule** button to add a new product security rule. Fill in the fields in the Rule Detail section to automatically fill in the fields in this table.

- **Delete Rule button:** Press the **Delete Rule** button to delete an product security rule. To delete a product security rule, press the rectangle to the left of the Rule Type column in the row you wish to delete and press the Delete Rule button.
- **Product:** Mobile application chosen in the Rule Detail section. This is a non-editable field.
- **Rule Type:** Rule Type from the Rule Detail section. This is a non-editable field.
- **Object Name:** Taken from the Profile field in the Rule Detail section if rule type Authorization Object is chosen. This is a non-editable field.
- **Authorization Field:** Taken from the Authorization Field selection if rule type Authorization Object is chosen. Standard SAP authorization object name. This is a non-editable field.
- **Authorization Field Value:** Taken from the Field Value selection if rule type Authorization Object is chosen.

Rule Detail - Security Rule Type: User Role

- **Product:** Select the mobile application that will contain the product security rule.
- **Rule Type: User Role:** In addition to the standard SAP user profile rules, the user role can add restrictions on what a user can or cannot see in menus or other mobile application screens.
- **Role:** Select a user role contained within SAP.
- **Name:** After the user role is selected, this non-editable field is filled in with the descriptive name of the user role.

Rule Detail - Security Rule Type: Authorization Profile

- **Product:** Select the mobile application that will contain the product security rule.
- **Rule Type: Authorization Profile:** A collection of objects, or roles, such as Technician or Supervisor.
- **Profile:** Select an authorization profile contained within SAP.
- **Text:** After the authorization profile is selected, this non-editable field is filled in with the descriptive name of the authorization profile.

Rule Detail - Security Rule Type: Authorization Object

- **Product:** Select the mobile application that will contain the product security rule.
- **Rule Type: Authorization Object:** Baseline object used across mobile applications.
- **Authorization Object:** Select an authorization object contained within SAP.
- **Authorization Field:** Select a field contained within SAP.
- **Field Value:** Free-text field for additional object configuration. Text entered depends on developer implementation in SAP.

Security Settings - Class Handler Security

Use this tab to configure class handler security settings that cross mobile applications, but are only applicable for the selected data object handler chosen in the Rule Detail pane.

Security Settings - Class Handler Security

The screenshot shows the 'Security Settings - Class Handler Security' interface. It features three tabs: 'System Security', 'Product Security', and 'Syclo Class Handler Security'. The 'Syclo Class Handler Security' tab is selected. Below the tabs is a 'Security Check Rule List' section containing a table with the following columns: 'Class Handler', 'Class Method', 'Rule Type', 'Object Name', 'Authorization Field', and 'Authorization Field Value'. Above the table are buttons for 'Add Rule' and 'Delete Rule'. Below the table is a status bar showing 'Row 1 of 1'. Below the table is a 'Rule Detail' section with the following fields:

- Data Object Handler Info:**
 - Data Object Handler: [Dropdown]
 - Handler Method: [Dropdown]
- Security Rule Type:**
 - Rule Type: [Dropdown]
- Select A User Role:**
 - Role: [Text Field]
 - Name: [Text Field]

Security Check Rule List

- **Add Rule button:** Press the [Add Rule] button to add a new class handler security rule. Fill in the fields in the Rule Detail section to automatically fill in the fields in this table.
- **Delete Rule button:** Press the [Delete Rule] button to delete an class handler security rule. To delete a class handler security rule, press the rectangle to the left of the Rule Type column in the row you wish to delete and press the Delete Rule button.

- **Class Handler:** Taken from the Data Object Handler field in the Rule Detail section. This is a non-editable field.
- **Class Method:** Taken from the Handler Method field in the Rule Detail section. This is a non-editable field.
- **Rule Type:** Rule Type from the Rule Detail section. This is a non-editable field.
- **Object Name:** Taken from the Profile field in the Rule Detail section if rule type Authorization Object is chosen. This is a non-editable field.
- **Authorization Field:** Taken from the Authorization Field selection if rule type Authorization Object is chosen. Standard SAP authorization object name. This is a non-editable field.
- **Authorization Field Value:** Taken from the Field Value selection if rule type Authorization Object is chosen.

Rule Detail - Security Rule Type: User Role

- **Data Object Handler:** Select the desired class handler from the drop-down list.
- **Handler Method:** Select the desired handler method from the drop-down list.
- **Rule Type: User Role:** In addition to the standard SAP user profile rules, the user role can add restrictions on what a user can or cannot see in menus or other mobile application screens.
- **Role:** Select a user role contained within SAP.
- **Name:** After the user role is selected, this non-editable field is filled in with the descriptive name of the user role.

Rule Detail - Security Rule Type: Authorization Profile

- **Data Object Handler:** Select the desired class handler from the drop-down list.
- **Handler Method:** Select the desired handler method from the drop-down list.
- **Rule Type: Authorization Profile:** A collection of objects, or roles, such as Technician or Supervisor.
- **Profile:** Select an authorization profile contained within SAP.
- **Text:** After the authorization profile is selected, this non-editable field is filled in with the descriptive name of the authorization profile.

Rule Detail - Security Rule Type: Authorization Object

- **Data Object Handler:** Select the desired class handler from the drop-down list.
- **Handler Method:** Select the desired handler method from the drop-down list.
- **Rule Type: Authorization Object:** Baseline object used across mobile applications
- **Authorization Object:** Select an authorization object contained within SAP.
- **Authorization Field:** Select a field contained within SAP.
- **Field Value:** Free-text field for additional object configuration. Text entered depends on developer implementation in SAP.

System Administration

All components of the Agency SAP Framework Administration, such as BAPI wrappers, mobile data objects and exchange objects, support logging. Activity logs generated by the Agency SAP Framework are integrated into the standard SAP Application Log database.

The recommended time frame for keeping the logs is:

- **Customization Table logs** - Do not delete
- **Exchange Table logs** - Keep no more than 6 months (180 days)
- **System logs** - Keep no more than 30 days

Viewing the Mobile Application Log

To view the application logs:

1. Start the Mobile Administration Menu by running the command `/n/syclo/smart` from the command field of SAPGUI session.
2. Select the transaction **/SYCLO/SLG1 - Application Log: Display Logs** to launch the start screen of the log display.

The screenshot displays the 'Analyze Application Log' transaction in SAP GUI. The interface includes a menu bar (Program, Edit, Goto, System, Help) and a toolbar. The main area is divided into several sections for filtering logs:

- Object:** Set to `/SYCLO/`.
- Subobject:** Empty.
- External ID:** Empty.
- Time Restriction:**
 - From (Date/Time): 05/12/2008 00:00:00
 - To (Date/Time): 05/12/2008 23:59:59
- Log Triggered By:**
 - User: *
 - Transaction code: *
 - Program: *
- Log Class:**
 - ☐ Only very important logs
 - ☐ Only important logs
 - ☐ Also less important logs
 - ☒ All logs
- Log Creation:**
 - ☒ Any
 - ☐ Dialog
 - ☐ In batch mode
 - ☐ Batch input
- Log Source and Formatting:**
 - ☒ Format Completely from Database
 - ☐ Format Only Header Data from Archive
 - ☐ Format Completely from Archive

3. Enter the desired the selection criteria. Make sure Object is set to `/SYCLO/`. Execute the transaction to view log details.

Display logs

Date/Time/User	Numb	External ID	Object/bt	Sub-object/text	Tran	Program	Mode	Log number
05/12/2008 12:58:55 DAVID01	38		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260551
05/12/2008 12:59:32 DAVID01	18		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260552
05/12/2008 13:09:10 JMWANG	16		Syco Smart Mobile Solutions	Default sub-object	SE37		Dialog processing	00000000000002260553
05/12/2008 13:20:19 DAVID01	84		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260554
05/12/2008 13:22:35 DAVID01	15		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260555
05/12/2008 13:23:25 DAVID01	15		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260556
05/12/2008 13:24:38 DAVID01	15		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260557
05/12/2008 13:25:34 DAVID01	41		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260558
05/12/2008 13:35:14 DAVID01	52		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260559
05/12/2008 13:36:45 DAVID01	13		Syco Smart Mobile Solutions	Default sub-object	SE37		Dialog processing	00000000000002260560
05/12/2008 13:41:44 DAVID01	88		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260561
05/12/2008 13:46:01 DAVID01	84		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260562
05/12/2008 13:49:57 DAVID01	16		Syco Smart Mobile Solutions	Default sub-object	SE37		Dialog processing	00000000000002260563
Problem class Additional info 16								
05/12/2008 13:52:15 DAVID01	23		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260564
05/12/2008 13:52:47 DAVID01	23		Syco Smart Mobile Solutions	Default sub-object			Dialog processing	00000000000002260565

Message Text

05/12/2008 13:49:57	*****START*****BAPI/SYCLOMM_CTMATPLANT_GET : BAPI Starts By User DAVID01 Device Dummy
05/12/2008 13:49:57	BAPI/SYCLOMM_CTMATPLANT_GET : Create Data Object CTMATPLANT
05/12/2008 13:49:57	Calling Data Object Handler /SYCLOCL_MM_MATERIAL_DO
05/12/2008 13:49:57	ICLASS=/SYCLOCL_MM_MATERIAL_DO : Entering CONSTRUCTOR...
05/12/2008 13:49:57	BAPI/SYCLOMM_CTMATPLANT_GET : Calling Handler Method GET_MATPLANT
05/12/2008 13:49:57	ICLASS=/SYCLOCL_MM_MATERIAL_DO : Entering method - EXECUTE...
05/12/2008 13:49:57	: Entering method - GET_STR_BAPI_INPUT...
05/12/2008 13:49:57	ICLASS=/SYCLOCL_MM_MATERIAL_DO : Entering method - GET_STR_BAPI_INPUT...
05/12/2008 13:49:57	ICLASS=/SYCLOCL_MM_MATERIAL_DO : Entering method - GET_MATPLANT...
05/12/2008 13:49:57	ICLASS=/SYCLOCL_MM_MATERIAL_DO : Entering method - GET_MATPLANT...
05/12/2008 13:49:57	: Entering method - DETERMINE_DATA_FILTER_VALUE...
05/12/2008 13:49:57	: Entering method - GET_MATPLANT...
05/12/2008 13:49:57	: Entering method - BUILD_FILED_SELECTOR_STRINGS...
05/12/2008 13:49:57	: Entering method - GET_FIELD_SELECTOR_TABLES...
05/12/2008 13:49:57	BAPI/SYCLOMM_CTMATPLANT_GET : Retrieving Output Parameters
05/12/2008 13:49:57	*****END*****BAPI/SYCLOMM_CTMATPLANT_GET : BAPI Ends By User DAVID01 Device Dummy

Deleting an SAP Mobile Application Log

You should purge these logs every 30 days. To delete an application log:

1. Select transaction **/SYCLO/SLG2 - Application Log: Delete Logs** to launch the selection screen.

Application log: Delete Expired Logs

Delete logs

All logs are deleted which satisfy the following selection conditions, and for which:

- the expiry date is reached or passed
- the expiry date is not defined

Expiry date

☐ Only logs which have reached their expiry date

☒ and logs which can be deleted before the expiry date

Selection conditions

Object	/SYCLO/	to		
Subobject		to		
External ID		to		
Transaction code		to		
User		to		
Log number		to		
Problem class		to		
from (datetime)			00:00:00	
to (datetime)			00:00:00	

Options

☐ Only calculate how many

☐ Generate list

☒ Delete immediately

Delete by Number of Logs

COMMIT Counter: 100

2. Enter selection criteria, and execute the transaction to purge the log database. The system will confirm deletion of logs in a popup window before deleting them.

Purge Utility for Exchange Persistent Layer

The purge utility for the exchange persistent layer is a tool that allows the system administrator to purge the content of the exchange tables. Exchange tables are not intended to be history tables. They should be purged periodically to maintain the exchange performance with the mobile application. It is recommended that you keep these tables no longer than every six months (180 days).

To purge the exchange table online:

1. Select transaction **/SYCLO/EX_PURGE - Exchange Table Purge Utility** to launch the selection screen.

Syclo Exchange Table Purge Utility Program

Selection Criteria

Mobile Application				
Exchange Object		to		
Last Changed By		to		
Object Key		to		
Exchange Action		to		
Record Status		to		

☐ Delete All Entries

2. Specify the list of **Mobile Applications** to be included. Leave this blank to include all applications installed.
3. Specify the list of **Exchange Objects** to be included. Leave this blank to include all exchange objects.
4. Enable the option **Delete All Entries** to delete all records in the exchange tables. If the delete all option is not chosen, check the **Value of 'Days To Keep History'** in the exchange object settings to determine what data should be purged.
5. Use the program **/SYCLO/CORE_EXCH_PURGE_PROG** to schedule a background job to carry out a periodic purge automatically.

Accessing the Administration and Monitoring Portal

Prerequisites

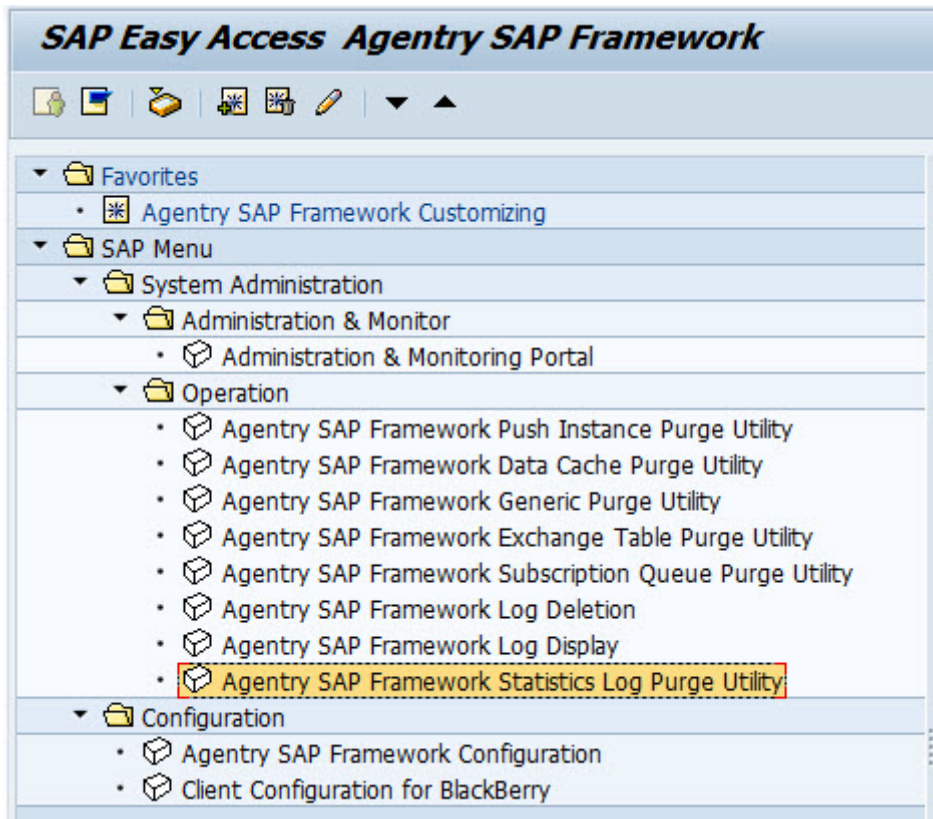
The person performing this procedure must log into the SAP system as a user with the following authorizations:

- *Authorization object* - S_ICF
- *Authorization field* - ICF_FIELD - SERVICE
- *Authorization field* - ICF_VALUE - SYCLOADM
- *Authorization object* - S_TCODE
- *Authorization field* - TCD - /SYCLO/ADMIN

Task

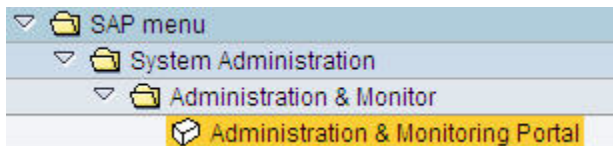
1. Log into SAP under an account with one of the authorizations provided in the prerequisites to this procedure.
2. Type `/n/syclo/smart` into the command field and click the green check mark to the left of the field, or press Enter.

The SAP Easy Access Agentry SAP Framework window displays.



- Expand the SAP menu tree by clicking the arrows to the left of the menu items. Expand as follows: **SAP menu | System Administration |Administration & Monitor**.

The available administrative functions display.



- Double-click on the Administration & Monitoring Portal menu item.

The SAP NetWeaver Web Application Server logon screen opens in a browser window.

SAP NetWeaver™
SAP Web Application Server

No switch to HTTPS occurred, so it is not secure to send a password

SSO logon not possible; logon tickets not activated on the server

Choose "Logon" to continue. A dialog box appears in which you can enter your user and password

No switch to HTTPS occurred, so it is not secure to send a password

System: QE0
Client *: 800
Users: Via Popup
Password: Via Popup
Language: English

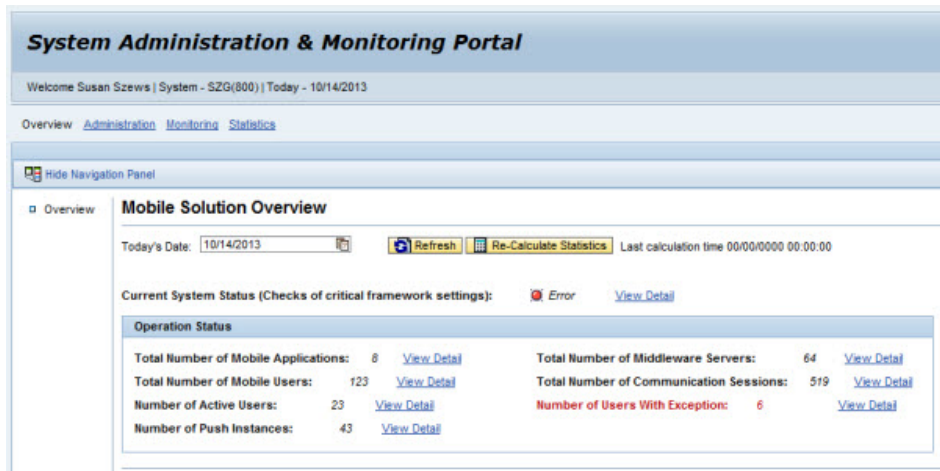
Log On

[Change Password](#)

Copyright 2002-2005 SAP AG All Rights Reserved

Note: Depending on your configuration, the log on screen may be different than what is shown.

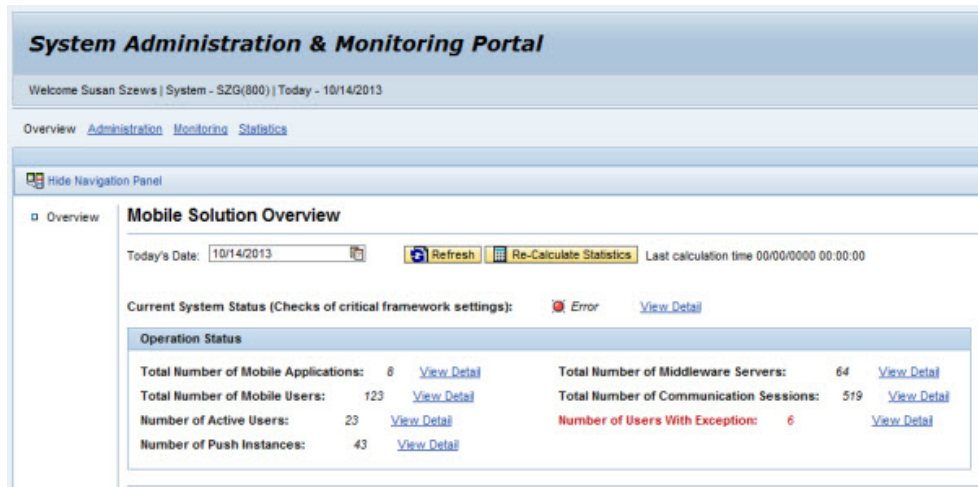
5. Fill in any necessary fields for your specific implementation and click the **Log On** button. The System Administration & Monitoring portal opens in the browser window.



Administration Portal - Mobile Solution Overview

An administrator can monitor the current system status through the main overview panel.

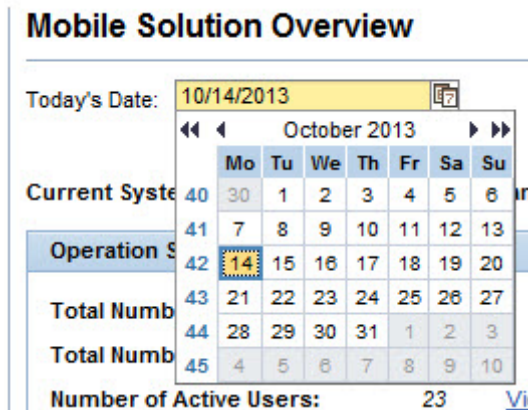
Mobile Solution Overview Panel



Today's Date: The date is automatically set to the current date when the Administration portal is first accessed. To view system status for past dates: click the calendar icon to the right of the date field and choose the desired date.

1. Click the calendar icon to the right of the date field and choose the desired date.

Mobile Solution Overview - Date Selection



2. Click the **Refresh** button to refresh the Operation Status statistics for the chosen date.
3. To revert back to the current date, select the calendar icon again and choose the current date, highlighted by the blue box. Then click the **Refresh** button to refresh the Operation Status statistics.

Current System Status (Checks of critical framework settings): Brief overview of the current system status through use of an icon and status text. Click on the **View Detail** hyperlink to view the System Status Detail table as shown in the following example.

Current System Status Detail

The System Status Detail table appears when the View Detail hyperlink to the right of the Current Systems Status (Checks of critical framework settings) line is clicked. This table provides an account of the framework and its status during the selected date.

Mobile Solution Overview - System Status Detail

System Status Detail					Close Message List	
Seq. No.	Status	System Status Message			Value	
7		Connection Test To Mobile App SAP_CRM_SERVICE_MANAGER_31 RFC Destination SSM31_CRM_SM_HOST			Failed	
9		Connection Test To Mobile App SAP_SALES_MANAGER_25 RFC Destination SSLM25_CRM_SD_HOST			Failed	
1		Configuration Framework System Setting			Exist	
2		Syclo Number Range Interval 01 for /SYCLO/C_1			Exist	
3		Syclo Number Range Interval 01 for /SYCLO/C_2			Exist	
4		Syclo Number Range Interval 02 for /SYCLO/C_2			Exist	
5		Syclo Number Range Interval 03 for /SYCLO/C_2			Exist	
6		Syclo Number Range Interval 04 for /SYCLO/C_2			Exist	
8		Connection Test To Mobile App SAP_CRM_SERVICE_MANAGER_40 RFC Destination SSM40_CRM_SM_HOST			Success	

- **Seq. No.:** Row number of the system tests performed
- **Status:** Icon display of the system status

- **System Status Message:** Text message giving a status description. There are three main system tests performed:
 - **Configuration framework system setting** - If this setting does not exist, or returns an error status message, applications will not be able to run on the framework.
 - **Syclo number range interval** - Automatic number counter. If this is not set up during the system installation, the counters will not work. See the installation guide for more details.
 - **Connection test** - This test is dynamic and only appears if the system is multi-back end enabled. For the system to post a Success value for this test, the RFC destinations must be defined.
- **Value:** Text status of the system test

Click the **Close Message List** button to close the table view.

Mobile Solution Overview - Operation Status

The Operation Status panel of the Mobile Solution Overview provides a high-level status of activity on the system. Clicking on the View Detail link to the right of each status displays a more detailed status table below the Operation Status panel.

To access operation status details for a different date, choose a different date using the calendar icon at the top of the panel. Be sure to click **Refresh** in order to refresh the panel statistics to the newly-selected date.

Note: If statistics collection is not enabled in SAP, some operation status details will not be available through the Administration & Monitoring portal.

Mobile Solution Overview - Operation Status

Operation Status					
Total Number of Mobile Applications:		8	View Detail	Total Number of Middleware Servers:	
Total Number of Mobile Users:		123	View Detail	Total Number of Communication Sessions:	
Number of Active Users:		23	View Detail	Number of Users With Exception:	
Number of Push Instances:		43	View Detail		

Total Number of Mobile Applications

Lists the number of mobile applications in use on the system.

Mobile Solution Overview - Mobile Application Count Detail

Mobile Application Count Detail				
Close Message List				
	Seq. No.	Status	Mobile Application	Value
	1		SAP_CRM_SERVICE_MANAGER_31, 3.1.0	1
	2		SAP_CRM_SERVICE_MANAGER_40, 4.0.0	1
	3		SAP_FLIGHT_MANAGER_10, 1.0.0	1
	4		SAP_INVENTORY_MANAGER_32, 3.2.0	1
	5		SAP_ROUNDS_MANAGER_23, 2.3.0	1
	6		SAP_SALES_MANAGER_25, 2.5	1
	7		SAP_WORK_MANAGER_53, 5.3.0	1
	8		SAP_WORK_MANAGER_60, 6.0.0	1

- **Seq. No.:** Row number in the table
- **Status:** Not used in this table
- **Mobile Application:** Name and release of the mobile application
- **Value:** Number of specific mobile applications in use on the system

Total Number of Mobile Users

Lists the total number of mobile users on the system on the date chosen.

Mobile Solution Overview - Mobile User Count Detail

Mobile User Count Detail				
Close Message List				
	Seq. No.	Status	Mobile Application	Value
	1		SAP_CRM_SERVICE_MANAGER_40	13
	2		SAP_WORK_MANAGER_53	23
	3		SAP_WORK_MANAGER_60	87

- **Seq. No.:** Row number in the table
- **Status:** Not used in this table
- **Mobile Application:** Name of the mobile application
- **Value:** Number of active users on the specific mobile application

Number of Active Users

Lists the total number of active users on the system on the date chosen.

Mobile Solution Overview - Active User Count Detail

Active User Count Detail				
Close Message List				
	Seq. No.	Status	Mobile Application	Value
	1		SAP_CRM_SERVICE_MANAGER_40	3
	2		SAP_WORK_MANAGER_53	2
	3		SAP_WORK_MANAGER_60	18

- **Seq. No.:** Row number in the table
- **Status:** Not used in this table
- **Mobile Application:** Name of the mobile application
- **Value:** Number of active users on the mobile application

Number of Push Instances

Lists the total number of pushes initiated on the system on the date chosen.

Mobile Solution Overview - Push Instance Count Detail

Push Instance Count Detail				
Close Message List				
	Seq. No.	Status	Mobile Application, Scenario Id, Push Status	Value
	1		SAP_WORK_MANAGER_53, SWM53_EMERGENCY_WORKORDER_PUSH, COMPLETED	10
	2		SAP_WORK_MANAGER_53, SWM53_EMERGENCY_WORKORDER_PUSH, PROCESS	1
	3		SAP_WORK_MANAGER_60, SWM60_BDS_DOCUMENT_PUSH, COMPLETED	16
	4		SAP_WORK_MANAGER_60, SWM60_BDS_DOCUMENT_PUSH, PROCESS	4
	5		SAP_WORK_MANAGER_60, SWM60_DMS_DOCUMENT_PUSH, COMPLETED	1
	6		SAP_WORK_MANAGER_60, SWM60_EMERGENCY_WORKORDER_PUSH, COMPLETED	11

- **Seq. No.:** Row number in the table

- **Status:** Not used in this table
- **Mobile Application, Scenario ID, Push Status:** Name of the mobile application, name of the push instance, and status of the push instance
- **Value:** Number of push instances initiated on each mobile application

Total Number of Middleware Servers

Lists the total number of middleware servers present on the system.

Mobile Solution Overview - Middleware Server Count Detail

Middleware Server Count Detail				
				Close Message List
	Seq. No.	Status	Server:Port, Serial Number	Value
	55		DEWDFGWD00689:00000, SMP	1
	56		CHIN00533675A:00000, AKJBGDBFFEIGAJKA	1
	57		DRSN00306878A:00000, AKJBGDBFFEIGAJKA	1
	58		DEWDFWSSP2415:00000, AKJBGDBFFEIGAJKA	1
	59		DEWDFWSSP2415:00000, SMP	1
	60		CHIV00545427B:00000, AKJBGDBFFEIGAJKA	1
	61		CHIN00533366A:00000, ALJL@ECFHIM@HUNF	1
	62		DEWDFGWD00691:00000, SAP_WORK_MANAGER_60	1
	63		DEWDFGWD00691:00000, SMP	1
	64		CHIN00533808A:00000, AKJBGDBFFEIGAJKA	1

- **Seq. No.:** Row number in the table
- **Status:** Not used in this table
- **Server: Port, Serial Number:** Name of the middleware server, the port it uses to connect to the framework, and the serial number assigned to it in SAP.
- **Value:** Number of middleware servers with that server name, port and serial number. This number will always be 1.

Total Number of Communication Sessions

Lists the total number of communication sessions for the date chosen.

Mobile Solution Overview - Communication Session Count Detail

Communication Session Count Detail				
				Close Message List
	Seq. No.	Status	Server:Port, Mobile Application	Value
	1		CHIN00533283A:00000, SAP_WORK_MANAGER_60	11
	2		CHIN00533339A:00000, SAP_WORK_MANAGER_60	2
	3		CHIN00533788A:00000, SAP_WORK_MANAGER_60	42
	4		DEWDFGWD00692:00000, SAP_WORK_MANAGER_60	96
	5		DEWDFGWP01270:00000, SAP_WORK_MANAGER_60	141
	6		TEST_SERVER:07003, SAP_CRM_SERVICE_MANAGER_40	8
	7		TEST_SERVER:07003, SAP_WORK_MANAGER_53	2
	8		TEST_SERVER:07003, SAP_WORK_MANAGER_60	209
	9		YMQN00538863A:00000, SAP_WORK_MANAGER_60	8

- **Seq. No.:** Row number in the table
- **Status:** Not used in this table
- **Server: Port, Mobile Application:** Name of the server, the port it uses to connect to the framework, and the mobile application used on the server for the communication sessions
- **Value:** Number of communication sessions on the specific server listed

Number of Users with Exception

If a user encounters issues during a transmit, the user ID, the mobile application, and the application ID is logged.

Mobile Solution Overview - User Exception Count Detail

User Exception Count Detail				
				Close Message List
	Seq. No.	Status	Mobile Application	Value
	1		SAP_WORK_MANAGER_60, 005056BA41F31ED382D38AA140779985, POPEM1	2
	2		SAP_WORK_MANAGER_60, 005056BA41F31EE2BE8FCD58A8305944, AGUSER1	7
	3		SAP_WORK_MANAGER_60, 005056BA41F31EE2BE90455C8E3119B9, CHILOVICH	3
	4		SAP_WORK_MANAGER_60, 005056BA41F31EE384B61081D3029964, AGUSER10	2
	5		SAP_WORK_MANAGER_60, 005056BA41F31EE38CD10D5230DF117E, CROWED	3
	6		SAP_WORK_MANAGER_60, 005056BA41F31EE38CD11E914BC2918B, KOEFELDA	3

- **Seq. No.:** Row number in the table
- **Status:** Visual indicator showing the severity of the exception
- **Mobile Application:** Name of the mobile application, GUID of the mobile application, and user name causing the exception
- **Value:** Number of exceptions the listed user has on the listed mobile application

Administration Portal - Administration

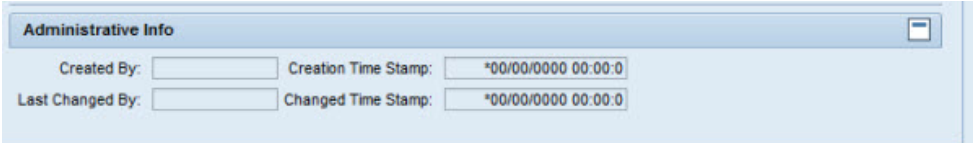
The Administration panel is used to create and manage the following three areas:

- **User Management** - System administrators can create new mobile user IDs, as well as manage all settings associated with specific user IDs.
- **Server Management** - System administrators can create new connections to new servers, as well as manage all settings associated with specific servers.
- **Runtime Logging Level Setting** - System administrators can create or modify new logging parameters associated with specific user GUIDs.
- **Mobile Application Parameters** - Use the Mobile Application Parameter Management panel to override parameter settings in a production environment for a specific user or for the whole application, depending on the selection you make for the parameter scope.

Administrative Info

Each management panel contains a section at the bottom of the page called Administrative Info. Expand the section by clicking on the white box icon to the right of the Administrative Info heading.

Administration Portal - Administrative Info



- **Created By:** SAP user ID of the person who created the information
- **Creation Time Stamp:** Time and date the information was created
- **Last Changed By:** SAP user ID of the person who modified information in the panel
- **Changed Time Stamp:** Time and date the information was modified

Administration - User Management

The Middleware User Management panel is used to create and manage mobile client user IDs. These IDs are layered on top of the user's SAP user ID.

A user must have an SAP user ID before being able to access any mobile applications. A user must also have a separate user GUID for each mobile application they access. In this way, the system can track and calculate exchange episodes for each application a user accesses while keeping the exchanges separated by mobile application.

Administration - Middleware User Management

Middleware User Management

Search Mobile Users

Basic Search Parameters

Mobile Application: * SAP Work Manager 6.0.0 with LAM User: WANGJIR
 Last Activity Time: All

Search Result

View: [Standard View] | Print Version | Export

User Id	Mobile App	Server Serial No	Lock Flag	Created On
WANGJIR	SAP_WORK_MANAGER_60	AKJBGDBFFEIGAJKA		08/16/2013 18:01:35

Mobile User Detail (Display Mode)

Change

Basic Info | Cross Reference List | Client Registration Info

Mobile Application: SAP Work Manager 6.0.0 with LAM User GUID: 005056BA41F31ED381D1A40DDCD
 User Name: WANGJIR SAP Personnel No: 00000000
 Middleware License No: AKJBGDBFFEIGAJKA
 Device Id: Device User Id: WANGJIR
 Group Id:
 E-Mail Address:
 HTTP Address:
 Default Address Type: Source System:
 Lock Flag: ☐

Administrative Info

Created By: WANGJIR Creation Time Stamp: 08/16/2013 18:01:35
 Last Changed By: VELPULA Changed Time Stamp: 10/04/2013 03:43:44

Basic Search Parameters

Use this field to search for users on a specific mobile application, or refine the search even further by the user ID or last activity time. Once specific user(s) are found, their information can be changed if necessary.

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application. This is a required field to enable the Search function.
- **User:** SAP user ID
- **Last Activity Time:** Use the drop-down menu to select a window of time. The default is set to All, which is equal to selecting no specific time period.
- **Search:** Click the **Search** button once after you have entered all required and additional search parameters. If the search returns results, they are displayed in the Search Result section. If the search parameters are not fulfilled, a message stating “No data found” displays in the Search Result section.

Search Result

When you click the **Search** button in the Basic Search Parameters section, the Search Result table is populated. You can also create a new user in this section using the **Create** button. In this case, it is not necessary to search for an existing user.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet.
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Mobile User Detail - Basic Info

When a user is selected in the Search Results table, the fields are populated in the Basic Info section. Click **Change** to modify editable fields. Click **Create** in the Search Result section to create a new user through the editable fields. Fields with a red asterisk beside them are mandatory.

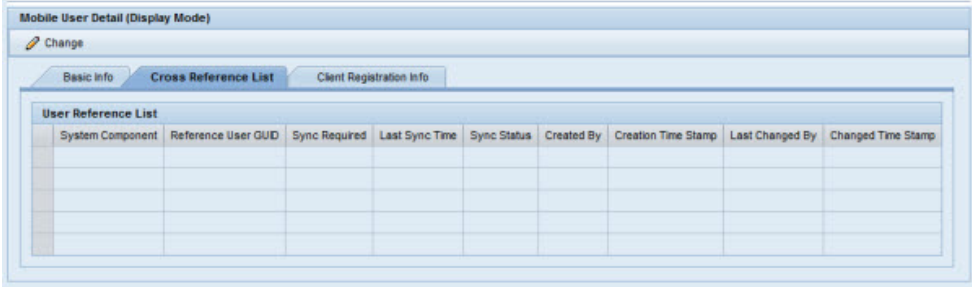
- **Mobile Application:** Name of the mobile application chosen in the Basic Search Parameters section
- **User GUID:** User GUID for the specific mobile application chosen. Each user has a different GUID for each mobile application assigned in SAP.
- **User Name:** User ID chosen from the table in the Search Result section
- **SAP Personnel Number:** User ID number assigned to the user in SAP
- **Middleware License Number:** Server serial number of the middleware server that is running the chosen mobile application
- **Device ID:** If a user is assigned to a mobile device connected to the chosen application and the mobile device has an ID, the field is populated.
- **Device User ID:** User ID assigned to the mobile device
- **Group ID:** Allows you to define additional information about a user, such as a crew number or team name. This is a free-text field.
- **Email Address:** Email address of the user. Email are sent through the Push Scenario Definition panel in the Configuration portal. Email are also used in the User Monitor panel under Monitoring in the Administration portal.

- **HTTP Address:** Web address or site links that users can access, such as a Twitter feed
- **Default Address Type:** Select either Email Address or HTTP Address from the drop-down menu for the default email sending method
- **Source System:** Originating SAP system of the user ID. This is useful in multi-backend systems.
- **Lock Flag:** If checked, user is unable to access the mobile application from the mobile device. This is used if a mobile device is lost or stolen and the application data must be made inaccessible.

Mobile User Detail - Cross Reference List

The Cross Reference list is used in systems that contain multi-back ends. When a user is created or modified, that user's information will need to be replicated across the systems. This detail view provides statistics and information for each user on a multi-back end system.

Figure 14: Mobile User Detail - Cross Reference List



User Reference List								
System Component	Reference User GUID	Sync Required	Last Sync Time	Sync Status	Created By	Creation Time Stamp	Last Changed By	Changed Time Stamp

- **System Component:** System where the user ID resides
- **Reference User GUID:** SAP user GUID
- **Sync Required:** If box is checked, synchronization is required
- **Last Sync Time:** Time and date of last synchronization
- **Sync Status:** Descriptive text detailing the synchronization status
- **Created By:** SAP ID of the user who created the new user ID
- **Creation Time Stamp:** Time and date of the
- **Last Changed By:** SAP ID of the user who made the change
- **Changed Time Stamp:** Time and date of last change

Mobile User Detail - Client Registration Info

The Client Registration Info tab is used for RIM products only.

Figure 15: Mobile User Detail - Client Registration Info

Mobile User Detail (Display Mode)

Change

Basic Info Cross Reference List Client Registration Info

User Name: IWANGJIR User GUID: 005056BA41F31ED381D1A40DCDE4

Device Id Registered: Client Definition Name: Client Definition Revision No: 0

Last Communication Session Id: 005056BA41F31EE38C8C950FED07B Middleware Server: CHN00533391A

Last Registered On: 00/00/0000 00:00:00 Registered By:

Administration - Server Management

The Middleware Server Management panel is used to create and manage the middleware servers on the system.

Administration - Middleware Server Management

Middleware Server Management

Search Middleware Servers

Basic Search Parameters

Mobile Application: SAP Work Manager 6.0.0 with LAM

Server Name: Server Port: 00000

Serial No:

Search

Search Result

View: [Standard View] Print Version Export

Server Name	Port	Serial No.	Mobile App	Lock Flag	Disable Outb. Trigger	Created On
DEWDFGWD00692		AKJBGDBFFEIGAJKA	SAP_WORK_MANAGER_60			07/30/2013 16:48:39
TEST_SERVER	7003	SAP_SIMULATION	SAP_WORK_MANAGER_60			07/30/2013 18:43:03
DEWDFGWD00703		SMP	SAP_WORK_MANAGER_60			08/02/2013 21:29:53
CHN00533788A		ALINGFIKKJALNVVO	SAP_WORK_MANAGER_60			08/05/2013 17:42:37
CHN00533391A		AKJBGDBFFEIGAJKA	SAP_WORK_MANAGER_60			08/16/2013 20:45:43

Create

Middleware Server Detail (Display Mode)

Change

Basic Info

Mobile Application: SAP Work Manager 6.0.0 with LAM Server GUID: 005056BA41F31ED2BEA4DAEA21B4

Server Name: DEWDFGWD00692 Port: 00000

Middleware Svr SerNo: AKJBGDBFFEIGAJKA

Server URL (FQDN): dewdfgwd00692.wdf.sap.corp

Target Host: 10.72.178.33

Local Outbound Trigger Port: 00000

Outbond Trigger URL Type: Use FQDN when available

Lock Flag: ☐

Disabled for Outbound Trigger: ☐

Administrative Info

Basic Search Parameters

Use this field to search for users on a specific mobile application, or refine the search even further by the user ID or last activity time. Once specific user(s) are found, their information can be changed if necessary.

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application. This is a required field to enable the Search function.
- **Server Name:** Name given to the middleware server either through SAP or when creating a new server configuration in this panel.
- **Server Port:** Port on the server assigned to the selected mobile application
- **Serial Number:** The serial number of the server assigned to the mobile application
- **Search:** Click **Search** once all required and additional search parameters are filled. If the search returns results, they are displayed in the Search Result section. If the search parameters are not fulfilled, a message stating “No data found” displays in the Search Result section.

Search Result

When you click **Search** in the Basic Search Parameters section, the Search Result table is populated. You can also create a new server in this section using **Create**. In this case, it is not necessary to search for an existing server.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet.
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click on the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click on Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Middleware Server Detail - Basic Info

When you select a server in the Search Results table, the fields are populated in the Basic Info section. Click **Change** to modify editable fields. Click **Create** in the Search Result section to create a new server by populating the editable fields. Fields with a red asterisk beside them are mandatory.

- **Mobile Application:** Name of the mobile application chosen in the Basic Search Parameters section
- **Server GUID:** Server GUID for the specific mobile application chosen. Each server has a different GUID for each mobile application assigned in SAP.
- **Server Name:** Server name chosen from the table in the Search Result section
- **Port:** Port the Agentry application in the SAP Mobile Server has assigned to the specific mobile application chosen
- **Middleware Svr (Server) SerNo (Serial Number):** Customer license number
- **Server URL (FQDN):** Fully qualified domain name that can identify the Agentry application defined in the SAP Mobile Server. In this way, SAP can identify and broadcast to the middleware server without using an IP address. If the IP address of the middleware server changes, use this method as the preferred way to communicate.
- **Target Host:** If a target host is specified, a broadcast of push notification is sent to the IP or URL specified in the Target Host field, instead of to the middleware server's own IP or URL. Use this if a dedicated middleware or server should receive the broadcast from SAP.
- **Local Outbound Trigger Port:** By default, each middleware server running the same mobile application use the same port number to receive push notification broadcasts from SAP. This is the default in the outbound trigger setting Target Host Port Number. If, for some reason, individual middleware servers listen to push notifications on a different port, specify the port number here. The listed port number in this field will then supercede the port number set in Outbound Trigger Settings.
- **Outbound Trigger URL Type:** Use either the IP address or server URL when broadcasting push notifications.
- **Lock Flag:** If checked, the server will no longer accept users attempting to connect. Set the lock flag to temporarily lock down a server without deleting it from the system.
- **Disabled for Outbound Trigger:** If checked, the Outbound Trigger is unavailable for the selected server.

Administration - Runtime Logging Level Setting

The Runtime Logging Level Setting panel is used to create and manage logging levels for specific mobile applications and user GUIDs. These settings override the logging level setting defined in the Technical Settings panel of the Configuration portal. The logging level setting defined in the Configuration portal applies to all applications running on the framework. However, at times more detail on a specific user or mobile application is desired without making changes to the entire framework.

With the runtime logging level setting, administrators can make dynamic adjustments to the logging level settings. Overrides to the framework settings can be made in two different areas: an individual mobile application or individual users running a specific mobile application. When troubleshooting of the user or the mobile application is complete, the administrator can reset the logging levels back to the default settings specified in the Configuration portal by un-checking the active flag.

Administration - Runtime Parameter Management

Runtime Parameter Management -

Search Runtime Parameters

Basic Search Parameters

Mobile Application: * SAP Work Manager 6.0.0 with LAM User GUID:

Param. Name:

Search Result

View: [Standard View] Print Version Export

Parameter Name	User Id	User GUID	Parameter Value	Active Flag	Effective Time	Duration
No data found						

Runtime Parameter Detail (Display Mode)

Basic Info

Mobile Application: * User GUID:

SAP User Id:

Param. Name: Param. Group:

Param. Value:

Active Flag: ☐

Effective Date: Time: 00:00:00

Duration (Hrs): 0

Administrative Info

Basic Search Parameters

Use these fields to search for logging parameters on a specific mobile application, or refine the search even further by the user ID. Once specific logging parameters are found, their information can be changed if necessary.

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application. This is a required field to enable the search function.
- **User GUID:** User GUID associated with the selected mobile application
- **Param Name:** Name of the selected parameter. Parameters are set and configured in the Configuration portal.
- **Search:** Click **Search** once all required and additional search parameters are set. If the search returns results, they are displayed in the Search Result section. If the search parameters are not fulfilled, a message stating “No data found” displays in the Search Result section.

Search Result

When you click **Search** in the Basic Search Parameters section, the Search Result table is populated. You can also create a new runtime parameter in this section using the **Create** button. In this case, it is not necessary to search for an existing runtime parameter.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click on the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click on Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Runtime Parameter Detail - Basic Info

- **Mobile Application:** Name of the mobile application chosen in the Basic Search Parameters section
- **User GUID:** User GUID for the specific mobile application chosen. Each user has a different GUID for each mobile application assigned in SAP.
- **SAP User ID:** User ID assigned in SAP
- **Param. Name:** Name assigned to the logging parameter
- **Parameter Group:** A non-editable field that identifies the system category of the parameter group
- **Param. Value:** A numerical value corresponding to logging levels:
 - 0 - No logging
 - 1 - Abort
 - 2 - Error
 - 3 - Warning
 - 4 - Info
 - 5 - Debug
 - 6 - Trace
- **Active Flag:** Check this box in order to enable the logging levels set through this panel. Uncheck to disable the logging levels set through this panel and to return to the logging levels set through the Technical Settings panel in the Configuration Panel.

- **Effective Date / Time:** Effective date and time of the logging level change
- **Duration (Hrs):** Duration, in hours, of the logging level change

Administration Portal - Settings

All of the screens in the Administration portal contain a Settings link in the Search Result section. Configuring the settings for results display can assist in filtering and sorting user data, especially if there are many user results returned in a search.

To access the Settings section, click on the Settings link at the top right of the Search Result section. This opens the Settings view.

Note: Each of the portal screens requires individual setup of the settings.

- **View:** If multiple views are saved, use the drop-down menu to select the appropriate view to change.

Use the **Save As** button to save the particular configuration created or modified with a descriptive title. Use the View drop-down menu in the Search Result section to select the desired view. The created view is available in all Settings tabs.

If a view is no longer needed, select the view in the drop-down menu and click **Delete**.

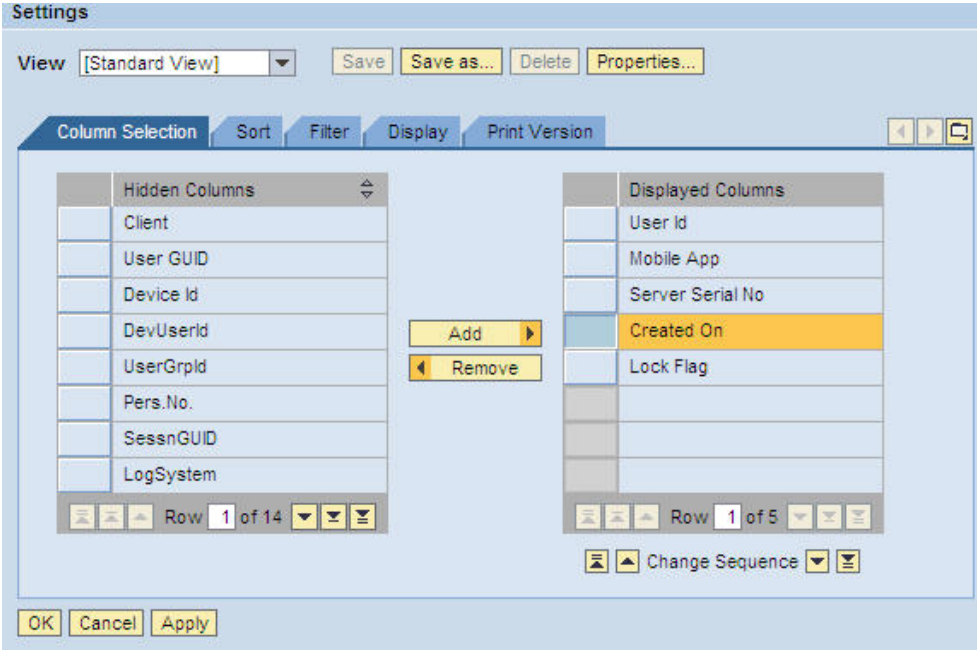
To display view properties or to rename the view, click the Properties button.

- **OK:** When done with configuration in the tabs, click **OK** to change the settings in the Search Result table or test the printing configuration. Clicking **OK** does not commit the changes permanently.
- **Cancel:** Click **Cancel** to cancel any changes made during the configuration session.
- **Apply:** Click **Apply** when all configuration is finished and tested. Clicking **Apply** commits the changes so they are available from session to session.

Column Selection

The Column Selection tab configures the columns and their results that are displayed in the search result table.

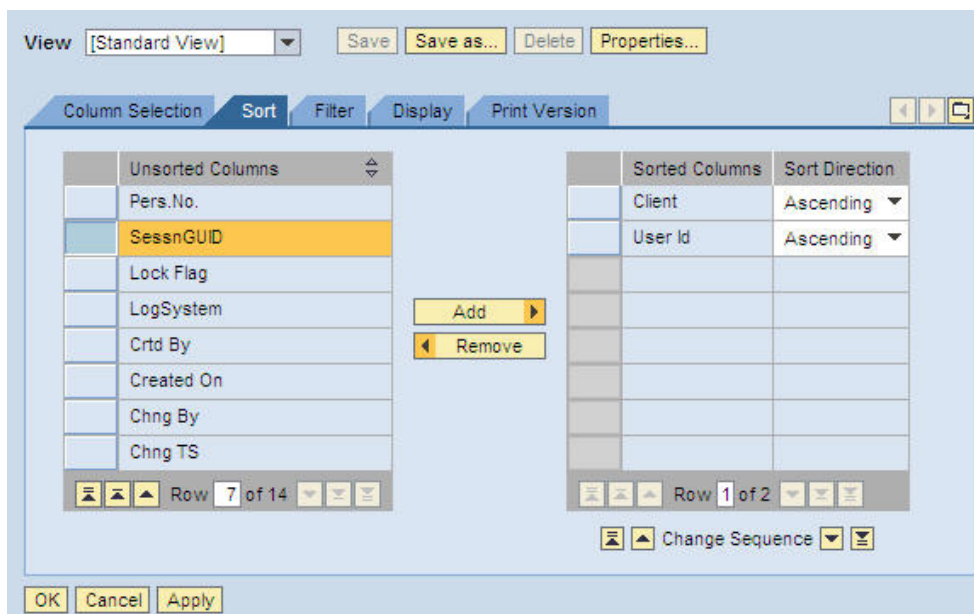
Management Settings - Column Selection



Sort

At times a search performed by using the standard Basic Search Parameters can result in a large amount of information displayed. Use the Sort tab to determine which columns are used to sort the information, depending on the needs of the administrator. The sorting function works in that the first row in the Sorted Columns is the primary source for sorting. Each additional row refines the sorting further. For instance, if a user is active on more than one server, sort by user name and then Server Serial No to display that user's activity in order of the serial number.

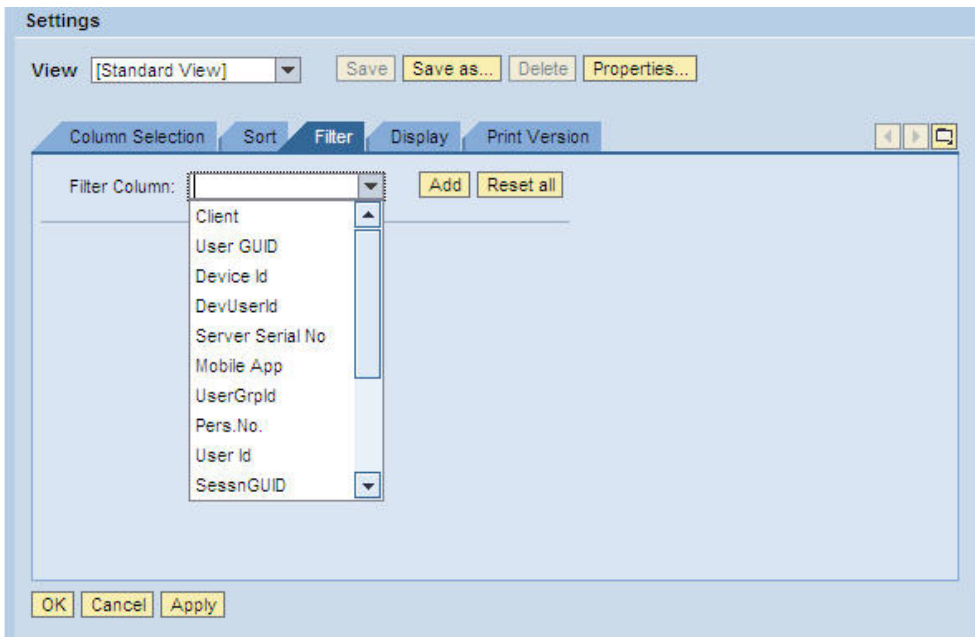
Management Settings - Sort



Filter

Administrators can create different filters in order obtain a more detailed view from the search results. When the filter is no longer needed on the search results table, click the Delete Filter link.

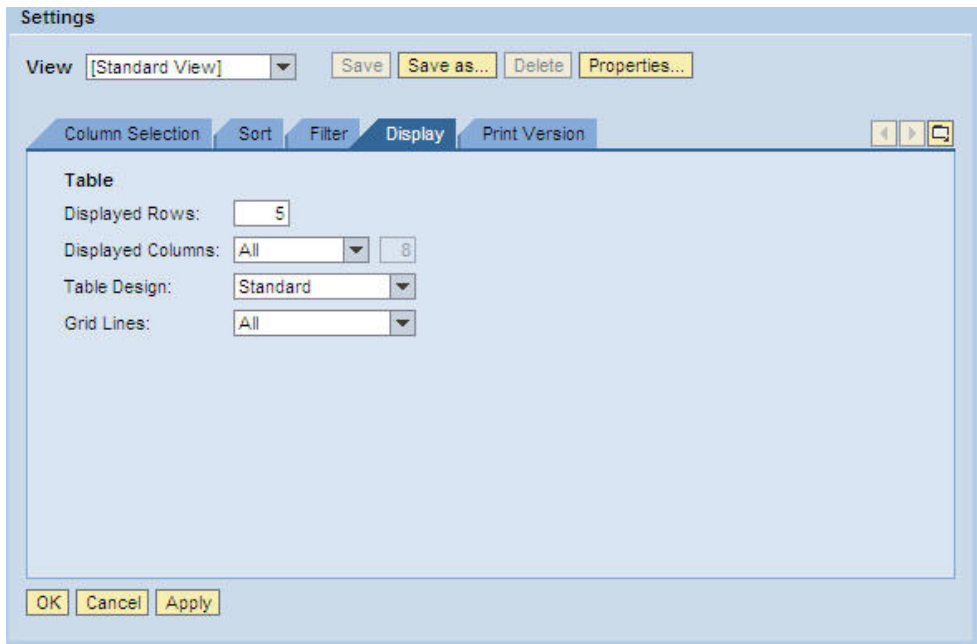
Management Settings - Filter



Display

Use the Display tab to control how the table in the Search Result section displays.

Management Settings - Display



Print Version

The Print Version tab configures specific print settings.

Management Settings - Print Version

Administration Portal - Monitoring

The Monitoring panel is used to monitor the following areas:

- User Monitor
- Push Instance Monitor
- Communication Session Monitor
- Object Mobile Status Monitor
- Mobile Transaction History Monitor
- Subscription Queue Monitor

The Monitoring panels are only used for monitoring users and activity. An administrator cannot create new information or change existing information through these panels. To create or modify information, navigate to the appropriate panel in the portal and save the changes.

Administration Portal - Settings

All of the screens in the Administration portal contain a Settings link in the Search Result section. Configuring the settings for results display can assist in filtering and sorting user data, especially if there are many user results returned in a search.

To access the Settings section, click on the Settings link at the top right of the Search Result section. This opens the Settings view.

Note: Each of the portal screens requires individual setup of the settings.

- **View:** If multiple views are saved, use the drop-down menu to select the appropriate view to change.
Use the **Save As** button to save the particular configuration created or modified with a descriptive title. Use the View drop-down menu in the Search Result section to select the desired view. The created view is available in all Settings tabs.
If a view is no longer needed, select the view in the drop-down menu and click **Delete**.
To display view properties or to rename the view, click the Properties button.
- **OK:** When done with configuration in the tabs, click **OK** to change the settings in the Search Result table or test the printing configuration. Clicking **OK** does not commit the changes permanently.
- **Cancel:** Click **Cancel** to cancel any changes made during the configuration session.
- **Apply:** Click **Apply** when all configuration is finished and tested. Clicking **Apply** commits the changes so they are available from session to session.

Column Selection

The Column Selection tab configures the columns and their results that are displayed in the search result table.

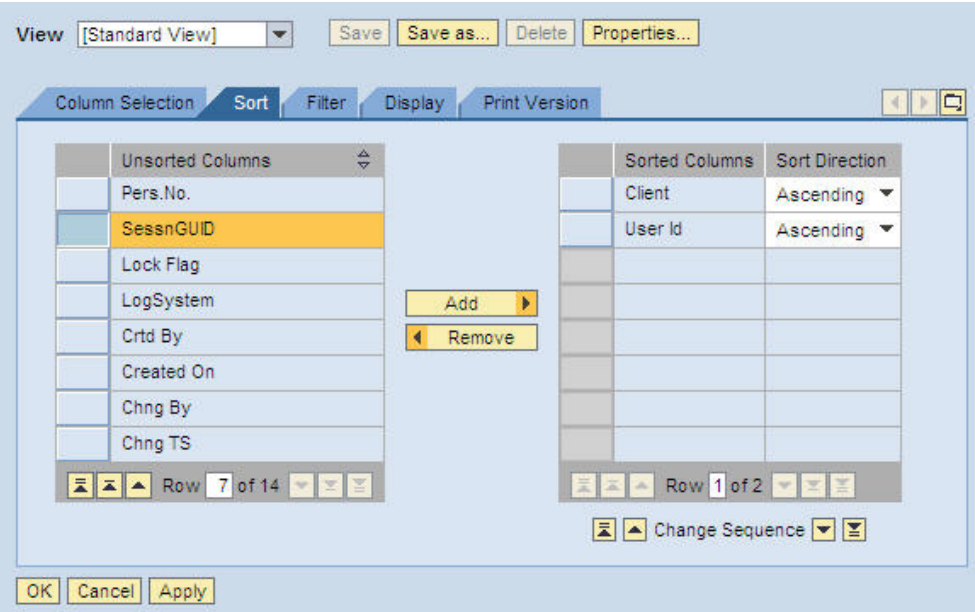
Management Settings - Column Selection

The screenshot shows the 'Settings' dialog box with the 'Column Selection' tab active. The 'View' dropdown is set to '[Standard View]'. The 'Column Selection' tab is selected, showing two lists: 'Hidden Columns' and 'Displayed Columns'. The 'Hidden Columns' list contains: Client, User GUID, Device Id, DevUserId, UserGrpId, Pers.No., SessnGUID, and LogSystem. The 'Displayed Columns' list contains: User Id, Mobile App, Server Serial No, Created On (highlighted), and Lock Flag. There are 'Add' and 'Remove' buttons between the lists. At the bottom are 'OK', 'Cancel', and 'Apply' buttons.

Sort

At times a search performed by using the standard Basic Search Parameters can result in a large amount of information displayed. Use the Sort tab to determine which columns are used to sort the information, depending on the needs of the administrator. The sorting function works in that the first row in the Sorted Columns is the primary source for sorting. Each additional row refines the sorting further. For instance, if a user is active on more than one server, sort by user name and then Server Serial No to display that user’s activity in order of the serial number.

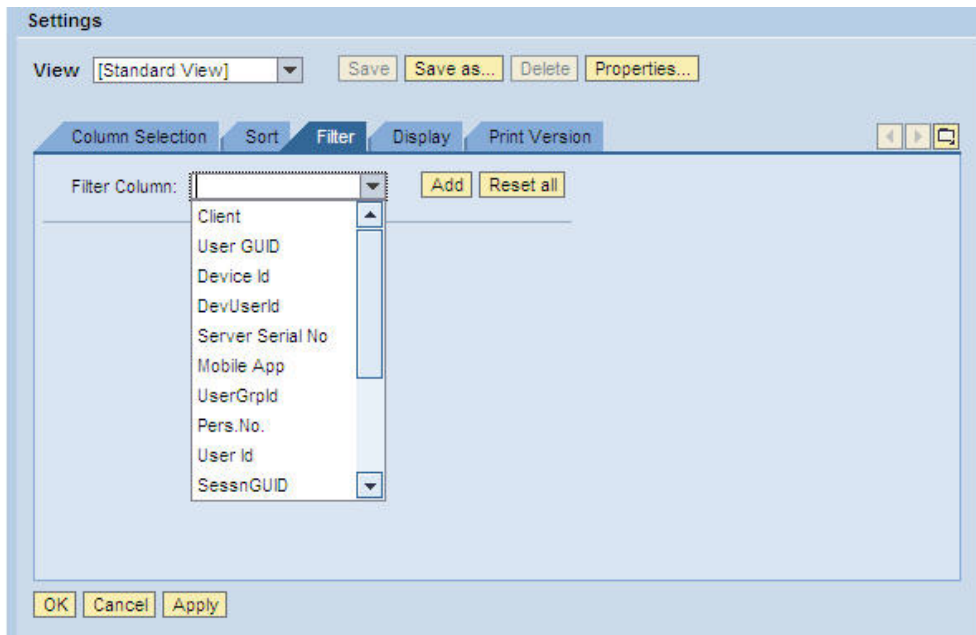
Management Settings - Sort



Filter

Administrators can create different filters in order obtain a more detailed view from the search results. When the filter is no longer needed on the search results table, click the Delete Filter link.

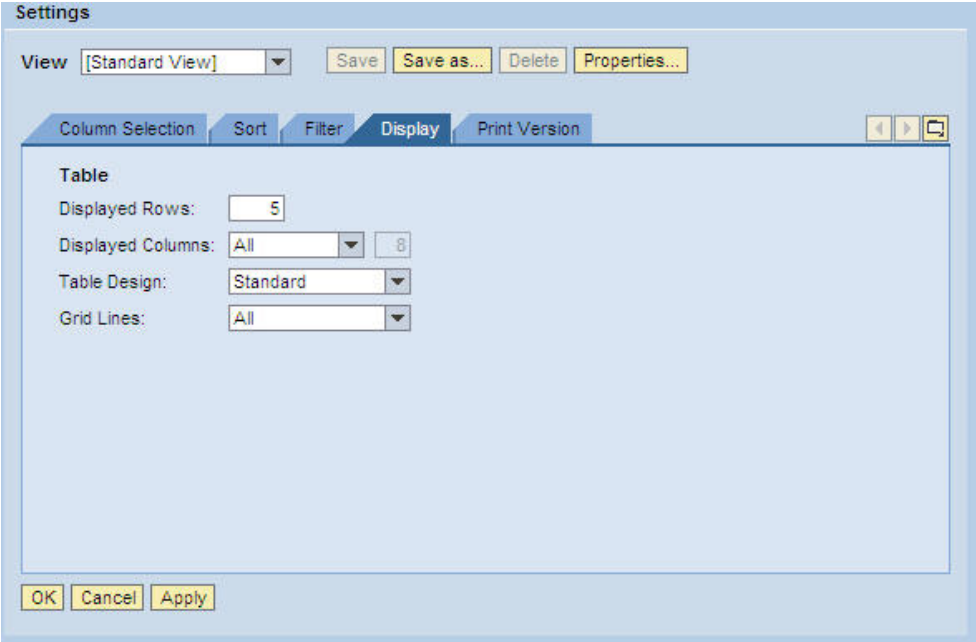
Management Settings - Filter



Display

Use the Display tab to control how the table in the Search Result section displays.

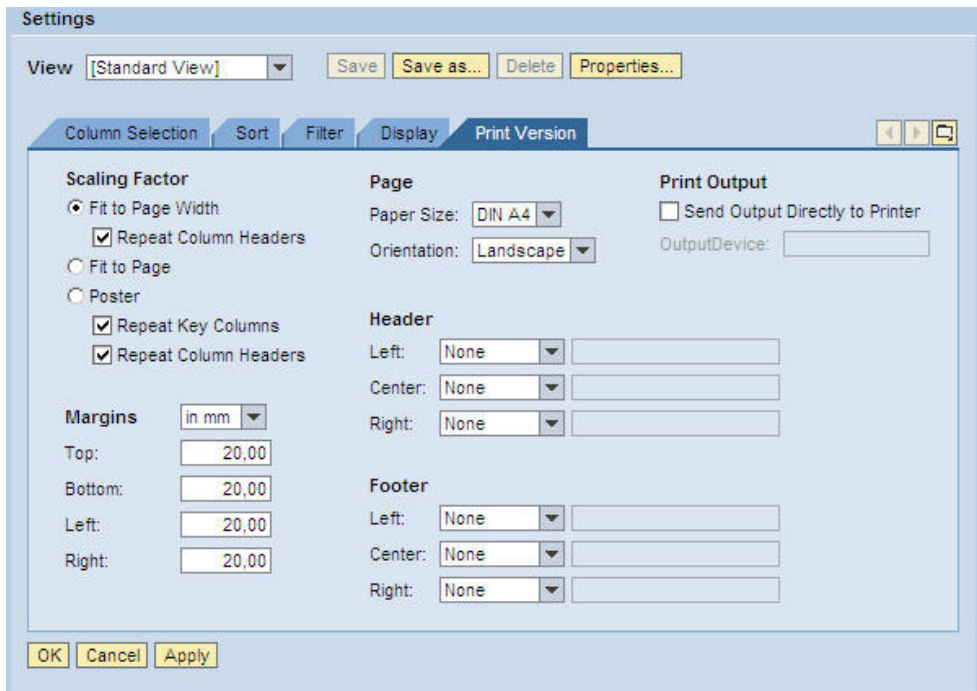
Management Settings - Display



Print Version

The Print Version tab configures specific print settings.

Management Settings - Print Version



Monitoring - User Monitor

The User Monitor panel allows an administrator to locate and view specific users and mobile application use. An administrator can also send specific users email to their mobile devices or HTTP addresses through this panel.

Note: The Client Registration Info tab is for RIM products only and will not be covered in this manual.

Monitoring - Mobile User Monitoring

Mobile User Monitoring

Search Mobile Users

Basic Search Parameters

Mobile Application: SAP Work Manager 6.0.0 with LAM User: Last Activity Time: All Search Send Email

Search Result

View: [Standard View] Print Version Export

User Id	Mobile App	Server Serial No	Lock Flag	Created On
SINGHPRIT	SAP_WORK_MANAGER_60	AKJBGBFFEGAJKA		08/19/2013 21:52:38
SMERPERVUSR	SAP_WORK_MANAGER_60	AKJBGBFFEGAJKA		09/16/2013 20:45:43
TALLUTO	SAP_WORK_MANAGER_60	AKJBGBFFEGAJKA		10/07/2013 16:27:34
VELPULA	SAP_WORK_MANAGER_60	ALJNGFKKJALNVVO		08/05/2013 18:04:18
VELPULA	SAP_WORK_MANAGER_60	AKJBGBFFEGAJKA		09/03/2013 17:44:45
WANGUR	SAP_WORK_MANAGER_60	AKJBGBFFEGAJKA		08/16/2013 18:01:35
YALAMATIS	SAP_WORK_MANAGER_60	SMP		10/09/2013 09:22:40
YALAMATIS	SAP_WORK_MANAGER_60	ALJNGFKKJALNVVO		10/04/2013 21:42:22
YALAMATIS	SAP_WORK_MANAGER_60	AKJBGBFFEGAJKA		09/28/2013 22:25:36
YALAMATIS	SAP_WORK_MANAGER_60	TEST_SERVER		09/18/2013 08:38:53

Mobile User Detail

Refresh

General Info Client Object Info Cross Reference List Outbound Message Queue Communication Sessions Client Registration Info

User Name: TALLUTO User GUID: 005056BA41F31ED38BE99373A5ACF

Personnel No.: 00000000 Middleware UserOrpId:

Lock Flag: ☐

Mobile Application: SAP_WORK_MANAGER_60 Middleware Svr Serial: AKJBGBFFEGAJKA

Created By: TALLUTO Creation Time Stamp: 10/07/2013 16:27:34

Last Changed By: TALLUTO Changed Time Stamp: 10/07/2013 16:27:34

Basic Search Parameters

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application.
- **User:** SAP user ID. Manually type in the SAP user ID or click the white box icon to the right of the field for optional search methods.
- **Last Activity Time:** Use the drop-down menu to select a window of time. The default is set to All, which is equal to selecting all activity times. All times are available unless the historical logs have been purged in SAP.
- **User GUID:** User GUID assigned to their mobile device.
- **Search:** Click **Search** to initiate the search process. If no results are valid in the search parameters, the message No Data Found displays in the Search Results table. If valid data is returned, it displays in the Search Results table according to the Settings and Filter setup configured by the administrator.
- **Send Email:** Press to start the Send System Emails action.

Search Result

When you click **Search** in the Basic Search Parameters section, the Search Result table is populated. Click on the rectangle to the left of the first column in the desired row to populate the tabs in the Mobile User Detail section below.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Mobile User Detail - General Info

The General Info tab displays detailed information about the specific user highlighted in the Search Results table.

Mobile User Detail - General Info

The screenshot shows the 'Mobile User Detail' window with the 'General Info' tab selected. The window has a 'Refresh' button and several tabs: 'General Info', 'Client Object Info', 'Cross Reference List', 'Outbound Message Queue', 'Communication Sessions', and 'Client Registration Info'. The 'General Info' tab contains a form with the following fields and values:

User Name:	TALLUTO	User GUID:	005056BA41F31ED38BE9373A0ADF
Personnel No.:	00000000	Middleware UserGrpid:	
Lock Flag:	<input type="checkbox"/>		
Mobile Application:	SAP_WORK_MANAGER_60	Middleware Svr SerId:	AKJBGDBFFKGAJKA
Created By:	TALLUTO	Creation Time Stamp:	10/07/2013 16:27:34
Last Changed By:	TALLUTO	Changed Time Stamp:	10/07/2013 16:27:34

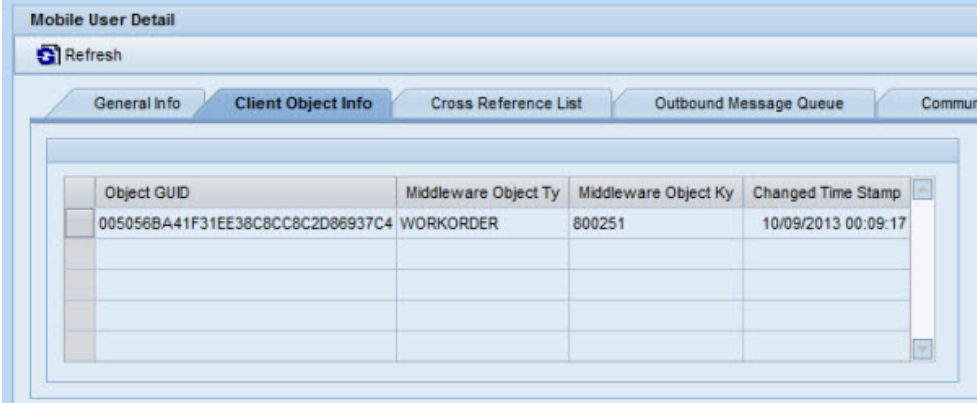
- **User Name:** SAP User ID
- **User GUID:** User GUID for the specific mobile application chosen. Each user has a different GUID for each mobile application assigned in SAP.
- **SAP Personnel Number:** User ID number assigned to the user in SAP
- **Middleware User Group ID:** Group ID assigned to the specific user through the User Management panel in the Administration portal
- **Lock Flag:** If checked in the User Management panel, the user is unable to access the mobile application from the mobile device. This is used if a mobile device is lost or stolen and the application data must be made inaccessible.
- **Mobile Application:** Name of the mobile application chosen in the Basic Search Parameters section

- **Middleware Svr SerNo:** Server serial number of the middleware server that is running the chosen mobile application
- **Created By:** SAP user ID of the person who created the user shown
- **Creation Time Stamp:** Creation time and date of the mobile user
- **Last Changed By:** SAP user ID of the person who modified information of the mobile user shown
- **Changed Time Stamp:** Time and date when the mobile user's information was modified

Mobile User Detail - Client Object Info

The Client Object Info tab displays everything that is contained on the client device associated with a specific mobile user GUID as of the last transmit.

Mobile User Detail - Client Object Info



Object GUID	Middleware Object Ty	Middleware Object Ky	Changed Time Stamp
005056BA41F31EE38C8CC8C2D86937C4	WORKORDER	800251	10/09/2013 00:09:17

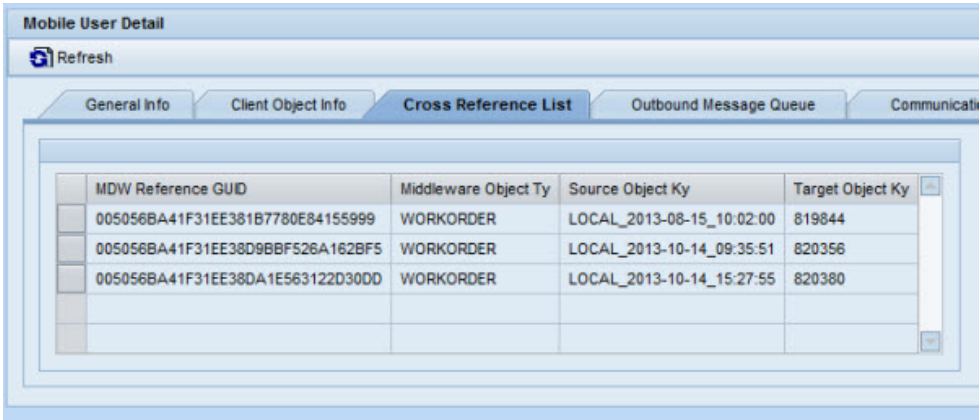
- **Object GUID:** Internal identifier of the record
- **Middleware Object Ty:** Type of object, such as notification or work order
- **Middleware Object Ky:** Object key, or ID of the object
- **Changed Time Stamp:** Time and date the object was entered into the system

Note: For details on how to disable and re-enable items, see the topic “Disabling and Re-enabling Contacts, Transactions, and FactSheets” in the “Common Changes” chapter.

Mobile User Detail - Cross Reference List

When a user creates an object on their mobile device, that object generates a local key. Upon the next transmit, SAP assigns a key to the object and maps the newly-assigned object key to the old local object key that was generated by the client device. The cross reference list displays the object GUIDs for a mobile application on a mobile device and their local object keys mapped to their SAP object keys.

Mobile User Detail - Cross Reference List



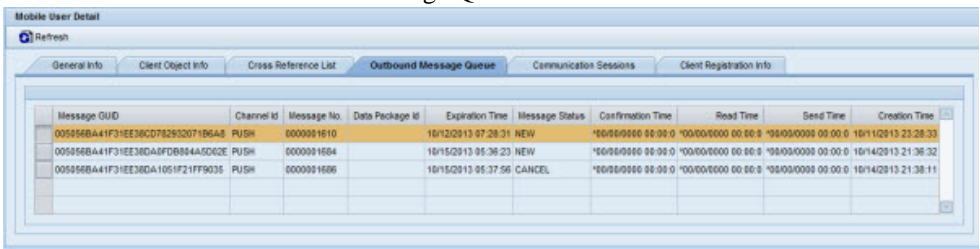
MDW Reference GUID	Middleware Object Ty	Source Object Ky	Target Object Ky
005056BA41F31EE381B7780E84155999	WORKORDER	LOCAL_2013-08-15_10:02:00	819844
005056BA41F31EE38D9BBF526A162BF5	WORKORDER	LOCAL_2013-10-14_09:35:51	820356
005056BA41F31EE38DA1E563122D30DD	WORKORDER	LOCAL_2013-10-14_15:27:55	820380

- **MDW Reference GUID:** Standard internal ID of the record
- **Middleware Object Ty:** Object type, such as work order or notification
- **Source Object Ky:** Object key originating from the client device, which is then mapped to the key assigned to the object by SAP
- **Target Object Ky:** Key that SAP assigns to the object, which is then mapped to the key originating from the client device

Mobile User Detail - Outbound Message Queue

The Outbound Message Queue tab displays all messages in the SAP outbound message queue sent through the client device associated with the user and their mobile application highlighted in the Search Results table.

Mobile User Details - Outbound Message Queue



Message GUID	Channel ID	Message No.	Data Package ID	Expiration Time	Message Status	Confirmation Time	Read Time	Send Time	Creation Time
005056BA41F31EE38CD73293207196A8	PUSH	0000001610		16/12/2013 07:28:31	NEW	'00/00/0000 00:00:00	'00/00/0000 00:00:00	'00/00/0000 00:00:00	16/11/2013 23:28:33
005056BA41F31EE38DA1FDB844A5D62E	PUSH	0000001604		16/11/2013 05:36:23	NEW	'00/00/0000 00:00:00	'00/00/0000 00:00:00	'00/00/0000 00:00:00	16/11/2013 21:36:32
005056BA41F31EE38DA1051F21FF9035	PUSH	0000001606		16/11/2013 05:37:56	CANCEL	'00/00/0000 00:00:00	'00/00/0000 00:00:00	'00/00/0000 00:00:00	16/11/2013 21:38:11

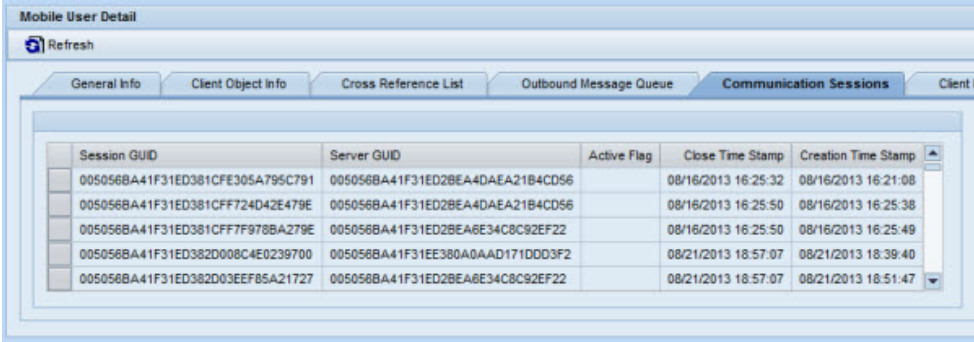
- **Message GUID:** GUID assigned to the message when it was created
- **Channel ID:** Channel for the outbound message that identifies the source of the message
- **Message No.:** Readable identifier of the message as an incremental integer
- **Data Package ID:** The data package identifier, if a data package is associated with the outbound message
- **Expiration Time:** Time set for the message to expire
- **Message Status:** Status of the message. Statuses include New, Read, Send, and Confirm.

- **Confirmation Time:** Time the message was successfully received on the device
- **Read Time:** Time the message was opened, or read, on the device
- **Send Time:** Time message is sent from the mobile device
- **Creation Time Stamp:** Time message was created, or started, on the system

Mobile User Detail - Communication Sessions

The Communication Sessions tab displays all transmissions associated with the user and the mobile application highlighted in the Search Results table.

Mobile User Detail - Communication Sessions



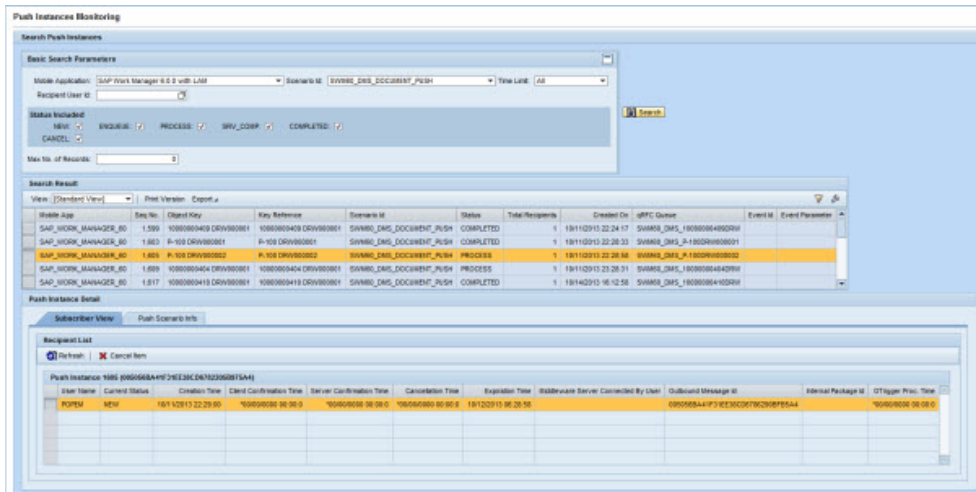
Session GUID	Server GUID	Active Flag	Close Time Stamp	Creation Time Stamp
005056BA41F31ED381CFE305A795C791	005056BA41F31ED2BEA4DAEA21B4CD56		08/16/2013 16:25:32	08/16/2013 16:21:08
005056BA41F31ED381CFF724D42E479E	005056BA41F31ED2BEA4DAEA21B4CD56		08/16/2013 16:25:50	08/16/2013 16:25:38
005056BA41F31ED381CFF7F978BA279E	005056BA41F31ED2BEA6E34C8C92EF22		08/16/2013 16:25:50	08/16/2013 16:25:49
005056BA41F31ED382D008C4E0239700	005056BA41F31EE380A0AAD171DD03F2		08/21/2013 18:57:07	08/21/2013 18:39:40
005056BA41F31ED382D03EEF85A21727	005056BA41F31ED2BEA6E34C8C92EF22		08/21/2013 18:57:07	08/21/2013 18:51:47

- **Session GUID:** Internal identifier of the session
- **Server GUID:** Identifier of the server on which the user initiated the session
- **Active Flag:** Indicator that a user stayed connected to the server throughout the session. If a client remains connected, the active flag is shown as **1**.
- **Close Time Stamp:** Time and date the session was ended
- **Creation Time Stamp:** Time and date the session was initiated

Monitoring - Push Instance Monitor

The Push Instance Monitor panel allows an administrator to search for and view details of push instances by specific mobile applications.

Monitoring - Push Instance Monitor



Basic Search Parameters

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application.
- **Scenario ID:** Type of push, selected from the drop-down menu, to search for in the pushes in the application
- **Time Limit:** Use the drop-down menu to select a window of time. The default is set to All, which is equal to selecting all push instances. All instances are available unless the historical logs have been purged in SAP.
- **Recipient User ID:** User ID receiving the push
- **Status Included:** Status of the push instance on the middleware server. Choosing no status returns all push instances on the mobile application. Multiple statuses can be chosen for the search. The following are the different status levels and their meanings:
 - **NEW** - Set when data in SAP has changed and the system configuration indicates that a push process is needed. No other information is available or set when status is in the NEW stage.
 - **ENQUEUE** - Set when the push process program is running on the data that triggered the push process. During this time, the data is locked so that it cannot be changed during the push process.
 - **PROCESS** - Standard status. The push process agent processes the instance in the push register and determines the proper recipients of the push data. The push data have been prepared for each recipient in the outbound queue in order for Agentry to pick it up.
 - **CANCEL** - Set for a push instance if there are subsequent newer push instances in the push register for the same work order. In this case, only the last push instance is processed in order to prevent multiple pushes for the same work order sent to the same mobile device.

- **COMPLETED** - Set when either no recipient is determined or all push recipients for the push have CLNT_CONF status with respect to the recipient's push message.
- **SRV_CONF** - Set for a push instance when all recipients have SRV_CONF status with respect to the individual push recipient's push message. If there are multiple push messages for the same work order and the recipients are waiting for Agentry to pick up the work orders, only the latest push event is sent to Agentry and the rest are set to SRV_CONF automatically by the push fetch BAPI. This prevents multiple copies of the same work order sent to each client.
- **Max. No. of Records:** Default is set to 2,000. Type in maximum number of records returned.
- **Search:** Click **Search** to initiate the search process. If no results are valid in the search parameters, the message No Data Found displays in the Search Results table. If valid data is returned, it displays in the Search Results table according to the Settings and Filter setup configured by the administrator.

Search Result

When you click **Search** in the Basic Search Parameters section, the Search Result table is populated. Click on the rectangle to the left of the first column in the desired row to populate the tabs in the Push Instance Detail section below.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Push Instance Monitor - Subscriber View

The Subscriber View tab on the Push Instance Monitor panel provides a recipient list containing all users and detailed information about the push instance associated with their user IDs.

Push Instance Detail - Subscriber View



Push Instance Detail

Subscriber View

Push Scenario Info

Scenario Detail

Scenario Id: EMERGENCY_WORKORDER_PUSH

Validity (Hr): 8

Mobile Application: SMART_WORK_MANAGER_S0

Source Type: EXCHOBJ

Source Object: SWM50_WORK_ORDER_PUSH

Source Handler: /SYSC/CL_PMI_AUFNR_EX_HNDLR

Subscriber Type: ALLUSER

Distribution Type: MDO

Distribution Object: SWM50_WORKORDER_PUSH

Distribution Handler: /SYSC/CL_PMI_PUSHWORKORDER_M

- **Scenario ID:** Name of the push instance scenario
- **Validity (Hr):** Hours that the push scenario remains valid after the initial push to clients
- **Mobile Application:** Name of the mobile application where the push instance resides
- **Source Type:** Type of source object associated with the push scenario
- **Source Object:** Name of the mobile data object that contains the push scenario. The source object is created or modified in the Mobile Data Object configuration panel in the Configuration portal.
- **Source Handler:** Class handler associated with the source object for the push scenario
- **Subscriber Type:** Corresponds to the Subscriber Type in the Subscriber Settings section of the Push Scenario Definition panel in the Configuration portal.
- **Distribution Type:** Corresponds to the Distribution Type in the Distribution Settings section of the Push Scenario Definition panel in the Configuration portal. MDO, or mobile data object, is the only setting available.
- **Distribution Object:** Name of the distribution object that is set in the Distribution Settings section of the Push Scenario Definition panel in the Configuration portal.
- **Distribution Handler:** Name of the distribution handler associated with the mobile data object contained in the push instance. The distribution handler is selected or changed in the Mobile Data Object Configuration panel, General Setting tab in the Configuration portal.

Monitoring - Communication Session Monitor

The Communication Session Monitor panel provides an administrator a detailed history of communications on a specific mobile application. The communications monitor can provide a history for everything in the system, unless the history has been purged from SAP.

Monitoring - Communication Session History Monitor

Mobile App/Middleware Server Name/User	User GUID	Session GUID	Creation Time	Close Time	Online Active
SAP_CRM_SERVICE_MANAGER_31					

Basic Search Parameters

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application.
- **Server ID:** Name of the middleware server
- **Time Limit:** Use the drop-down menu to select a window of time. The default is set to All, which is equal to selecting all communication sessions. All sessions are available unless the historical logs have been purged in SAP.
- **User:** SAP user ID. Manually type in the SAP user ID or click the white box icon to the right of the field for optional search methods.
- **Online Session Only:** Select from True or False:
 - True - Picks up online-only communication sessions
 - False - Picks up both online and offline communication sessions
- **User GUID:** User GUID assigned to their mobile device.
- **Max. No. of Records:** Default is set to 2,000. Type in maximum number of records returned.

Search Result

When you click **Search** in the Basic Search Parameters section, the Search Result table is populated. Click on the triangle to the left of the initial search result row to display the tree of results.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Monitoring - Object Mobile Status Monitor

The Data Object Mobile Status Monitor provides the administrator a history of SAP objects affected by a mobile user's actions. The actions are recorded in SAP and the monitor panel provides a user-friendly way to access the information.

Monitoring - Data Object Mobile Status History Monitor

Basic Search Parameters

- **Mobile Application:** Name of the mobile application. Use the drop-down menu to choose the correct mobile application.
- **Last Changed Time:** Use the drop-down menu to select a window of time. The default is set to All, which is equal to selecting all mobile object change times. All logs are available unless the they have been purged in SAP.
- **Object Type:** Type of object defined in the mobile application definition.
- **Mobile Status:** Historical view of status changes for the mobile object
- **Sort Field:** Additional information about the object, if any is provided
- **Object Key:** Internal ID of the object
- **Max. No. of Records:** Default is set to 2,000. Type in maximum number of records returned.

Search Result

When you click **Search** in the Basic Search Parameters section, the Search Result table is populated. Click on the triangle to the left of the initial search result row to display the tree of results.

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.

- **Search Result Table:** Table that displays the search results. Columns are dependent on the configuration built through the Settings link above the table. See the section *Administration Portal - Management Settings* for details on specific settings.

Administration Portal - Statistics

The Statistics panel is used to view statistics in the following three areas:

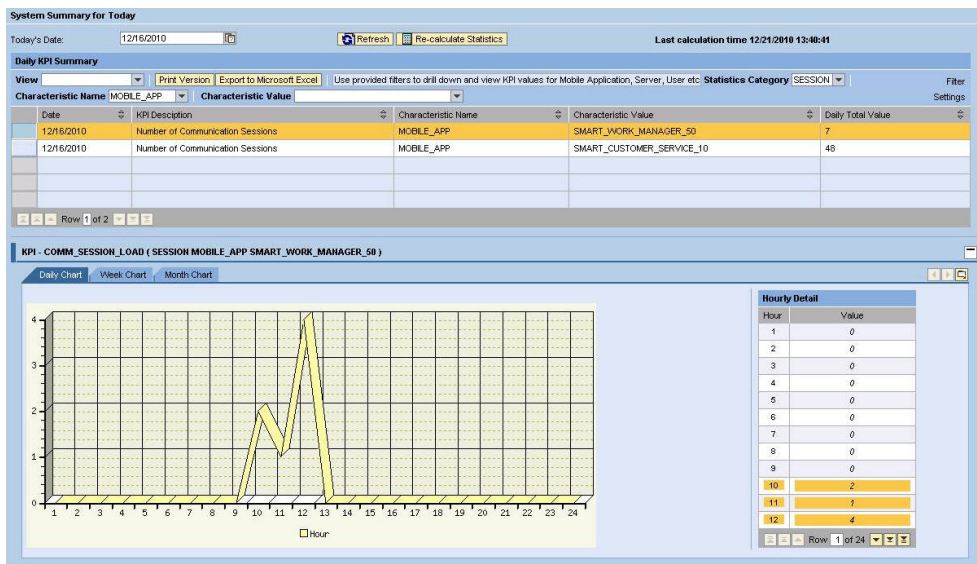
- Communication Session Statistics
- Application BAPI Wrapper Call Statistics
- Push Scenario Statistics

The statistics panels display graphical views of various key performance indicator (KPI) data. The Statistics panels are view-only. An administrator cannot create new information or change existing information through these panels.

There are three tabs for each statistic, each allowing a different graphical view: Daily, Weekly, and Monthly. The following examples depict a representation of each type of graph.

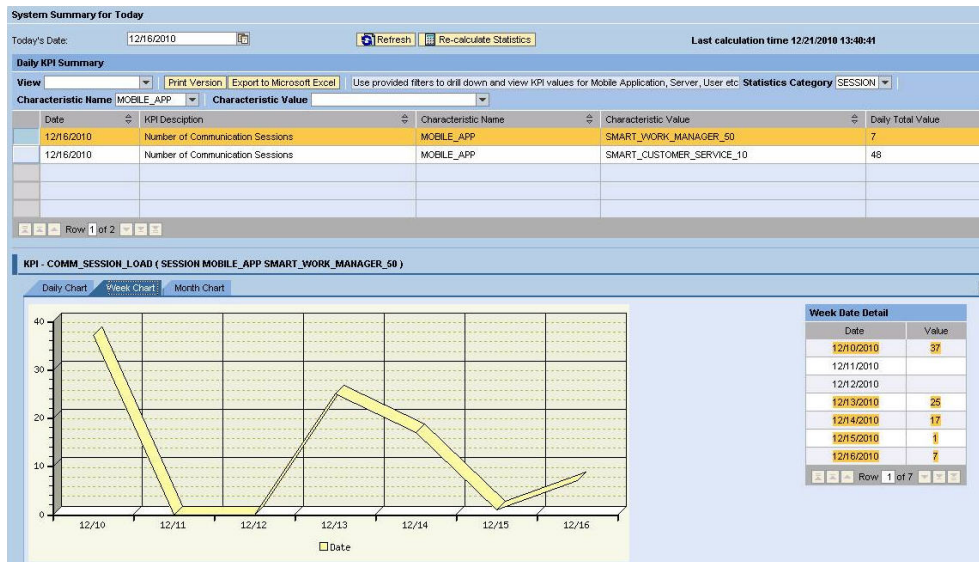
The Daily Chart tab shows a graphical representation of the chosen statistics broken down into hours.

Figure 16: Daily KPI Chart



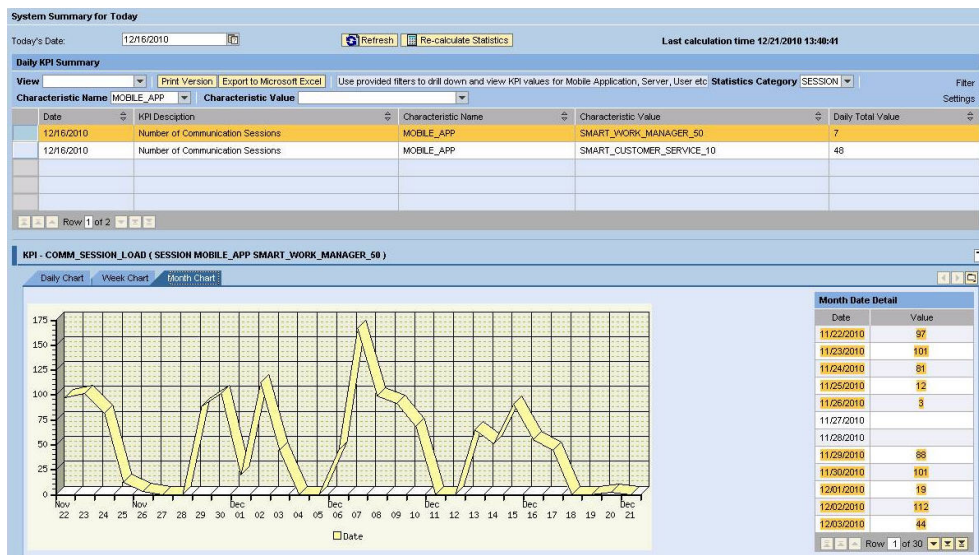
The Weekly Chart tab shows a graphical representation of the chosen statistics broken down into days, with a total of seven days.

Figure 17: Weekly KPI Chart



The Monthly Chart tab shows a graphical representation of the chosen statistics broken down into days, with a total of the amount of days in the selected month.

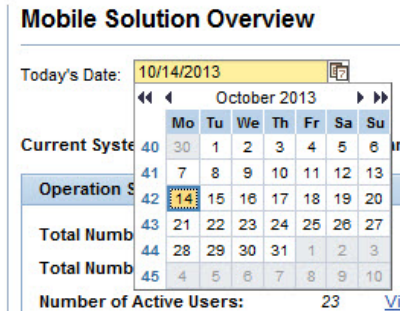
Figure 18: Monthly KPI Chart



Retrieving and Recalculating Statistics

Each panel in the Statistics section automatically displays the current date's statistics. If statistics are needed for prior dates, use the following procedure.

1. Click the calendar icon to the right of the date field and choose the desired date.



2. Click the **[Refresh]** button to refresh the statistics for the chosen date.

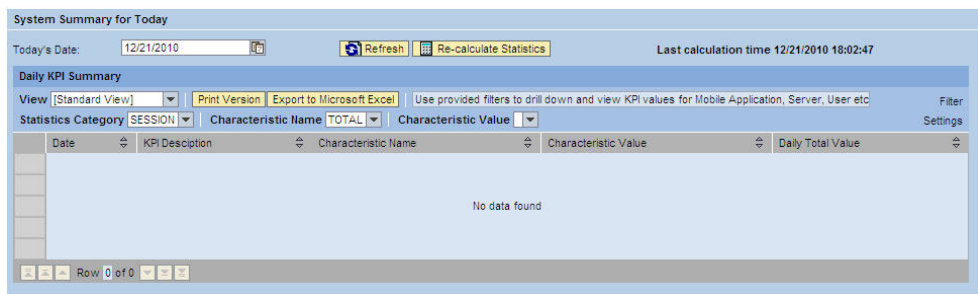
The new statistics graph displays in the KPI table.

3. If the current date is selected, to check if the statistics have changed, click **[Re-calculate Statistics]** in order to recalculate.

Statistics - Communication Session Statistics

The KPIs available through the Communication Session Statistics panel display information on the system operation status for the chosen date.

Statistics - Communication Session Statistics



Daily KPI Summary

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.

- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Statistics Category/Characteristic Name/Characteristic Value:** Use to filter statistics further. Select from the choices available through the dropdown menus. If the field is blank, no choices are available.
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Daily KPI Summary table:** Contains rows that display which statistics are available for the selected date. Highlight a row to display that graph. The graphs available are as follows:
 - **KPI - COMM_SESSION_LOAD:** The KPI - COMM_SESSION_LOAD is a graphical representation of the system load for the date chosen. The more sessions there are per hour, the heavier the load on the system, which could cause performance issues. Session load statistics are based on the number of sessions connecting; it does not take into account duration of the sessions.
 - **KPI - CONNECTING_USER_COUNT:** The KPI - CONNECTING_USER_COUNT is a graphical representation of the total users connecting to the system for the date chosen. Data gathered for this graph does not take into account a single user logging into multiple sessions; it only counts unique user IDs connecting to the system.

Statistics - Application BAPI Wrapper Call Statistics

The KPIs available through the Application BAPI Wrapper Call Stats panel display information on the number of BAPI calls for the chosen date. This statistical information can be useful in troubleshooting end user behaviors and use of the mobile application(s).

Statistics - Application BAPI Wrapper Call Statistics

System Summary for Today

Today's Date: 12/01/2010 Refresh Re-calculate Statistics Last calculation time 12/22/2010 00:14:10

Daily KPI Summary

View Print Version Export to Microsoft Excel Use provided filters to drill down and view KPI values for Mobile Application, Server, User etc Filter Settings

Statistics Category BAPI Characteristic Name TOTAL Characteristic Value

Date	KPI Description	Characteristic Name	Characteristic Value	Daily Total Value
No data found				

Row 0 of 0

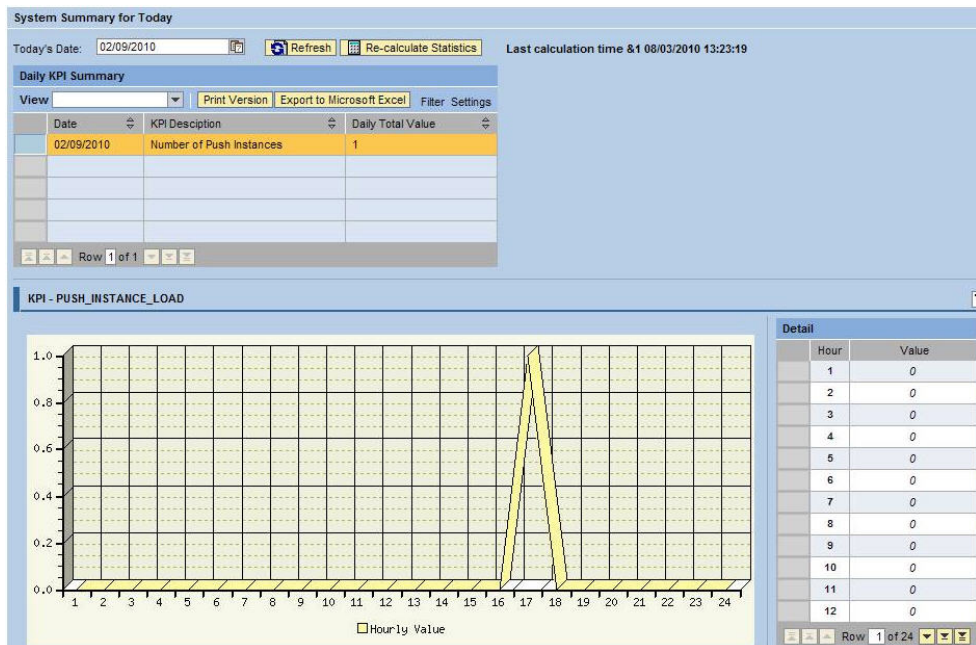
Daily KPI Summary

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Statistics Category/Characteristic Name/Characteristic Value:** Use to filter statistics further. Select from the choices available through the drop-down menus. If field is blank, no choices are available.
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.
- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Daily KPI Summary table:** Contains rows that display which statistics are available for the selected date. Highlight a row to display that graph. The graphs available are as follows:
 - **KPI - CRT_BAPI_CALL_COUNT:** The KPI - CRT_BAPI_CALL_COUNT is a graphical representation of how many BAPI calls were detected. The more calls there are per hour, the heavier the load on the backend system, which could cause performance issues. BAPI call statistics are based on the number of calls; it does not take into account what is called during the transaction process.
 - **KPI - ERROR_BAPI_CALL_COUNT:** The KPI - ERROR_BAPI_CALL_COUNT is a graphical representation of how many BAPI calls returned with an error message. These statistics can provide a good indicator to begin troubleshooting performance issues with the system, if necessary.
 - **KPI - GET_BAPI_CALL_COUNT:** The KPI - GET_BAPI_CALL_COUNT is a graphical representation of how many GET BAPI calls were made.
 - **KPI - DEL_BAPI_CALL_COUNT:** The KPI - DEL_BAPI_CALL_COUNT is a graphical representation of how many DELETE BAPI calls were made.
 - **KPI - UPD_BAPI_CALL_COUNT:** The KPI - UPD_BAPI_CALL_COUNT is a graphical representation of how many UPDATE BAPI calls were made.
 - **KPI - INITSYNC_BAPI_CALL_COUNT:** The KPI - INITSYNC_BAPI_CALL_COUNT is a graphical representation of initial synchronizations (i.e., all data is transmitted to a mobile device, not just data changed since last transmit) on the system. If a large amount of initial synchronization transmissions are present on the system for a specific time period, it could indicate issues with the system.
 - **KPI - ALL_BAPI_CALL_COUNT:** The KPI - ALL_BAPI_CALL_COUNT is a graphical representation of all BAPI calls made during the time period.

Statistics - Push Scenario Statistics

The KPIs available through the Push Scenario Statistics panel display the amount of pushes that occurred for the chosen date on an hourly basis. This statistical information can be useful in determining peak loads and to assess factors that drive push scenarios.

Statistics - Push Scenario Statistics



Daily KPI Summary

- **View:** If the administrator sets up different views using the Settings link, the drop-down menu will display those view names. Select a different view for specific data needs.
- **Print Version:** If enabled, creates a PDF version of the data in the Search Results table.
- **Export to Microsoft Excel:** Exports all data in the Search Results table to an Excel spreadsheet
- **Statistics Category/Characteristic Name/Characteristic Value:** Use to filter statistics further. Select from the choices available through the drop-down menus. If field is blank, no choices are available.
- **Filter/Delete Filter:** If the Filter tab is utilized in the Settings window accessed by the Settings link, click the Filter link to display filter choices in order to further filter the data displayed. If a filter is in use, click Delete Filter to remove the filter and display all data returned by the initial search performed.

- **Settings:** Click this link to display a Settings panel in order to modify how search results are displayed. See the section on changing Administrator settings for more details.
- **Daily KPI Summary table:** Contains rows that display which statistics are available for the selected date. Highlight a row to display that graph. The graphs available are as follows:
 - **KPI - PUSH_INSTANCE_LOAD:** The KPI - PUSH_INSTANCE_LOAD is a graphical representation of total number of pushes on the system.
 - **KPI - PUSH_INST_CANCEL_COUNT:** The KPI - PUSH_INSTANCE_CANCEL_COUNT is a graphical representation of total number of push instances with a CANCEL status.
 - **KPI - PUSH_AVG_TIME_CMPLTE:** The KPI - PUSH_AVG_TIME_CMPLTE is a graphical representation of the average time between when a push instance is created and when it reaches a COMPLETE status.
 - **KPI - PUSH_AVG_TIME_PROCESS:** The KPI - PUSH_AVG_TIME_PROCESS is a graphical representation of the average time between when a push instance is created and when it is processed by the push processor.
 - **KPI - PUSH_AVG_TIME_SVR_CONF:** The KPI - PUSH_AVG_TIME_SVR_CONF is a graphical representation of the average time between when a push instance is created and when it reaches SVR_CONF status.
 - **KPI - PUSH_AVG_TIME_CLNT_CONF:** The KPI - PUSH_AVG_TIME_CLNT_CONF is a graphical representation of the average time between when a push instance is created and when it reaches CLNT_CONF status.
 - **KPI - PUSH_CLNTCNF_TIME_TOPRANK:** The KPI - PUSH_CLNTCNF_TIME_TOPRANK is a graphical representation of the shortest time between when a push instance is created and when it reaches CLNT_CONF status.
 - **KPI - PUSH_CLNTCNF_TIME_LOWRANK:** The KPI - PUSH_CLNTCNF_TIME_LOWRANK is a graphical representation of the longest time between when a push instance is created and when it reaches CLNT_CONF status.

Copying an Object to the Customer Namespace

The following procedure provides information on making a copy of a mobile data object (MDO) or exchange object within the Agentry SAP Framework. For configuration changes where an MDO or exchange object is being modified, it is recommended that a copy is first made and placed in the customer namespace. In any of the procedures provided where an MDO or an exchange object should be copied, refer to this procedure for instructions. Copying these elements before making a modification ensures changes made to the application can be easily rolled back without affecting the originals.

Once a copy is made of an MDO and that copy is then modified for a configuration change, it will likely be necessary to adjust the BAPI wrapper assignment to reference the new MDO. Similarly, when an exchange object is copied and then modified, the EFI trigger assignment may need to be changed to the new exchange object. These procedures are covered separately.

1. Log into the Configuration Panel of the Agency SAP Framework.
2. Click either **Exchange Object Configuration** or **Mobile Data Object Configuration** from the navigation menu.

The Data Object Detail panel opens.

Note: Figures shown in this procedure are taken from the Mobile Data Object configuration page. Screens may look different when configuring an exchange object. For either, the ability to copy is provided.

Data Object Detail (Display Mode)

Create Copy Delete

General Setting ResultSet Field Selection Data Filter Data Staging Proxy Setting Composite Settings

Basic Data

Mobile Data Object Id:

Description:

Data Object Type: Mobile Application:

Reference Business Object:

Data Object Handler Settings

Data Object Handler: Skip Exception Processing: ☐

Get Method:

Create Method:

Update Method:

Delete Method:

Exchange Object Settings

Exchange Object:

Conversion Exit Setting

Enable Conv. Exit Overwrite: ☐

Middleware Reference Info

Reference Middleware Object Type:

Activation

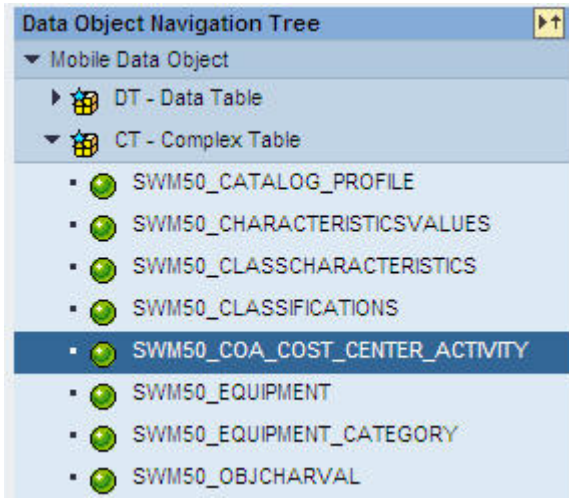
Data Object Active: ☐

Administrative Info

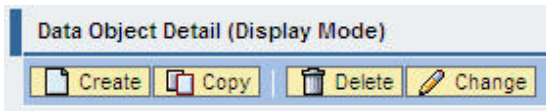
Created By: Creation Time Stamp: *00/00/0000 00:00:0

Last Changed By: Changed Time Stamp: *00/00/0000 00:00:0

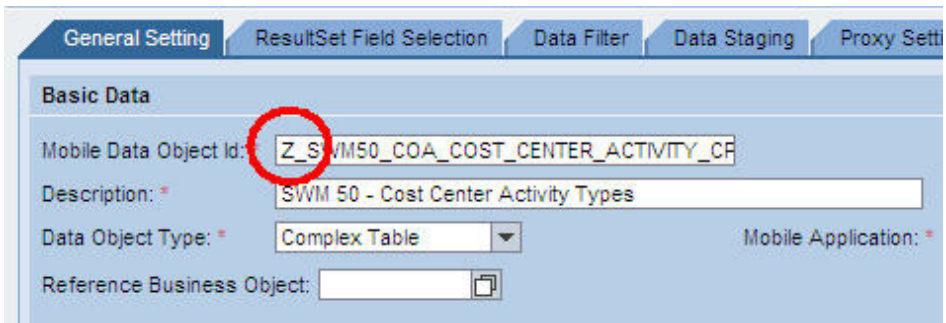
3. In the list of MDOs or exchange objects, select the one to be copied.



4. Click **Copy**.



5. In the Basic Data, or main panel object ID field, add a 'Z' to the beginning of the object name.



6. When finished, click **Save** to save the object copy.

A copy of the original object is created in the SAP customer namespace. The element can now be modified, with the back up element available for rollback purposes, if necessary.

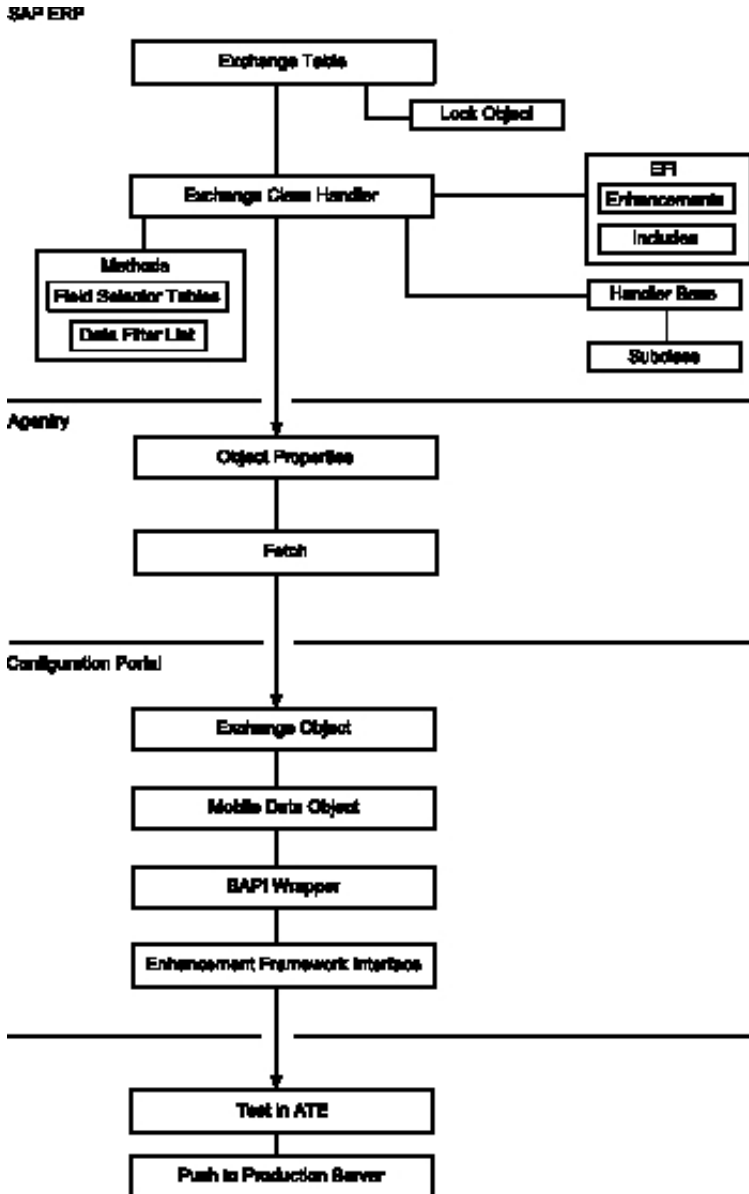
Adding a New Downstream Synchronization Process

The exchange process detects changes from transaction updates and records the change activity into an exchange table `/SYCLO/<fieldname>_EX`. The exchange table information is used by the mobile data object class handler to accomplish the delta exchange process so that the mobile application only sends and receives delta data using a time stamp.

The source code plug-in is implemented using the enhancement framework. To implement this, either the multi-instance BADI or the enhancement spot must include source code to update the exchange table for the current change. In order to implement the enhancement, the location of the enhancement must be identified. The location of the enhancement should be such that the old/new values of the changed records are available. These old/new values will be used by the exchange process to identify changes based on field selections and filter conditions configured for the exchange object.

The source code plug-in is either an include program `/SYCLO/<module>_EFI_<key field name>_EX_INCL` or an update module `/SYCLO/<module>_<exchange name>_SAVE_EX`. If the enhancement is being implemented in an UPDATE module, the plug-in is an INCLUDE program. Otherwise, it is an UPDATE module (the INCLUDE program is called within this module) to ensure the exchange table is updated in an UPDATE TASK.

The INCLUDE program is then assigned to an exchange object in the Agentry SAP Framework Configuration portal to activate the actual change detection process.



Implementing Downstream Synchronization

1. Implement the exchange process in SAP:

If there are similar objects already existing in SAP for any of these steps, it may be easier to select that object, right click, and choose the Copy menu item. After the object is copied, right click and select the Change menu item to make the necessary changes.

- a) Add or modify the appropriate exchange table to store the change activity record.
- b) Add or modify the appropriate lock object to trigger the lock mechanism for the update process.
- c) Add or modify the exchange class handler.
- d) Add or modify the methods by redefining the method `GET_FIELD_SELECTOR_TABLES` to enable the field selector function for the exchange object.
- e) List the database tables supported by the exchange handler for change detection field selection and pass it to `ET_FLDSEL_TABLE_LIST`.
- f) If required, redefine the method `GET_TABLE_FILTER_LIST` to enable the change detection filter function for the class handler. This provides the ability to restrict change detection based on data content.
- g) List the filters supported by this exchange class handler along with the table name and the field name and pass it to the output table `ET_DATA_FILTERS`. Take the data object filter name from the technical properties of the field.
- h) Add or modify the EFI enhancements.
- i) Add or modify the EFI includes and insert in the appropriate enhancement location.

The following areas may need to be added or modified when developing the source code plug-in:

- Determine the current transaction update mechanism.
 - Construct an object key for the exchange table.
 - Build the data table with old and new values to detect changes based on field selections and filter conditions configured for the exchange object.
 - Call the exchange object class handler method `UPDATE_EXCHANGE` to update the exchange table with the current action and timestamp.
- j) Add or modify the handler base by inheriting all the properties from the superclass / `SYCLO/CL_CORE_EX_HANDLER_BASE`. This superclass contains the base methods to fulfill the business logic related to the exchange process.
 - k) Add or modify the subclass.

2. Implement the downstream synchronization in Agentry Editor:

- a) Create new steplets for fetches and transactions.
- b) Create the associated stephandlers.
- c) Create the new POJO for the exchange object.

3. Add the new exchange object in the Agentry SAP Framework Configuration portal:

- a) Add or modify the exchange object.
- b) Add or modify the mobile data object.
- c) Add or modify the BAPI wrapper and assign the object to the BAPI wrapper.
- d) Add or modify the EFI.

Working with BAPI Wrappers

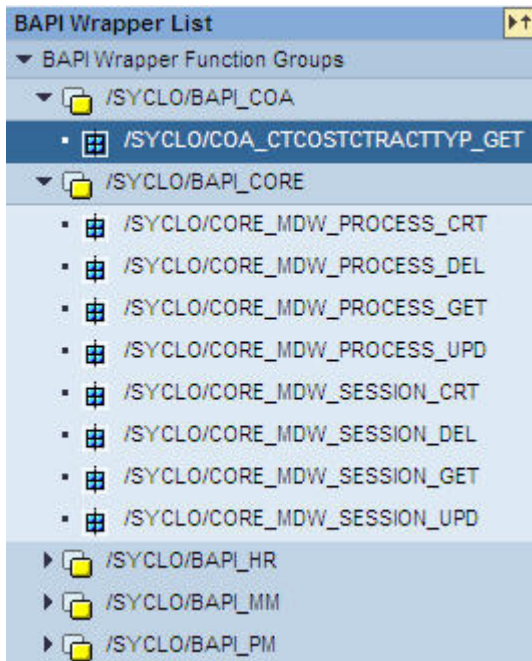
BAPI wrappers are an element type within the Agency SAP Framework. The BAPI wrappers are the elements through which calls are made by the SAP Agency Server to the elements configured in the MDOs, (i.e., the GET, CREATE, UPDATE, and DELETE methods). A BAPI wrapper is assigned to an MDO in the Agency SAP Framework. The BAPI wrapper defines the inputs and outputs for the calls made to it and the data it returns to the SAP Agency Server.

When making modifications or configuration changes to MDOs, and when a copy of the MDO is made in the customer namespace as a part of the modification, the assignment settings of a BAPI wrapper must be changed to reference the new copy.

Changing the MDO Assignment of a BAPI Wrapper

Once an MDO is created, it must be assigned to a BAPI wrapper. During runtime, the MDO invoked is determined based on the BAPI wrapper assignment. This procedure describes the steps needed to change a BAPI wrapper's MDO assignment.

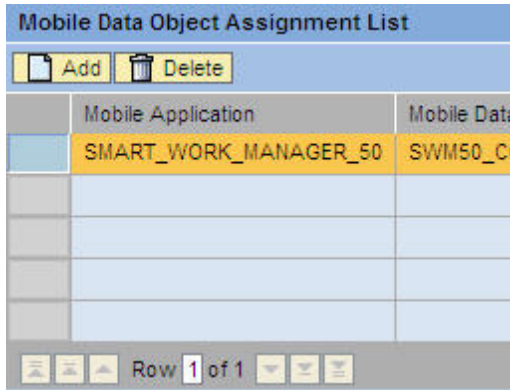
1. Open the Syclo Agency SAP Framework Configuration portal through SAP.
2. Click the **BAPI Wrapper Configuration** menu item.
3. Navigate to and highlight the BAPI wrapper that will use the new or modified MDO or Z-MDO in the BAPI Wrapper List tree.



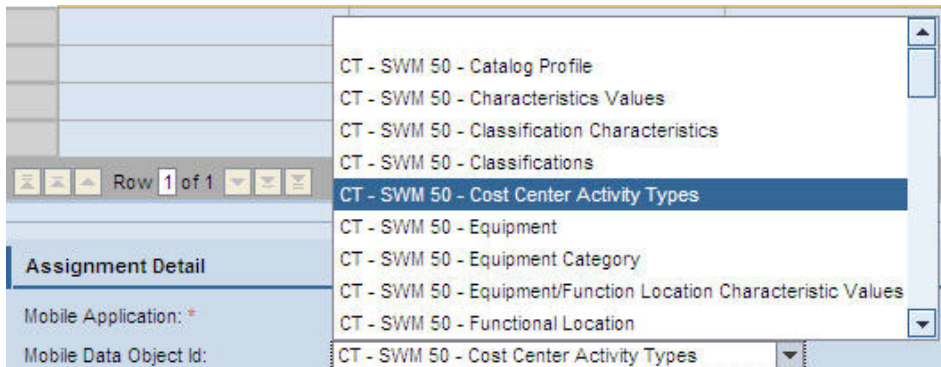
4. Click the **Assignment** tab.
5. Click **Change**.



6. Use the **Add** button in the Mobile Data Object Assignment List panel to add the new Z-MDO.



7. Click the arrow to the right of the Mobile Data Object ID field and select the desired MDO from the drop-down list.



8. Change the Method Type field to select the MDO method to be called by the BAPI wrapper.
9. Highlight the original MDO in the Mobile Data Object Assignment List table and click **Delete**.

The original is no longer assigned to the BAPI wrapper, leaving the newly-added Z-MDO.

10. Click **Save** to save the changes.

Changing MDO Filter Rules

Many of the common configuration changes made for an implementation involve the modification or addition of one or more filter rules within an MDO. In SAP, each user is assigned a role-based profile with authorization permissions on viewable data and available activities. For example, a user working in one plant should not be able to view data for another plant. When business activities performed by a user are mobilized through the mobile application, the ability to extend the same restrictions to the mobile application is necessary. Data filter rules provide the function to restrict data access for mobile applications.

This procedure describes the steps necessary, in general, to modify a data filter rule for an MDO. The specific settings of a given rule will vary depending on the overall nature of the change being made. Subsequent procedures will reference this process and provide the detailed values and settings for the filter rules involved in the specific change.

1. Open the Agentry SAP Framework Configuration portal through SAP.
2. From the ConfigPanel Home page, click the **Mobile Data Object Configuration** menu item.

The Mobile Data Object Configuration page displays.

3. Expand the Mobile Data Object tree in the Data Object Navigation Tree section and highlight the MDO created earlier in this procedure.
4. Expand the desired method in the Defined Filters list.
5. Under the method are listed all current filters defined for the method. Select the filter whose rules should be modified.

The current rule filter settings are displayed in the Rule Editor section. All existing rules for the filter are displayed in the Rule List table.

6. To add a new rule, edit an existing one, or delete a rule from the filter, click **Change** at the top of the page. Many of the fields in the Rule Editor section become editable, and two new buttons are displayed to the right of the Rule List field, **Create** and **Delete**.
7. To delete a rule from the filter, select that rule in the list and click **Delete**. If no further changes are to be made, click **Save** at the top of the page to completely remove the rule from the filter. If additional changes are still needed, the filter can be saved after all changes are complete.
8. To add a new rule to the filter, click **Create** to the right of the Rule List. To edit an existing rule within the filter, select it in the Rule List. This will display the current settings for the rule in the DOF Rule Type, Sign, Option, Low Value, High Value, and Active Flag fields within the Rule Editor section.
9. Set or modify the editable fields according to the needs of the application. For a detailed description of all fields, see *Mobile Data Object - Data Filter*.

10. Be sure the Active Flag field is set to true for each added or edited field before saving changes. Inactive filter rules will have no effect on the synchronization processing. An Active Flag is set by clicking in the box in order to display a checkmark.
11. When all desired changes have been made to the filter rules click **Save** to apply those changes.

Creating a New Filter Rule

1. Click the BAPI Wrapper Settings in the Navigation Panel.
2. Navigate to the BAPI Wrapper that uses the MDO you are modifying, in the BAPI Wrapper List.
3. Click the **Assignment** Tab and click the **Change** Button.
4. Use the Add button to add ZMDO version of the complex table, with the newly created field. Use the delete button to delete the original complex table.
5. Click the **Save** button to save the changes.
6. From the SAP user menu, click **SMART Mobile Suite Configuration**.

This will launch a web browser, which will log into the SMART Mobile Suite Configuration Site.

7. Click the **Mobile Data Object Settings**.

This will display the Mobile Data Object Settings Drop down menu:

8. Expand this Menu and select desired Data object for the data filter.
9. Select the Data Filter Tab.
10. Click the Create button near the top of the screen.
11. Use the filter settings to define the new filter.
12. Click the **Save** button to save your changes.
13. Use the ATE to verify the filters perform as expected on the client. Validate the expected material is fetched to the device, by paying attention to the transmit log as it completes.

Adding a New Data Table

1. **Create a new data table in the Configuration portal.**
 - a) Create a new subclass in the Z namespace from `/syclo/cl_dt_DO`.
 - b) Override `get_data_table` method to retrieve data from SAP.
 - c) Override `get_data_filter_list` method to support any filter logic.
 - d) Create a new mobile data object in the Z namespace, referred to as a Z-MDO.
2. **Add the new data table definition to Agentry.**
 - a) Open the SAP project in Agentry.

- b) Create the new data table.
- c) Make any other applicable changes to the application that is using the new data table, such as displaying the new value or modifying rules to search on the new values.

3. Configure Java synchronization between the data table and Agentry.

- a) Open the appropriate Java project for the SAP application.
- b) Modify the new data table class created when the data table definitions was added to the Agentry application project.
- c) Create the associated stephandlers.
- d) Create the new POJO for the data table.
- e) Publish the application to the SAP Agentry Development Server in preparation for testing. Restart the Server as changes were also made to the Java logic in support of the new data table.

Next

Perform a transmit from the Agentry Test Environment and verify that the new data table and all related functionality are producing desired results.

Adding a New Complex Table

Adding a new complex table to the application requires four main tasks:

- Creating or modifying the exchange tables and associated objects in SAP.
- Using the Agentry SAP Framework Configuration portal to create a new MDO or Z-MDO for the complex table.
- Creating new complex table Java class, step handler and POJO for the new complex table in the Java editor in Eclipse.
- Using Agentry Editor to modify the application to make use of and support the new complex table.

1. Create or modify the exchange process in SAP

- a) Determine the enhancement spot in SAP if an exchange process is required. An enhancement spot is a location in the SAP code where access is available to the data that is changing either through a transaction or a BAPI.
- b) Create the enhancement implementation. The enhancement implementation should be in an update module.
- c) Create the exchange table.
- d) Create the lock object associated with the exchange table.
- e) Create the EFI implementation. The EFI implementation reads the data being changed in SAP and calls the method `/syclo/cl_cor_exch_serve=>update_exchange` to update the exchange table.
- f) Determine and create the structure of the data that is sent back in the `ET_COMPLEX_TABLE` parameter of the BAPI wrapper.

- g) Determine and create the structure of the deleted records that are sent back in the ET_EXCHANGE_ACTION_DELETED parameter of the BAPI wrapper.
- h) Create the MDO handler class, which will inherit from the class /SYCLO/CL_CORE_CT_HANDLER.

2. Create a new complex table in the Configuration portal

- a) Create a new complex table MDO or copy an existing MDO to the Z-namespace.
- b) If needed, create a new BAPI wrapper in the Z-namespace using the BAPI include template /syclo/core_bapi_template_incl.
- c) Assign the new Z-MDO to the new BAPI wrapper, or assign the new Z-MDO to an existing BAPI wrapper, depending on your configuration.

3. Create the complex table in the Agentry Editor

- a) Create a new complex table in Agentry Editor to make use of and support the data in SAP.
- b) Create the field definitions within the complex table to
- c) Add any necessary indexes to the complex table.

4. Configure Java synchronization between the complex table and Agentry

- a) Open the appropriate Java project for the SAP application.
- b) Modify the new complex table class created when the complex table definitions was added to the Agentry application project.
- c) Create the associated step handlers.
- d) Create the new POJO for the complex table.
- e) Publish the application to the SAP Agentry Development Server in preparation for testing. Restart the Server as changes were also made to the Java logic in support of the new complex table.

Next

Perform a transmit from the Agentry Test Environment and verify that the new complex table and all related functionality are producing desired results.

Adding a New Data Object

Adding a new data object to the application requires four main tasks:

- Redefine methods in SAP to allow for the new data object.
- Using the Agentry SAP Framework Configuration portal to create a new MDO or Z-MDO.
- Creating new data object Java class, stephandler and POJO for the new data object in the the Java editor in Eclipse.
- Using the Agentry Editor to modify the application to make use of and support the new data object.

1. Redefine methods in SAP

- a) Redefine the **get_field_selector_tables** method for the class handler to enable field selector function for the mobile data object.
 - b) In the **get_field_selector_tables** method, list the database tables supported by the class handler GET method for the result set field selection **ET_FLDSEL_TABLE_LIST**.
 - c) List out the mandatory key fields from the required tables and pass them to the output table **ET_REQ_FIELD_LIST** to ensure that all key fields are included in the output selection.
 - d) Redefine the **get_data_filter_list** method to enable data filter function for the class handler and define the necessary filter rules to control what data can be viewed by the mobile application.
 - e) List the filters supported by the **get_data_filter_list** method along with the table and field names and pass them to the output table **ET_DATA_FILTERS**. Take the data object filter (DOF) name from the technical properties of the field.
2. **Create a new data object in the Configuration portal**
 - a) Create a new mobile data object in the Z namespace, referred to as a Z-MDO.
 - b) Create a new subclass in the Z namespace from **/sycolo/cl_DO_Handler_base**.
 - c) Override the GET, CREATE, UPDATE, and DELETE methods as needed to retrieve data from SAP.
 - d) Create a new BAPI Wrapper in the Z namespace using the BAPI include template: **/sycolo/core_bapi_template_incl**. Or, assign the Z-MDO to an existing BAPI wrapper.
 - e) Assign the new Z-MDO to the new BAPI wrapper if it was not assigned to an existing BAPI wrapper.
 3. **Configure Java synchronization between the data object and Agentry**
 - a) Create new steplets for fetchs and transactions.
 - b) Create the associated stephandlers.
 - c) Create the new POJO for the data object.
 4. **Add the new data object definitions to Agentry**
 - a) Open the SAP project in Agentry.
 - b) Create the new data object to represent the business object from SAP.
 - c) Create the transactions and read steps of the data object.
 - d) Create any necessary fetches or transactions associated with the data object.
 - e) Publish the application to the SAP Agentry Development Server in preparation for testing. Restart the Server as changes were also made to the Java logic in support of the new data object.

Next

1. Perform a transmit from the Agentry Test Environment and verify that the new data object and all related functionality are producing desired results.
2. Publish the application, including the supporting Java synchronization logic, to the SAP Agentry Production Server for deployment.

Adding new Values to be Retrieved for Mobile Application Definitions

Prerequisites

The following items must be addressed prior to performing this procedure:

- The desired values to be added to those being retrieved should be determined and noted, as should the tables within which they reside within SAP.
- The person performing this procedure must have access to the Agentry SAP Framework Configuration Panel and have permissions to change the configuration settings of the elements within it.

Task

Use this procedure to add new fields to complex tables or objects. The following procedure uses a complex table as an example. However, these steps will accomplish the same goal to add new values to object definitions as well. Where there are differences, they are noted in this procedure.

1. Within the Agentry SAP Framework Configuration Panel ConfigPanel Home page, select the menu link **Mobile Data Object Configuration**. Select the proper MDO from the list on the left of the configuration page, which displays its current settings.
2. Copy this MDO to the Z namespace. All changes to the MDO should be made to this new copy in the Z namespace. The original MDO should not be modified.

For information on copying an MDO, see the procedure section “Copying an MDO or Exchange Object to the Customer Namespace.”

3. **Select the additional fields in the MDO:**
 - a) Navigate to the Mobile Data Object Configuration panel in the Configuration Panel and double-click the appropriate MDO.
 - b) Click the **Result Set Field Selection** tab and click the **Change** button.
 - c) In the Field Selection Detail pane, expand the table to which active fields will be added.

The Field Active column displays in white, with check mark boxes enabled.

Field Selection Detail			
		Search	<input type="text"/>
Field Catalog	Field Active	Field Description	Data Format
▼ Handler Method - GET_EMPLOYEE_LIST	<input checked="" type="checkbox"/>		
▼ Table - PA0001	<input checked="" type="checkbox"/>	HR Master Record: Infotype 0001 (Org. Assignment)	
▪ Field - ABKRS	<input checked="" type="checkbox"/>	Payroll area	CHAR 2
▪ Field - AEDTM	<input checked="" type="checkbox"/>	Changed on	DATS 8
▪ Field - ANSVH	<input checked="" type="checkbox"/>	Work contract	CHAR 2
▪ Field - BEGDA	<input checked="" type="checkbox"/>	Start Date	DATS 8

- d) Scroll through the fields to determine which ones to enable or disable. Checking the box enables the field, while un-checking the box disables the field.
 - e) Click **Save** to save your changes.
4. Navigate to the ConfigPanel Home page and select the **BAPI Configuration** link. In this page, change the BAPI wrapper assignment of the proper BAPI wrapper to use the MDO in the Z namespace just modified in the previous step.

For information and instructions on changing a BAPI wrapper's MDO assignment, see the procedure section "Changing the MDO Assignment of a BAPI Wrapper."

5. Complete Java synchronization through Eclipse:

- a) Open the appropriate Java project in Eclipse for the mobile application.
- b) Locate the appropriate BAPI wrapper Java file in the project. You can find this by locating the BAPI wrapper name that is associated with the MDO in the Configuration portal. Copy this BAPI wrapper name and perform a file search within Eclipse to locate the correct Java file.
- c) Open the POJO declaration within the Java file.
- d) Locate the public strings and the properties. Add new data members to the class matching the fields selected for retrieval in the MDO. Also define the new setter methods for those selected fields.

```
public class Plant extends ComplexTableObject {

    public String ID;
    public String Name;
    public String ValuationArea;
    public String CompanyCode;

    public Plant() {
        super();
    }

    public void setProperties(JCO.Table tbl) throws Exception {
        setID(tbl.getString("WERKS"));
        setName(tbl.getString("NAME1"));
        setValuationArea(tbl.getString("BWKEY"));
        setCompanyCode(tbl.getString("BUKRS"));
    }
}
```

- e) Save and compile the Java code by running the build-mm.xml script. This should initially be performed on the SAP Agentry Development Server to allow for testing before publishing to the Production server.

6. Add the new fields to the complex table or object definition in Agentry:

- a) Open the SAP project in Agentry.
- b) Navigate to the complex table or object that will contain the newly-added fields.
- c) For complex tables, select the **Fields** tab and click the green plus icon to add a new field.
- d) Add the new field using the wizard, naming it to match the name given to the values in the Java class. When done, click the **Finish** button.
- e) Click on the **Indexes** tab to add new indexes if necessary.
- f) For objects, select the **Properties** tab and click the green plus icon to add a new field.
- g) Add the new property using the wizard, naming it to match the name given to the values in the Java class. When done, click the **Finish** button.
- h) Make any other applicable changes to the application that is using the complex table or object, such as displaying the new value or modifying rules to search on the new values.
- i) Publish the application to the SAP Agentry Development Server in preparation for testing. Restart the SAP Agentry Development Server as changes were also made to the Java logic in support of the new fields.

With the completion of this procedure, one or more new values will be retrieved as a part of the data for the object or complex table definition. These new values may be displayed, edited, searched on, or used in any other appropriate manner on the client.

Next

When these modifications are complete, the application should be thoroughly tested. It should be verified that the values added are retrieved as expected. Any functionality related to these

values should also be tested. Once testing is successful, the modifications made should be migrated to the production system according to migration processes in place at the implementation site.

Adding New Fields to an Exchange Object

Use this procedure to add new fields exchange objects.

The following are accomplished in this procedure:

- A new field is enabled in the exchange object through the Configuration portal
 - Java synchronization is achieved in Eclipse
 - The new object is added in the Agentry editor
1. If a new exchange object is needed, copy an existing exchange object to the Z namespace in SAP.

For more information, see the section Copying an Object to the Z Namespace.

2. **Add new fields to the exchange object:**

- a) Navigate to the Exchange Object Configuration panel in the Configuration portal and double-click the appropriate exchange object.
- b) Click the **Change Detection Field Selection** tab and click the **Change** the button.
- c) In the Exchange Object Field Selector Detail pane, expand the table to which active fields will be added.

The Field Active column displays in white, with checkmark boxes enabled.

Exchange Object Field Selector			
		Search	<input type="text"/>
Field Catalog	Active Flag	Short Description	
▼ Table - IFLO*	<input type="checkbox"/>	Generated Table for View IFLO	
▪ Field - IWERK	<input checked="" type="checkbox"/>	Planning plant	
▪ Field - LVORM	<input checked="" type="checkbox"/>		
▪ Field - PLTXT	<input checked="" type="checkbox"/>	FunctLocDescrip.	
▪ Field - RBNR	<input checked="" type="checkbox"/>	Catalog profile	
▪ Field - SWERK	<input checked="" type="checkbox"/>	Maintenance plant	
▪ Field - TPLMA	<input checked="" type="checkbox"/>	Superior FunctLoc.	
▪ Field - TPLNR	<input checked="" type="checkbox"/>	Functional Location	
▪ Field - ABCKZ	<input type="checkbox"/>	ABC indicator	

- d) Scroll through the fields to determine which ones to enable or disable. Checking the box enables the field, which unchecking the box disables the field.
- e) Save your changes by clicking the **Save** button.

3. **Complete Java synchronization through Eclipse:**

- a) Open the appropriate project in Java, or create a new project.
- b) Locate and open the appropriate Java file in the project.
- c) Locate the public strings and the properties. Add the new table fields checked in the Exchange Object Field Selector Detail pane in the Configuration portal.

```
public void setProperties(Table tbl) throws Exception {
    DBTPLNR = tbl.getString("TPLNR");
    functionalLocationDB = tbl.getString("TPLNR");
    if (tbl.getName().equalsIgnoreCase(JCO_UPDATES_TABLENAME))
        // following fields only returned in updates table
    {
        functionalLocation = tbl.getString("TPLNR_&EXT");
        superiorFunctionalLocation = tbl.getString("TPLMA_&EXT");
        description = tbl.getString("PLTXT");
        site = tbl.getString("IWERK");
        catalogProfile = tbl.getString("RBNR");
        planningPlant = tbl.getString("IWERK");
        status = tbl.getString("SYSTEM_STATUS");
    }
}
```

- d) Save and compile the Java code by running the build-mm.xml script. This should initially be performed on the Development server to allow for testing before publishing to the Production server.

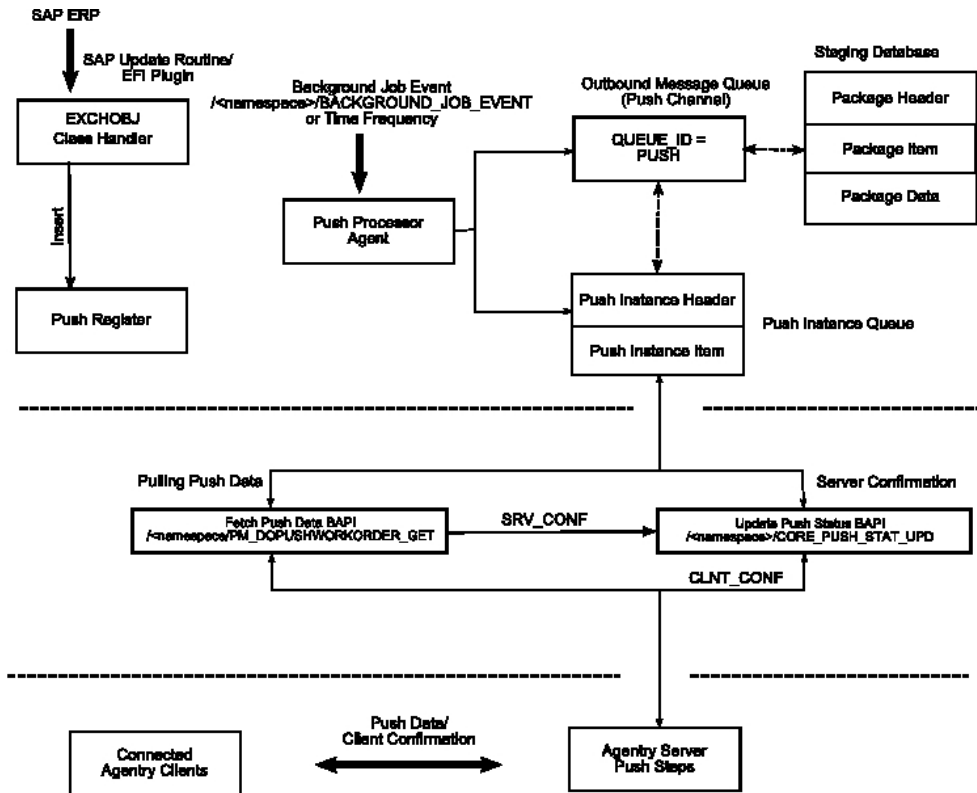
4. Add the new fields to the exchange object in Agentry:

- a) Open the SAP project in Agentry.
- b) Navigate to the appropriate object.
- c) Add any new required properties, transactions, or read steps associated with the new fields.
- d) Click the **Finish** button when done.
- e) Make any other applicable changes to the application that is using the modified object.
- f) Publish the application to the SAP Agentry Development Server in preparation for testing. Restart the Server as changes were also made to the Java logic in support of the new fields.

Working with Push Scenarios

A push scenario pushes emergency work orders to the corresponding recipients. Use the following diagram and steps to follow a push instance from generation in SAP to reception on the Client.

Push Process Flow



1. The push exchange process initiates the push trigger based on the push conditions. Conditions are defined as filter rules in the push exchange object. For instance, `work order priority = 1` is considered an emergency work order in the base product release.
2. The work order that satisfies the push conditions inserts a record into the push register table /SYCLO/PSH01 with an object key as the work order number and a push status of NEW.
3. The event /SYCLO/BACKGROUND_JOB_EVENT is raised after the work order is saved, which triggers the background job for the push processor agent.
4. The push processor job /SYCLO/CORE_PUSH_PROC_PROG is triggered, either by using the event or the time frequency. This trigger is based on specific customer processes.
5. The push processor determines the recipients for the push work order and builds the data for each recipient as a separate instance. The instance is stored in the outbound message queue /SYCLO/PSH02 with queue ID = PUSH, using the staging database.
6. The push instance displays one of the following statuses, viewable in the push monitor in the Administration Panel:
 - NEW
 - PROCESS

- CANCEL
 - COMPLETED
 - SRV_COMP
7. The Agentry application within the SAP Mobile Server calls the push BAPI /SYCLO/PM_DOPUSHWORKORDER_GET for every predefined time interval and checks the push queue for new items.
 8. The Agentry application within the SAP Mobile Server sends the push data to the respective Clients depending on the user credentials that match the push instance.
 9. Once the Client receives the push message, it sends the Client confirmation back to the Server and the Server calls the BAPI /SYCLO/CORE_PUSH_STAT_UPD to update the confirmation with status CLNT_CONF back to SAP.

Adding a New Push Scenario

In order to create a new push scenario, the following need to be defined:

- Source attributes
- Distribution settings
- Subscriber list

1. Generate the MDO class handler for the work order push:

- a) Create a new class interface by inheriting all the properties from the base class / SYCLO/CL_CORE_DO_HANDLER_BASE.
- b) Activate the push by redefining the method CHECK_PUSH_SUPPORTED to determine whether the push service is supported by this handler.
- c) Redefine the method DETERMINE_PUSH_RECIPIENTS to identify valid users for the emergency work order push.
- d) Based on the work order assignment type, determine valid partner information and obtain the recipients from the middleware user registry table /<namespace>/MDW00.
- e) Redefine the method BUILD_PUSH_DATA to prepare user-dependent data to be pushed to the recipients. Each recipient will have a separate set of data collection images (push instance) and moved to the outbound queue.
- f) Redefine the method GET_DATA_FILTER_LIST to enable the data filter function for the class handler.
- g) List the filters supported by this class handler method along with the table name and filter name and pass it to the output table ET_DATA_FILTERS.

2. Redefine the GET method to enable the fetch process for the class handler:

- a) Convert the RFC parameter list into OO parameter format.
- b) Build filter rules from the data object filter service and BAPI input parameters.
- c) Get the outbound message queue data from the push registry cache tables for the MDW user GUID range supplied through the BAPI parameter interface.
- d) Build the output data in OO parameter format.

- e) Cache the exceptions, if any, to the output return table.
- f) In the Agency SAP Framework Configuration portal, navigate to the Mobile Data Object panel and assign the GET class handler to the appropriate mobile data object.

3. Create the BAPI wrapper for the push work order fetch (GET):

- a) Create a new RFC.
- b) Assign the input parameter IS_BAPI_INPUT Type: /SYCLO/CORE_BAPI_INPUT_STR, which is the Syclo BAPI wrapper standard input setting. This structure contains the mobile object data such as the mobile user name, device ID, timestamp from the mobile, mobile data object ID, staging information, session and user GUIDs, etc.
- c) Assign the export parameter ES_BAPI_OUTPUT Type: /SYCLO/CORE_BAPI_OUTPUT_STR, which is the Syclo BAPI wrapper standard output structure. This structure contains the timestamp from SAP to the mobile device, package information, etc.
- d) Assign the appropriate tables parameters. This tab contains all the filter ranges and output data structures.

4. Implement the exchange process for the work order push:

- a) Create a new configuration entry for the exchange process that is utilizing the push.
- b) Enable the push settings and additional filter conditions relevant to the push instance.
- c) Configure the exchange push settings to identify whether the push is active or not, as well as additional filter criteria used to maintain push conditions.

Sending Email Using the Administration & Monitoring Portal

At times, administrators may need to broadcast system messages or other information to a group of users or all users on the system. The Administration portal provides a way to communicate with users through email or text messages rather than through the mobile device, which may not be connected and available.


Note: User email or HTTP addresses and preferences are set in the Administration portal, Administration menu, User Management panel.

- 1. Access the Administration & Monitoring portal.
- 2. Click on the **Monitoring** hyperlink at the top of the screen, and then click the **User Monitor** menu option in the navigation panel.
- 3. Click the **[Send Email]** button.

The Send System Emails screen displays.

4. Check or uncheck the Selected boxes for all desired mobile user GUIDs.
5. Fill out the title, or header of the email.
6. Type in the body of the email in the Email Content box.
7. Click **[Send]** to send the email to all marked user GUIDs or **[Close]** to cancel out of the screen and discard changes.

If the email is sent successfully, a message displays.

 Message has been sent successfully to selected recipients 05/24/2010 13:53:04

Agentry Device Client Branding SDK

Use the Agentry Device Client Branding SDK to brand the device clients before provisioning to users.

Agentry Client Installer and Executable Branding

The Agentry Clients as provided with the SAP® Mobile Platform are branded based on SAP standards. It is possible to rebrand the client components of the Agentry archetype, including both the installers for these components as well as the actual client executables.

Windows PC and Windows Mobile Clients

The branding process for Windows and Windows Mobile Agentry Clients and their installers involves the use of a branding SDK provided with the development tools for SAP Mobile

Platform. Instructions are provided in this guide on the setup and usage of this SDK to rebrand these clients.

Android Clients and iOS Clients

To rebrand the Agentry Clients for iOS and Android, see the information provided on the OpenUI SDK. Included in this SDK are the resources needed to also rebrand the clients for these platforms.

Agentry Windows and Windows Mobile Clients Branding Overview

These instructions will walk you through the steps needed to create branded Agentry Client installers using the Nullsoft NSIS software. This simple process is comprised of the following steps:

- Download and install Nullsoft NSIS
- Create the branding sources
- Customize the installers
- Build and test the branded installers

Branding Agentry Installers

Prerequisites

NSIS (Nullsoft Scriptable Install System) is a professional open source system that is used to create Windows installers. This is the software used to create the branded installers for the Agentry Clients.

In order to brand the Client installers, you will need to have Microsoft's `Cabwiz.exe` installed on your system.

Task

The following main steps walk you through the basics of creating branded installers for the Agentry Client.

1. Download and install Nullsoft NSIS.

- a) Open a browser and go to <http://nsis.sourceforge.net>.

You are brought to the Nullsoft Scriptable Install System Main Page.

- b) Click the **[Download]** link under "Latest NSIS release."

An NSIS setup file is downloaded to your computer.

- c) Run the `nsis-[version]-setup.exe`.

Make sure to note the location where you install the NSIS files because you will need this information later in the branding process.

After you run the NSIS setup program, a full set of NSIS files is installed on your computer. Typically, you will put the NSIS folder inside your Program Files folder. Be sure to note the

location because you will need the full path information when you build the branded installer.

2. Create the branding source. The same Agentry installers that are used to install the products are used to create the branding installer sources for the Client.
 - a) To create the branding Client installer source, run `ClientWinCE_Branding_sdk.exe`.
 Use the switch `"/Branding=[folder name]"` to put the branding Client installer into a folder of your choice.
 - b) When creating a Client installer source, you must also build CAB files for each supported device and for each supported scanner. To do this in a single step, run the command script `cabs/BuildCabs.cmd`.

Note: In order to run the Cabs script, you must have Microsoft's `Cabwiz.exe`.

These steps create all the files necessary to complete the customization of the Client installer.

3. Customize the installers. Once you have the branding source files, you are ready to use those files to modify items such as the company name, product name, and installer names. You must customize the installers for the Client `.nsi` files, as necessary.
 - a) Optionally, make a copy of the `AgentryClientWinCE.nsi` file.

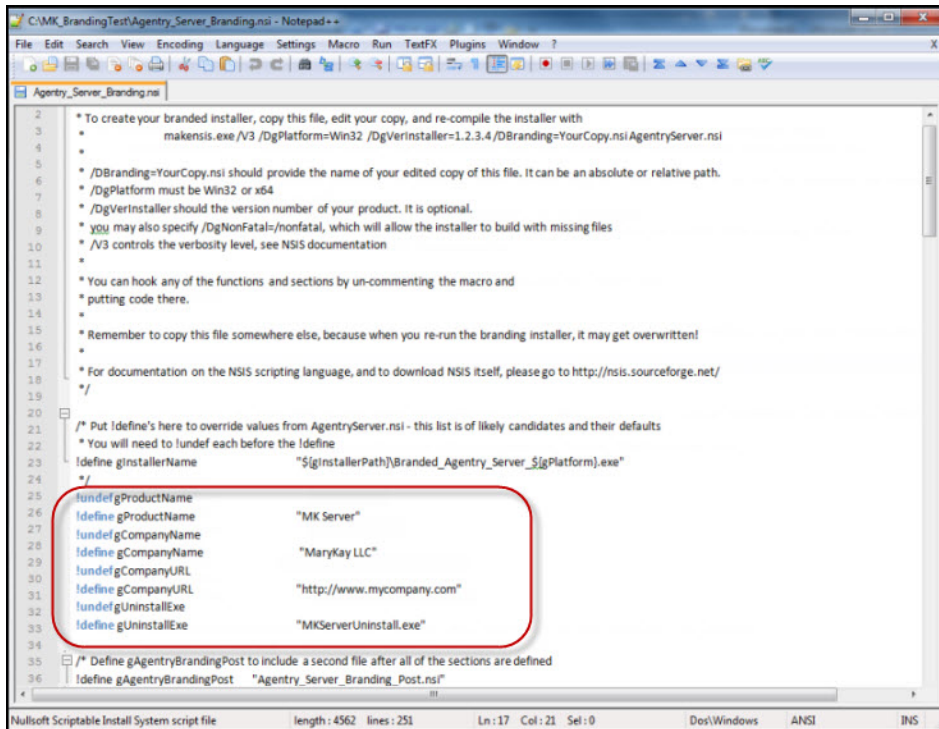
You can optionally make a copy of the files so that you always have the original in case your modified file ever gets overwritten. If you do not make a copy, and need the original files, you can re-install them.

- b) Open the `.nsi` files in a text editor, one at a time, and follow the instructions in the file to modify the common items. These items include things like product name, company name, installer name, company URL, and the uninstaller executable name.

Your modified `.nsi` files should now contain all custom branding details for your Client installer.

Note: If you want to do more advanced branding, including things such as your company logos, you need to go to the Nullsoft Web site for complete programming instructions.

Here is a screen shot of the text editor, highlighting the lines you would typically modify for the Server installer.



```
2  * To create your branded installer, copy this file, edit your copy, and re-compile the installer with
3  * makensis.exe /V3 /DgPlatform=Win32 /DgVerinstaller=1.2.3.4/DBranding=YourCopy.nsiAgentryServer.nsi
4  *
5  * /DBranding=YourCopy.nsi should provide the name of your edited copy of this file. It can be an absolute or relative path.
6  * /DgPlatform must be Win32 or x64
7  * /DgVerinstaller should be the version number of your product. It is optional.
8  * you may also specify /DgNonFatal=/nonfatal, which will allow the installer to build with missing files
9  * /V3 controls the verbosity level, see NSIS documentation
10 *
11 *
12 * You can hook any of the functions and sections by un-commenting the macro and
13 * putting code there.
14 *
15 * Remember to copy this file somewhere else, because when you re-run the branding installer, it may get overwritten!
16 *
17 * For documentation on the NSIS scripting language, and to download NSIS itself, please go to http://nsis.sourceforge.net/
18 */
19
20 /* Put !define's here to override values from AgentryServer.nsi - this list is of likely candidates and their defaults
21 * You will need to !undef each before the !define
22 *
23 !define gInstallerName "$[gInstallerPath]\Branded_Agentry_Server_${gPlatform}.exe"
24 */
25 !undef gProductName
26 !define gProductName "MK Server"
27 !undef gCompanyName
28 !define gCompanyName "MaryKay LLC"
29 !undef gCompanyURL
30 !define gCompanyURL "http://www.mycompany.com"
31 !undef gUninstallExe
32 !define gUninstallExe "MKServerUninstall.exe"
33
34 /* Define gAgentryBrandingPost to include a second file after all of the sections are defined
35 !define gAgentryBrandingPost "Agentry_Server_Branding_Post.nsi"
36
Nullsoft Scriptable Install System script file length: 4562 lines: 251 Ln: 17 Col: 21 Sel: 0 Dos\Windows ANSI INS
```

4. Once you have downloaded and customized the Agentry Client .nsi files, you must recompile the installer to incorporate the branded files.

- a) Open a Command Prompt and follow the instructions to compile the installer.

An example of the command for the Client installer would be:

```
"c:\Program Files (x86)\nsis\makensis.exe" /DgPlatform=Win32 /
DBranding=c:\MK_BrandingFiles\MK_Client_Branding.nsi
AgentryClient.nsi
```

- b) Re-run the Client installers and check that your custom modifications properly display in the applicable installation Wizard.

You now have branded installers for the Agentry Client.

Next

If you want to do more extensive branding, such as adding your company logo, you need to go to the Nullsoft Web site for programming instructions.

Agentry OpenUI API

Learn about the Agentry OpenUI API, which provides the interface to support development of custom controls to be displayed within the Agentry Client. Review the concepts and general procedure, then use the OpenUI API for your target platform.

Agentry Client OpenUI API Overview

Installed with the SAP® Mobile Platform SDK there is the `AgentryClientFramework.zip` archive. This archive includes the OpenUI API components to allow development of custom controls for the Agentry Client. The API is provided for Android, iOS, and Windows client devices. Using this API developers can create customer controls using the native language of the target device client and to display those controls within the detail screens of the mobile application.

The API allows for any field edit type of detail screen fields to be overridden with a custom control. When implemented, the custom control is then displayed in place of the detail screen field. Communication between the control and the Agentry Client is supported, with information about changes to the underlying field, data capture in the custom control, and other similar interactions supported.

In order to make use of the OpenUI, the ZIP archive installed by the SAP Mobile Platform SDK includes Agentry Client resources which can be built with the customer control code to generate a new Agentry Client executable containing the custom control or controls. This executable is then deployed to the client devices in lieu of the standard Agentry Client provided by SAP.

In order to work with the OpenUI API it is necessary to have installed and configured, separate from the items provided in the SAP Mobile Platform SDK, the proper development tools relative the client platform or platforms for which you wish to develop custom controls or make branding changes. Details on these tools are provided in each of the installation sections relative to the OpenUI API for each platform.

It is also necessary, as a developer, to have the skill set to develop in these native languages. Information provided here assumes the developer is experienced in developing in the language (Objective C, Java, Visual Basic, C#, etc.) corresponding to the client device type for which the controls are being created.

OpenUI Replaces the Agentry Client SDK and the ActiveX API for New Development

In previous releases of the Agentry Mobile Platform (6.0 and prior) as well as the Agentry archetype in SAP Mobile Platform 2.3, the option to create custom controls was supported with the ActiveX API provided as a part of the Agentry Client SDK. This API is still available within the SAP Mobile SDK and is provided for backwards compatibility of existing

implementations. Applications making use of the ActiveX API can be migrated to SAP Mobile Platform 3.0 without modification to the ActiveX custom controls.

Going forward, there will be no further functional advancement of the ActiveX API. It is highly recommended that all new implementations in which custom controls are to be added make use of the OpenUI API. First, ActiveX is only available for custom control development on Windows desktop and Windows mobile client devices. The OpenUI API includes support for Windows, iOS, and Android devices. Second, later versions of Windows Mobile do not provide support for ActiveX in the same manner and the ActiveX API provided for Agentry applications are somewhat more challenging to work with.

OpenUI SDK Concepts, Usage and Guidance

The OpenUI provided within the SAP Mobile Platform SDK includes three API's, one for each of the platforms Android, iOS, and Windows .NET. Each of these API's provides the interface to support the development of custom controls to be displayed within the Agentry Client. These controls override the default display and behavior of the detail screen fields that are a part of the Agentry application project.

When looking to make use of this SDK, you should first investigate the standard field edit types available to you to verify the behavior you desire is not already one which can be defined within the application project. Assuming there is a need, however, the OpenUI SDK can be used to create almost limitless variations in the user interface of the Agentry Client.

In addition to the necessary code written to create the custom control, it is also necessary to make modifications to the Agentry application project within the Agentry Editor, specifically within the field definition. These changes include modifying the attributes of the field to be overridden, providing information about the class containing the override code, as well as specifying the values and actions available within the Agentry application project the custom control can access and execute, and finally the values available to the Agentry Client from the custom control.

General Procedure to Create Custom Controls

Creating a custom control for your mobile application includes the following tasks:

1. Install the OpenUI SDK API component for the target client platform, per the instructions provided in the guide *Setting Up the Development Environment - Agentry Toolkit*
2. Using the Agentry Editor modify the application project by defining the detail screen field to be overridden by the custom control This includes specifying the Extension Adapter Name, as well as the Extension Values, Agentry Values, and Agentry Actions.
3. Using the IDE appropriate for the client platform, create the custom control using the OpenUI API.
4. Build the project within the IDE, which will result in either a full Agentry Client build that includes the custom control logic (Android, iOS); or a DLL containing the customer control logic to be deployed with the Agentry Client executable (Windows .NET).

5. Deploy the Agentry Client to a device and test all behaviors. Make needed changes based on testing and repeat the build and deploy steps until the functionality is considered fully developed and ready for distribution.
6. Distribute the application to the client devices according to the standard procedures of the client device platform. (Continue reading for more information on distribution.)

Distributing the Agentry Client With Custom Controls

Once custom controls have been developed the Agentry Client must be rebuilt or repackaged, depending on the client platform, in order to distribute them to the mobile users. For both Android and iOS devices, this requires the Agentry Client to be rebuilt and resigned. For Windows devices, the Agentry Client can be repackaged using the Agentry Client Branding SDK.

The projects included in the OpenUI SDK for both Android and iOS are structured for this purpose. Included in both are resource projects which can be modified to both resign the application as well as rebrand it as needed. When built a distributable application (. apk file for Android; . ipa file for iOS) is created.

Developer Requirements and Responsibilities

As a developer of custom controls using the OpenUI SDK, you are expected to provide certain information about the control to the Agentry Client at runtime. Of course this includes the field's behavior itself, including all display aspects and behaviors. Additionally, it includes items such as size of the control displayed on the detail screen, including whether this size is dictated by the Agentry Client via the sizing attributes of the field definition or by the custom control. You must also specify and create logic for the values available to the Agentry Client, including the value used to set the target property of the detail screen field being overridden.

Included in this behavior for all custom controls, regardless of the field edit type which they override, should be appropriate behaviors related to the various states a field can be in. This includes whether the field is enabled or disabled, and whether the field is visible or hidden. An enabled field and a disabled field can both still be visible, so the custom control should then allow for this and be displayed appropriately. As a basic example, some types of controls are grayed out or have an otherwise different appearance to indicate visually to the user the field cannot be interacted with. If the field is not visible, the Agentry Client will not display the custom control to the user.

In such a situation, the logic should account for this state and handle any values it would otherwise display or make available appropriately. In the event of a non-visible field that is in an enabled, state, the Agentry Client will still enforce any requirements regarding the value returned by the field, for example a minimum string length. In such a situation, a reasonable default value should be provided by the custom control. Disabled fields, regardless of visible state, will not have their values validated. Note that a read-only field is not the same as a disabled field within the Agentry Client. The enabled or disabled state is controlled by the Enabled attribute for a field definition. This is typically set at runtime on the Agentry Client

based on a return value from a rule. As such, it is important to have a full understanding of the field definition's defined behavior while implementing the custom control logic.

Runtime Behavior of the Agentry Client With Custom Controls

The Agentry Client will look to load the referenced custom control, based on the settings of the External Adapter Name attribute within the detail screen field definition, when that field is displayed on it's parent details screen. If it cannot find an adaptor with the referenced named, it will display the field defined within the Agentry application project according to that field's edit type.

When a custom control is displayed, the user will see the custom control on the detail screen as if it were a built in control. The behavior of the custom control is then dictated by the logic the you have implemented for that control.

Agentry OpenUI API for Android

Use the OpenUI API for Android to add custom controls to Agentry applications.

com.sap.mobile.platform package

client package

openui package

adapters package

BooleanDisplayAdapter class

The class that any extension class for boolean display needs to extend.

Syntax

```
public abstract class BooleanDisplayAdapter extends  
FieldAdapter
```

Members

All members of BooleanDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(BooleanDisplayModel, Context)</i> on page 998	Called to initialize the extension with its model and Android context.

Modifier and Type	Method	Description
public void	<i>valueChanged(boolean)</i> on page 998	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.

Modifier and Type	Member	Description
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(BooleanDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( BooleanDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(boolean) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( boolean value )
```

Parameters

- **value** – the new value for the field

BooleanEditAdapter class

The class that any extension class for boolean edit needs to extend.

Syntax

```
public abstract class BooleanEditAdapter extends FieldAdapter
```

Members

All members of BooleanEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(BooleanEditModel, Context)</i> on page 1000	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(boolean)</i> on page 1001	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.

Modifier and Type	Member	Description
public boolean	<i>isAgencyDisplayingValidationFailure()</i> on page 1034	Called to ask if Agency should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agency field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agency field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agency field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(BooleanEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( BooleanEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agency.
- **context** – Android context to use

valueChanged(boolean) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( boolean value )
```

Parameters

- **value** – the new value for the field

ButtonDisplayAdapter class

The class that any extension class for button display needs to extend.

Syntax

```
public abstract class ButtonDisplayAdapter extends  
FieldAdapter
```

Members

All members of ButtonDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public void	<i>buttonImageChanged(Agentry-Image)</i> on page 1003	This method notifies the extension that the field's image has changed.
public abstract void	<i>initialize(ButtonDisplayModel, Context)</i> on page 1003	Called to initialize the extension with its model and Android context.
public void	<i>selectedStateChanged(boolean)</i> on page 1003	This method is called when the field's selected state has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.

Modifier and Type	Member	Description
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

buttonImageChanged(AgentryImage) method

This method notifies the extension that the field's image has changed.

Syntax

```
public void buttonImageChanged ( AgentryImage newImage )
```

Parameters

- **newImage** – the new image to display on the button.

initialize(ButtonDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( ButtonDisplayModel model ,  
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

selectedStateChanged(boolean) method

This method is called when the field's selected state has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void selectedStateChanged ( boolean selected )
```

Parameters

- **selected** – the new selected state for the field

DateAndTimeDisplayAdapter class

The class that any extension class for time and date display needs to extend.

Syntax

```
public abstract class DateAndTimeDisplayAdapter extends  
FieldAdapter
```

Members

All members of DateAndTimeDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(DateAndTimeDisplayModel, Context)</i> on page 1005	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(GregorianCalendar)</i> on page 1005	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.

Modifier and Type	Member	Description
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(DateAndTimeDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DateAndTimeDisplayModel
model , Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(GregorianCalendar) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( GregorianCalendar value )
```

Parameters

- **value** – the new value for the field

DateAndTimeEditAdapter class

The class that any extension class for time and date edit needs to extend.

Syntax

```
public abstract class DateAndTimeEditAdapter extends
FieldAdapter
```

Members

All members of *DateAndTimeEditAdapter*, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(DateAndTimeEdit-Model, Context)</i> on page 1007	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(GregorianCalendar)</i> on page 1008	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from *FieldAdapter*

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agency to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agency will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agency to get the value for the specified string.

Modifier and Type	Member	Description
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(DateAndTimeEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DateAndTimeEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(GregorianCalendar) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( GregorianCalendar value )
```

Parameters

- **value** – the new value for the field

DateDisplayAdapter class

The class that any extension class for date display needs to extend.

Syntax

```
public abstract class DateDisplayAdapter extends FieldAdapter
```

Members

All members of DateDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(DateDisplayModel, Context)</i> on page 1010	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(GregorianCalendar)</i> on page 1010	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.

Modifier and Type	Member	Description
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(DateDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DateDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(GregorianCalendar) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( GregorianCalendar value )
```

Parameters

- **value** – the new value for the field

DateEditAdapter class

The class that any extension class for date edit needs to extend.

Syntax

```
public abstract class DateEditAdapter extends FieldAdapter
```

Members

All members of DateEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(DateEditModel, Context)</i> on page 1012	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(GregorianCalendar)</i> on page 1012	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.

Modifier and Type	Member	Description
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(DateEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DateEditModel model , Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(GregorianCalendar) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( GregorianCalendar value )
```

Parameters

- **value** – the new value for the field

DecimalDisplayAdapter class

The class that any extension class for decimal display needs to extend.

Syntax

```
public abstract class DecimalDisplayAdapter extends
FieldAdapter
```

Members

All members of DecimalDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(DecimalDisplayModel, Context)</i> on page 1014	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(double)</i> on page 1015	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.

Modifier and Type	Member	Description
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(DecimalDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DecimalDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(double) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( double value )
```

Parameters

- **value** – the new value for the field

DecimalEditAdapter class

The class that any extension class for decimal edit needs to extend.

Syntax

```
public abstract class DecimalEditAdapter extends FieldAdapter
```

Members

All members of DecimalEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(DecimalEditModel, Context)</i> on page 1017	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(double)</i> on page 1017	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.

Modifier and Type	Member	Description
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(DecimalEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DecimalEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(double) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( double value )
```

Parameters

- **value** – the new value for the field

DurationDisplayAdapter class

The class that any extension class for duration display needs to extend.

Syntax

```
public abstract class DurationDisplayAdapter extends
FieldAdapter
```

Members

All members of DurationDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public void	<i>fractionalHourValueChanged(double)</i> on page 1019	This method is called when the field's underlying value has changed and the UI needs to be updated to display the correct value.
public abstract void	<i>initialize(DurationDisplayModel, Context)</i> on page 1020	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(int)</i> on page 1020	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.

Modifier and Type	Member	Description
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

Usage

Depending on the display format, the host will call either `valueChanged(int)` or `fractionalHourValueChanged(double)` to notify the adapter of updates.

fractionalHourValueChanged(double) method

This method is called when the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void fractionalHourValueChanged ( double value )
```

Parameters

- **value** – the new value for the field (in hours)

Usage

This should be used when the `DurationDisplayFormat` is set to `DecHour`, which can be checked by calling `DurationDisplayModel.getDurationDisplayFormat()`.

initialize(DurationDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DurationDisplayModel model ,  
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(int) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( int value )
```

Parameters

- **value** – the new value for the field (in seconds)

DurationEditAdapter class

The class that any extension class for duration edit needs to extend.

Syntax

```
public abstract class DurationEditAdapter extends  
FieldAdapter
```

Members

All members of DurationEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public void	<i>fractionalHourValue-Changed(double)</i> on page 1022	This method is called when the field's underlying value has changed and the UI needs to be updated to display the correct value.

Modifier and Type	Method	Description
public abstract void	<i>initialize(DurationEditModel, Context)</i> on page 1023	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(int)</i> on page 1023	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.

Modifier and Type	Member	Description
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

Usage

Depending on the display format, the host will call either `valueChanged(int)` or `fractionalHourValueChanged(double)` to notify the adapter of updates.

fractionalHourValueChanged(double) method

This method is called when the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void fractionalHourValueChanged ( double value )
```

Parameters

- **value** – the new value for the field (in hours)

Usage

This should be used when the `DurationDisplayFormat` is set to `DecHour`.

initialize(DurationEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( DurationEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(int) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( int value )
```

Parameters

- **value** – the new value for the field (in seconds)

EmbeddedImageDisplayAdapter class

The class that any extension class for embedded image display needs to extend.

Syntax

```
public abstract class EmbeddedImageDisplayAdapter extends
FieldAdapter
```

Members

All members of EmbeddedImageDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public void	<i>imageChanged()</i> on page 1025	This method is called when the field's underlying image has changed.
public void	<i>imageSelectionChanged()</i> on page 1025	This method is called when the field's underlying cell selection has changed.

Modifier and Type	Method	Description
public abstract void	<i>initialize(EmbeddedImageDisplayModel, Context)</i> on page 1025	Called to initialize the extension with its model and Android context.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.

Modifier and Type	Member	Description
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

imageChanged() method

This method is called when the field's underlying image has changed.

Syntax

```
public void imageChanged ()
```

Usage

It only notifies the extension that there is a change. It is the extension's responsibility to call back into the host model to get the new image when it's ready.

imageSelectionChanged() method

This method is called when the field's underlying cell selection has changed.

Syntax

```
public void imageSelectionChanged ()
```

Usage

It only notifies the extension that there is a change. It is the extension's responsibility to call back into the host model to get the selected cells.

initialize(EmbeddedImageDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( EmbeddedImageDisplayModel  
model , Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

ExternalDataDisplayAdapter class

The class that any extension class for external data display needs to extend.

Syntax

```
public abstract class ExternalDataDisplayAdapter extends
FieldAdapter
```

Members

All members of ExternalDataDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(ExternalDataDisplay-Model, Context)</i> on page 1027	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(String)</i> on page 1028	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.

Modifier and Type	Member	Description
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(ExternalDataDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( ExternalDataDisplayModel
model , Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(String) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( String value )
```

Parameters

- **value** – the new value for the field

ExternalDataEditAdapter class

The class that any extension class for external data edit needs to extend.

Syntax

```
public abstract class ExternalDataEditAdapter extends  
FieldAdapter
```

Members

All members of ExternalDataEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(ExternalDataEditModel, Context)</i> on page 1030	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(String)</i> on page 1030	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.

Modifier and Type	Member	Description
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(ExternalDataEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( ExternalDataEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(String) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( String value )
```

Parameters

- **value** – the new value for the field

FieldAdapter class

This is the abstract class all open UI adapter abstract classes derive from.

Syntax

```
public abstract class FieldAdapter
```

Derived classes

- *BooleanDisplayAdapter* on page 996
- *BooleanEditAdapter* on page 999
- *ButtonDisplayAdapter* on page 1001
- *DateAndTimeDisplayAdapter* on page 1003

- *DateAndTimeEditAdapter* on page 1006
- *DateDisplayAdapter* on page 1008
- *DateEditAdapter* on page 1010
- *DecimalDisplayAdapter* on page 1013
- *DecimalEditAdapter* on page 1015
- *DurationDisplayAdapter* on page 1017
- *DurationEditAdapter* on page 1020
- *EmbeddedImageDisplayAdapter* on page 1023
- *ExternalDataDisplayAdapter* on page 1026
- *ExternalDataEditAdapter* on page 1028
- *IntegerDisplayAdapter* on page 1037
- *IntegerEditAdapter* on page 1039
- *LabelDisplayAdapter* on page 1041
- *LocationDisplayAdapter* on page 1044
- *LocationEditAdapter* on page 1046
- *StringDisplayAdapter* on page 1048
- *StringEditAdapter* on page 1051
- *TimeDisplayAdapter* on page 1053
- *TimeEditAdapter* on page 1055

Members

All members of FieldAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public AutocompleteBehavior	<i>getAutocompleteBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutocompleteBehavior()</i> is overridden to return <i>AutocompleteBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.

Modifier and Type	Method	Description
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

getAutosizeBehavior() method

Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.

Syntax

```
public AutosizeBehavior getAutosizeBehavior ()
```

Returns

the autosize behavior

Usage

If this returns `Autosize_WrapContent` and the field's height in the editor is set to "Auto", Agentry will call `getContentHeightForAutosizing` to get the needed height from

the extension. If this returns `Autosize_FillVisible`, Agentry will size the field to fill the available screen area. If this returns `Autosize_None`, Agentry will size the extension without asking.

This works in conjunction with `FieldModel.isAutosizeSupported()` which allows the extension to ask if the editor definitions support autosizing.

`autosizeBehavior()` is Agentry's way of asking if the extension is able to handle autosizing.

getContentHeightForAutosizing(int) method

Agentry will call this method if `getAutosizeBehavior()` is overridden to return `AutosizeBehavior.Autosize_WrapContent`.

Syntax

```
public int getContentHeightForAutosizing ( int width )
```

Parameters

- **width** – the width of the extension's content area in pixels

Returns

the height needed in pixels for the extension to show the current data

Usage

Agentry passes in the extension view's width in pixels. The extension then needs measure the height needed for the content in pixels and return it.

getExtensionString(String) method

Called by the Agentry to get the value for the specified string.

Syntax

```
public String getExtensionString ( String name )
```

Parameters

- **name** – the string that Agentry is requesting

Returns

the value the extension determines based on the specified key

Usage

In the definitions, there are specified keys. The string passed in is a key, the value is returned from the extension.

getView() method

Called to get the Android View that will be added as a subview to the Agentry layout.

Syntax

```
public abstract View getView ()
```

Returns

Android view to display

Usage

This will be called one time from Agentry.

isAgentryDisplayingLabel() method

Called to ask if Agentry should handle displaying the label.

Syntax

```
public boolean isAgentryDisplayingLabel ()
```

Returns

true if the Agentry should handle displaying the label, false if the extension will handle displaying the label

Usage

If this method returns true, Agentry will handle displaying the label, including hyperlink functionality. If this method returns false, the extension takes responsibility for the label (and is free to just not bother with it).

By default, the extension is responsible for displaying the label.

isAgentryDisplayingValidationFailure() method

Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.

Syntax

```
public boolean isAgentryDisplayingValidationFailure ()
```

Returns

true if the Agentry should handle displaying the validation failure text, false if the extension will handle displaying the validation failure text

Usage

If this method returns true, Agentry will handle displaying the field validation failure text. If this method returns false, the extension takes responsibility for the field validation failure text.

By default, Agentry is responsible for displaying the validation failure text.

onActivityResult(int, int, Intent) method

Called from activity launched through `FieldModel.launchActivity(Intent intent, int requestCode)` Allows extension the opportunity to handle any result from the now closed activity.

Syntax

```
public void onActivityResult ( int requestCode , int resultCode ,
Intent intent )
```

Parameters

- **requestCode** – the integer request code that was supplied to the activity
- **resultCode** – the activity result
- **intent** – the means to get at the data

setEnabled(boolean) method

Called to inform the extension that the Agentry field's enable state has changed.

Syntax

```
public void setEnabled ( boolean enabled )
```

Parameters

- **enabled** – true to indicate it is enabled, false to indicate it is disabled

setHyperlinkEnabled(boolean) method

Called to inform the extension that the enabled state of the label hyperlink action has changed.

Syntax

```
public void setHyperlinkEnabled ( boolean enabled )
```

Parameters

- **enabled** – true if hyperlink is enabled, false if hyperlink is disabled

Usage

Only called if the extension is handling the label functionality and a hyperlink is defined.

setValid(boolean, String) method

Called to inform the extension that the Agentry field's valid state has changed.

Syntax

```
public void setValid ( boolean valid ,   String validationMessage )
```

Parameters

- **valid** – true if the field value is valid, false for invalid
- **validationMessage** – the message to display to the user

Usage

The field has either become invalid and the user needs to be informed with the validation message, or it has become valid and any previously displayed validation failure text needs to be hidden.

The validation message will contain information that tells the user why their field is invalid.

setVisible(boolean) method

Called to inform the extension that the Agentry field's visibility has changed.

Syntax

```
public void setVisible ( boolean visible )
```

Parameters

- **visible** – true to indicate it is visible, false to indicate it is hidden

Usage

The view for the extension will be shown or hidden automatically. The extension will receive this call to do any additional actions it needs to do when the visible state changes.

updateLabel(String) method

Called to inform the extension that the label text has changed.

Syntax

```
public void updateLabel ( String label )
```

Parameters

- **label** – the new value for the label

Usage

Only called if the extension is handling the label functionality and the label is defined with a rule.

IntegerDisplayAdapter class

The class that any extension class for integer display needs to extend.

Syntax

```
public abstract class IntegerDisplayAdapter extends
FieldAdapter
```

Members

All members of IntegerDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(IntegerDisplayModel, Context)</i> on page 1038	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(int)</i> on page 1039	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.

Modifier and Type	Member	Description
public boolean	<i>isAgencyDisplayingValidationFailure()</i> on page 1034	Called to ask if Agency should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agency field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agency field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agency field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(IntegerDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( IntegerDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agency.
- **context** – Android context to use

valueChanged(int) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( int value )
```

Parameters

- **value** – the new value for the field

IntegerEditAdapter class

The class that any extension class for integer editing needs to extend.

Syntax

```
public abstract class IntegerEditAdapter extends FieldAdapter
```

Members

All members of IntegerEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(IntegerEditModel, Context)</i> on page 1041	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(int)</i> on page 1041	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.

Modifier and Type	Member	Description
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(IntegerEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( IntegerEditModel model ,  
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(int) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( int value )
```

Parameters

- **value** – the new value for the field

LabelDisplayAdapter class

The class that any extension class for label display needs to extend.

Syntax

```
public abstract class LabelDisplayAdapter extends  
FieldAdapter
```

Members

All members of LabelDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(LabelDisplayModel, Context)</i> on page 1043	Called to initialize the extension with its model and Android context.

Modifier and Type	Method	Description
public void	<i>valueChanged(String)</i> on page 1043	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.

Modifier and Type	Member	Description
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

Usage

This extension is responsible for displaying the label. Its `getView()` method should return the label view. The control's value is considered to be the label text.

initialize(LabelDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( LabelDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(String) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( String value )
```

Parameters

- **value** – the new value for the field

LocationDisplayAdapter class

The class that any extension class for location display needs to extend.

Syntax

```
public abstract class LocationDisplayAdapter extends
FieldAdapter
```

Members

All members of LocationDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(LocationDisplayModel, Context)</i> on page 1045	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(AgentryLocation)</i> on page 1046	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.

Modifier and Type	Member	Description
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(LocationDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( LocationDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(AgentryLocation) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( AgentryLocation value )
```

Parameters

- **value** – the new value for the field

LocationEditAdapter class

The class that any extension class for location edit needs to extend.

Syntax

```
public abstract class LocationEditAdapter extends  
FieldAdapter
```

Members

All members of LocationEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(LocationEditModel, Context)</i> on page 1048	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(AgentryLocation)</i> on page 1048	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.

Modifier and Type	Member	Description
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(LocationEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( LocationEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(AgentryLocation) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( AgentryLocation value )
```

Parameters

- **value** – the new value for the field

StringDisplayAdapter class

The class that any extension class for string display needs to extend.

Syntax

```
public abstract class StringDisplayAdapter extends
FieldAdapter
```

Members

All members of StringDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(StringDisplayModel, Context)</i> on page 1050	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(String)</i> on page 1050	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutosizeBehavior()</i> is overridden to return <i>AutosizeBehavior.Autosize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.

Modifier and Type	Member	Description
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(StringDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( StringDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(String) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( String value )
```

Parameters

- **value** – the new value for the field

StringEditAdapter class

The class that any extension class for string edit needs to extend.

Syntax

```
public abstract class StringEditAdapter extends FieldAdapter
```

Members

All members of StringEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(StringEditModel, Context)</i> on page 1052	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(String)</i> on page 1053	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.

Modifier and Type	Member	Description
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through FieldModel.launchActivity(Intent intent, int requestCode) Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(StringEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize (StringEditModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.

- **context** – Android context to use

valueChanged(String) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( String value )
```

Parameters

- **value** – the new value for the field

TimeDisplayAdapter class

The class that any extension class for time display needs to extend.

Syntax

```
public abstract class TimeDisplayAdapter extends FieldAdapter
```

Members

All members of TimeDisplayAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(TimeDisplayModel, Context)</i> on page 1055	Called to initialize the extension with its model and Android context.
public void	<i>valueChanged(GregorianCalendar)</i> on page 1055	This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutosizeBehavior	<i>getAutosizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.

Modifier and Type	Member	Description
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(TimeDisplayModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( TimeDisplayModel model ,
Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(GregorianCalendar) method

This method is called by the to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( GregorianCalendar value )
```

Parameters

- **value** – the new value for the field

TimeEditAdapter class

The class that any extension class for time edit needs to extend.

Syntax

```
public abstract class TimeEditAdapter extends FieldAdapter
```

Members

All members of TimeEditAdapter, including inherited members. **Methods**

Modifier and Type	Method	Description
public abstract void	<i>initialize(TimeEditModel, Context)</i> on page 1057	Called to initialize the extension with its model and Android context.

Modifier and Type	Method	Description
public void	<i>valueChanged(GregorianCalendar)</i> on page 1057	This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Inherited members from FieldAdapter

Modifier and Type	Member	Description
public AutoSizeBehavior	<i>getAutoSizeBehavior()</i> on page 1032	Called by Agentry to ask if the extension view needs to auto-size to accommodate the displayed data.
public int	<i>getContentHeightForAutosizing(int)</i> on page 1033	Agentry will call this method if <i>getAutoSizeBehavior()</i> is overridden to return <i>AutoSizeBehavior.AutoSize_WrapContent</i> .
public String	<i>getExtensionString(String)</i> on page 1033	Called by the Agentry to get the value for the specified string.
public abstract View	<i>getView()</i> on page 1034	Called to get the Android View that will be added as a subview to the Agentry layout.
public boolean	<i>isAgentryDisplayingLabel()</i> on page 1034	Called to ask if Agentry should handle displaying the label.
public boolean	<i>isAgentryDisplayingValidationFailure()</i> on page 1034	Called to ask if Agentry should handle displaying validation failure text or leave it to the extension.
public void	<i>onActivityResult(int, int, Intent)</i> on page 1035	Called from activity launched through <i>FieldModel.launchActivity(Intent intent, int requestCode)</i> Allows extension the opportunity to handle any result from the now closed activity.

Modifier and Type	Member	Description
public void	<i>setEnabled(boolean)</i> on page 1035	Called to inform the extension that the Agentry field's enable state has changed.
public void	<i>setHyperlinkEnabled(boolean)</i> on page 1035	Called to inform the extension that the enabled state of the label hyperlink action has changed.
public void	<i>setValid(boolean, String)</i> on page 1036	Called to inform the extension that the Agentry field's valid state has changed.
public void	<i>setVisible(boolean)</i> on page 1036	Called to inform the extension that the Agentry field's visibility has changed.
public void	<i>updateLabel(String)</i> on page 1036	Called to inform the extension that the label text has changed.

initialize(TimeEditModel, Context) method

Called to initialize the extension with its model and Android context.

Syntax

```
public abstract void initialize ( TimeEditModel model , Context context )
```

Parameters

- **model** – a reference to the object that implements the model. This will be the extension's means of calling into Agentry.
- **context** – Android context to use

valueChanged(GregorianCalendar) method

This method is called by the host to inform the adapter that the field's underlying value has changed and the UI needs to be updated to display the correct value.

Syntax

```
public void valueChanged ( GregorianCalendar value )
```

Parameters

- **value** – the new value for the field

models package

BooleanDisplayModel interface

Interface given to a boolean display extension object so it can call back into the host.

Syntax

```
public interface BooleanDisplayModel extends FieldModel
```

Derived classes

- *BooleanEditModel* on page 1059

Members

All members of BooleanDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public boolean	<i>getValue()</i> on page 1059	Returns the current value of the field.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.

Modifier and Type	Member	Description
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch a new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the current value of the field.

Syntax

```
public boolean getValue ()
```

Returns

the field value

BooleanEditModel interface

Interface given to a boolean edit extension object so it can call back into the host.

Syntax

```
public interface BooleanEditModel extends BooleanDisplayModel
```

Members

All members of BooleanEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public ProcessInputReturn	<i>processInput(boolean)</i> on page 1061	Processes the input of the field.

Inherited members from BooleanDisplayModel

Modifier and Type	Member	Description
public boolean	<i>getValue()</i> on page 1059	Returns the current value of the field.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

processInput(boolean) method

Processes the input of the field.

Syntax

```
public ProcessInputReturn processInput ( boolean value )
```

Parameters

- **value** – the value to process

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

ButtonDisplayModel interface

Interface given to a button display extension object so it can call back into the host.

Syntax

```
public interface ButtonDisplayModel extends FieldModel
```

Members

All members of `ButtonDisplayModel`, including inherited members. **Methods**

Modifier and Type	Method	Description
public AgentryImage	<i>getButtonImage()</i> on page 1062	Returns the image associated with the button.
public String	<i>getButtonText()</i> on page 1063	Returns the text that the button should display.
public ButtonType	<i>getButtonType()</i> on page 1063	Returns the button type.
public boolean	<i>hasAction()</i> on page 1063	Returns whether or not there is an action tied to the button.
public boolean	<i>isButton.Selected()</i> on page 1063	Returns whether or not the button is selected.
public ProcessInputReturn	<i>processInput()</i> on page 1064	Called to process the button push.

Inherited members from `FieldModel`

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getButtonImage() method

Returns the image associated with the button.

Syntax

```
public AgentryImage getButtonImage ()
```

Returns

the associated image

getButtonText() method

Returns the text that the button should display.

Syntax

```
public String getButtonText ()
```

Returns

the button text

getButtonType() method

Returns the button type.

Syntax

```
public ButtonType getButtonType ()
```

Returns

the type of button

Usage

Possible types are checkbox, radio and push button.

hasAction() method

Returns whether or not there is an action tied to the button.

Syntax

```
public boolean hasAction ()
```

Returns

true if action is supported, false if action is not supported

isButtonSelected() method

Returns whether or not the button is selected.

Syntax

```
public boolean isButtonSelected ()
```

Returns

true if selected, false if not selected

processInput() method

Called to process the button push.

Syntax

```
public ProcessInputReturn processInput ()
```

Returns

result of processing the push

Usage

Returns a `ProcessInputReturn` representing the result of processing the push.

DateAndTimeDisplayModel interface

Interface given to a time and date display extension object so it can call back into the host.

Syntax

```
public interface DateAndTimeDisplayModel extends FieldModel
```

Derived classes

- *DateAndTimeEditModel* on page 1065

Members

All members of `DateAndTimeDisplayModel`, including inherited members. **Methods**

Modifier and Type	Method	Description
public <code>GregorianCalendar</code>	<i>getValue()</i> on page 1065	Returns the field's current date and time value.

Inherited members from `FieldModel`

Modifier and Type	Member	Description
public <code>ActionResult</code>	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public <code>ActionResult</code>	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public <code>ActionEnableType</code>	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public <code>String</code>	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.

Modifier and Type	Member	Description
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutoSizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the field's current date and time value.

Syntax

```
public GregorianCalendar getValue ()
```

Returns

the date

DateAndTimeEditModel interface

Interface given to a time and date edit extension object so it can call back into the host.

Syntax

```
public interface DateAndTimeEditModel extends
DateAndTimeDisplayModel
```

Members

All members of `DateAndTimeEditModel`, including inherited members. **Methods**

Modifier and Type	Method	Description
public <code>ProcessInputReturn</code>	<i>processInput(<code>GregorianCalendar</code>)</i> on page 1067	Processes the date and time input.

Inherited members from `DateAndTimeDisplayModel`

Modifier and Type	Member	Description
public <code>GregorianCalendar</code>	<i>getValue()</i> on page 1065	Returns the field's current date and time value.

Inherited members from `FieldModel`

Modifier and Type	Member	Description
public <code>ActionResult</code>	<i>executeAgentryAction(<code>String</code>)</i> on page 1089	Asks Agentry to execute the action specified by name.
public <code>ActionResult</code>	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public <code>ActionEnableType</code>	<i>getAgentryActionEnableState(<code>String</code>)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public <code>String</code>	<i>getAgentryString(<code>String</code>)</i> on page 1090	Asks Agentry for a specific string value.
public <code>String</code>	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public <code>boolean</code>	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public <code>boolean</code>	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public <code>boolean</code>	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.

Modifier and Type	Member	Description
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

processInput(GregorianCalendar) method

Processes the date and time input.

Syntax

```
public ProcessInputReturn processInput ( GregorianCalendar
dateAndTime )
```

Parameters

- **dateAndTime** – the date and time value to process

Returns

result based on the value passed in

Usage

Returns a ProcessInputReturn representing the result of processing the input.

DateDisplayModel interface

Interface given to a date display extension object so it can call back into the host.

Syntax

```
public interface DateDisplayModel extends FieldModel
```

Derived classes

- *DateEditModel* on page 1069

Members

All members of DateDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public GregorianCalendar	<i>getValue()</i> on page 1069	Returns the field's current date value.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the field's current date value.

Syntax

```
public GregorianCalendar getValue ()
```

Returns

the date

DateEditModel interface

Interface given to a date edit extension object so it can call back into the host.

Syntax

```
public interface DateEditModel extends DateDisplayModel
```

Members

All members of DateEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public ProcessInputReturn	<i>processInput(GregorianCalendar)</i> on page 1070	Processes the entered date value.

Inherited members from DateDisplayModel

Modifier and Type	Member	Description
public GregorianCalendar	<i>getValue()</i> on page 1069	Returns the field's current date value.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgencyAction(String)</i> on page 1089	Asks Agency to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agency to run the field's hyperlink action.
public ActionEnableType	<i>getAgencyActionEnableState(String)</i> on page 1090	Asks Agency what the current enable state is for the action specified by name.
public String	<i>getAgencyString(String)</i> on page 1090	Asks Agency for a specific string value.

Modifier and Type	Member	Description
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutoSizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

processInput(GregorianCalendar) method

Processes the entered date value.

Syntax

```
public ProcessInputReturn processInput ( GregorianCalendar
date )
```

Parameters

- **date** – the value to process

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

DecimalDisplayModel interface

Interface given to an decimal display extension object so it can call back into the host.

Syntax

```
public interface DecimalDisplayModel extends FieldModel
```

Derived classes

- *DecimalEditModel* on page 1072

Members

All members of *DecimalDisplayModel*, including inherited members. **Methods**

Modifier and Type	Method	Description
public double	<i>getValue()</i> on page 1072	Gets the current value.

Inherited members from *FieldModel*

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.

Modifier and Type	Member	Description
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Gets the current value.

Syntax

```
public double getValue ()
```

Returns

the current value

DecimalEditModel interface

Interface given to an decimal edit extension object so it can call back into the host.

Syntax

```
public interface DecimalEditModel extends DecimalDisplayModel
```

Members

All members of DecimalEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public double	<i>getMaximumValue()</i> on page 1074	The maximum value accepted for the decimal field.
public double	<i>getMinimumValue()</i> on page 1074	The minimum value accepted for the decimal field.
public ProcessInputReturn	<i>processInput(double)</i> on page 1074	Process the current double input.

Inherited members from DecimalDisplayModel

Modifier and Type	Member	Description
public double	<i>getValue()</i> on page 1072	Gets the current value.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getMaximumValue() method

The maximum value accepted for the decimal field.

Syntax

```
public double getMaximumValue ()
```

Returns

maximum value

getMinimumValue() method

The minimum value accepted for the decimal field.

Syntax

```
public double getMinimumValue ()
```

Returns

minimum value

processInput(double) method

Process the current double input.

Syntax

```
public ProcessInputReturn processInput ( double value )
```

Parameters

- **value** – input value

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

DurationDisplayModel interface

Interface given to a duration display extension object so it can call back into the host.

Syntax

```
public interface DurationDisplayModel extends FieldModel
```

Derived classes

- *DurationEditModel* on page 1077

Members

All members of `DurationDisplayModel`, including inherited members. **Methods**

Modifier and Type	Method	Description
public <code>DurationDisplayFormat</code>	<i>getDurationDisplayFormat()</i> on page 1076	Returns the display format specified for the duration.
public double	<i>getFractionalHourValue()</i> on page 1076	Returns the current value for the duration in decimal hour.
public int	<i>getValue()</i> on page 1076	Returns the current value for the duration in seconds.

Inherited members from `FieldModel`

Modifier and Type	Member	Description
public <code>ActionResult</code>	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public <code>ActionResult</code>	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public <code>ActionEnableType</code>	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public <code>String</code>	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public <code>String</code>	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.

Modifier and Type	Member	Description
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getDurationDisplayFormat() method

Returns the display format specified for the duration.

Syntax

```
public DurationDisplayFormat getDurationDisplayFormat ()
```

Returns

the display format

getFractionalHourValue() method

Returns the current value for the duration in decimal hour.

Syntax

```
public double getFractionalHourValue ()
```

Returns

value as decimal hour

Usage

This should be used when the `getDurationDisplayFormat()` returns `DecHour`.

getValue() method

Returns the current value for the duration in seconds.

Syntax

```
public int getValue ()
```

Returns

value in seconds

DurationEditModel interface

Interface given to a duration edit extension object so it can call back into the host.

Syntax

```
public interface DurationEditModel extends
DurationDisplayModel
```

Members

All members of DurationEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public double	<i>getMaximumFractionalHour()</i> on page 1078	Returns the maximum value allowed.
public int	<i>getMaximumValue()</i> on page 1079	Returns the maximum value allowed.
public double	<i>getMinimumFractionalHour()</i> on page 1079	Returns the minimum value that is enforced.
public int	<i>getMinimumValue()</i> on page 1079	Returns the minimum value that is enforced.
public ProcessInputReturn	<i>processDecimalInput(double)</i> on page 1080	Processes the double input.
public ProcessInputReturn	<i>processInput(int)</i> on page 1080	Processes the integer input.

Inherited members from DurationDisplayModel

Modifier and Type	Member	Description
public DurationDisplayFormat	<i>getDurationDisplayFormat()</i> on page 1076	Returns the display format specified for the duration.
public double	<i>getFractionalHourValue()</i> on page 1076	Returns the current value for the duration in decimal hour.
public int	<i>getValue()</i> on page 1076	Returns the current value for the duration in seconds.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getMaximumFractionalHour() method

Returns the maximum value allowed.

Syntax

```
public double getMaximumFractionalHour ()
```

Returns

minimum value in hours

Usage

This is a double value representing fractional hour. This should be used when `DurationDisplayModel.getDurationDisplayFormat()` returns `DecHour`.

getMaximumValue() method

Returns the maximum value allowed.

Syntax

```
public int getMaximumValue ()
```

Returns

minimum value in seconds

Usage

This is an integer value in seconds.

getMinimumFractionalHour() method

Returns the minimum value that is enforced.

Syntax

```
public double getMinimumFractionalHour ()
```

Returns

minimum value in hours

Usage

This is a double value representing fractional hour. This should be used when `DurationDisplayModel.getDurationDisplayFormat()` returns `DecHour`.

getMinimumValue() method

Returns the minimum value that is enforced.

Syntax

```
public int getMinimumValue ()
```

Returns

minimum value in seconds

Usage

This is an integer value in seconds.

processDecimalInput(double) method

Processes the double input.

Syntax

```
public ProcessInputReturn processDecimalInput ( double value )
```

Parameters

- **value** – the value to process

Returns

result based on the value passed in

Usage

Input is given as fractional hour. Returns a ProcessInputReturn representing the result of processing the input. This should only be used when the DurationDisplayModel.getDurationDisplayFormat() returns DecHour.

processInput(int) method

Processes the integer input.

Syntax

```
public ProcessInputReturn processInput ( int value )
```

Parameters

- **value** – the value to process

Returns

result based on the value passed in

Usage

Input is given in seconds. Returns a ProcessInputReturn representing the result of processing the input.

EmbeddedImageDisplayModel interface

Interface given to a embedded image display extension object so it can call back into the host.

Syntax

```
public interface EmbeddedImageDisplayModel extends FieldModel
```

Members

All members of EmbeddedImageDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public long	<i>getColumnCount()</i> on page 1082	Retrieves from the specified number of columns in the clickable image grid.
public MaskColor	<i>getHighlightColor()</i> on page 1082	Retrieves the highlight selected color to use for showing an image cell as selected.
public OpenUIImage	<i>getImage()</i> on page 1083	Retrieves the OpenUIImage.
public ImagePosition	<i>getImagePosition()</i> on page 1083	Retrieves the image position.
public ImagePresentation	<i>getImagePresentation()</i> on page 1083	Retrieves the image presentation (scaling mode).
public long	<i>getRowCount()</i> on page 1084	Retrieves from the specified number of rows in the clickable image grid.
public boolean	<i>isImageCellSelected(long, long)</i> on page 1084	Retrieves if the specified cell is selected.
public void	<i>setImageCellSelected(long, long)</i> on page 1084	Called to inform Agentry that an image cell has been clicked.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.

Modifier and Type	Member	Description
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

Usage

Depending on the editor settings, the image can be split into multiple cells. Cells are identified by x- and y-coordinates, with the origin (0, 0) being at the top-left.

getColumnCount() method

Retrieves from the specified number of columns in the clickable image grid.

Syntax

```
public long getColumnCount ()
```

Returns

number of columns

getHighlightColor() method

Retrieves the highlight selected color to use for showing an image cell as selected.

Syntax

```
public MaskColor getHighlightColor ()
```

Returns

highlight color to use

Usage

Color is valid if the return object's `isValid()` method returns true.

getImage() method

Retrieves the `OpenUIImage`.

Syntax

```
public OpenUIImage getImage ()
```

Returns

the image to display

getImagePosition() method

Retrieves the image position.

Syntax

```
public ImagePosition getImagePosition ()
```

Returns

the image position

Usage

This will return null if there is no image.

getImagePresentation() method

Retrieves the image presentation (scaling mode).

Syntax

```
public ImagePresentation getImagePresentation ()
```

Returns

the image presentation (scaling mode)

Usage

This will return null if there is no image.

getRowCount() method

Retrieves from the specified number of rows in the clickable image grid.

Syntax

```
public long getRowCount ()
```

Returns

number of rows

isImageCellSelected(long, long) method

Retrieves if the specified cell is selected.

Syntax

```
public boolean isImageCellSelected ( long x , long y )
```

Parameters

- **x** – horizontal cell. In other words, the column.
- **y** – vertical cell. In other words, the row.

Returns

true if the specified cell is selected, false if the specified cell is not selected

setImageCellSelected(long, long) method

Called to inform Agentry that an image cell has been clicked.

Syntax

```
public void setImageCellSelected ( long x , long y )
```

Parameters

- **x** – horizontal cell clicked
- **y** – vertical cell clicked

ExternalDataDisplayModel interface

Interface given to a external data display extension object so it can call back into the host.

Syntax

```
public interface ExternalDataDisplayModel extends FieldModel
```

Derived classes

- *ExternalDataEditModel* on page 1086

Members

All members of ExternalDataDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public String	<i>getFilePath()</i> on page 1086	Returns the path to the external data file.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.

Modifier and Type	Member	Description
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getFilePath() method

Returns the path to the external data file.

Syntax

```
public String getFilePath ()
```

Returns

the file path

ExternalDataEditModel interface

Interface given to a external data edit extension object so it can call back into the host.

Syntax

```
public interface ExternalDataEditModel extends  
ExternalDataDisplayModel
```

Members

All members of ExternalDataEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public ProcessInputReturn	<i>processInput(String)</i> on page 1087	Processes the entered file path.

Inherited members from ExternalDataDisplayModel

Modifier and Type	Member	Description
public String	<i>getFilePath()</i> on page 1086	Returns the path to the external data file.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.

Modifier and Type	Member	Description
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

processInput(String) method

Processes the entered file path.

Syntax

```
public ProcessInputReturn processInput ( String filePath )
```

Parameters

- **filePath** – path to the file

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

FieldModel interface

Interface given to an extension object so it can call back into the host.

Syntax

```
public interface FieldModel
```

Derived classes

- *BooleanDisplayModel* on page 1058
- *ButtonDisplayModel* on page 1061
- *DateAndTimeDisplayModel* on page 1064
- *DateDisplayModel* on page 1067
- *DecimalDisplayModel* on page 1071
- *DurationDisplayModel* on page 1074
- *EmbeddedImageDisplayModel* on page 1080
- *ExternalDataDisplayModel* on page 1084
- *IntegerDisplayModel* on page 1093
- *LabelDisplayModel* on page 1097
- *LocationDisplayModel* on page 1098
- *StringDisplayModel* on page 1101
- *TimeDisplayModel* on page 1107

Members

All members of `FieldModel`, including inherited members. **Methods**

Modifier and Type	Method	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.

Modifier and Type	Method	Description
public String	<i>getAgencyString(String)</i> on page 1090	Asks Agency for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

executeAgencyAction(String) method

Asks Agency to execute the action specified by name.

Syntax

```
public ActionResult executeAgencyAction (String actionName)
```

Parameters

- **actionName** – the action name as a string

Returns

the result of trying to run the action

Usage

This should only be called if `getAgentryActionEnableState` returns `ActionEnable` for the specified action. Only actions defined for this control in the Agentry Editor can be executed.

executeHyperlinkAction() method

Asks Agentry to run the field's hyperlink action.

Syntax

```
public ActionResult executeHyperlinkAction ()
```

Returns

the action result

getAgentryActionEnableState(String) method

Asks Agentry what the current enable state is for the action specified by name.

Syntax

```
public ActionEnableType getAgentryActionEnableState ( String  
actionName )
```

Parameters

- **actionName** – the action name

Returns

the enable state

Usage

It will either be enabled, disabled, no-op(action not found), or error.

getAgentryString(String) method

Asks Agentry for a specific string value.

Syntax

```
public String getAgentryString ( String name )
```

Parameters

- **name** – the string the extension is requesting.

Returns

the value paired with that string.

Usage

In the definitions there are key/value pairs. The String passed in is a key, the value is returned. If no key exists for the specified string, null will be returned.

getLabel() method

Returns the label text for the field.

Syntax

```
public String getLabel ()
```

Returns

the label text

isAutoSizeSupported() method

Checks whether the the field is allowed to automatically decide its own height.

Syntax

```
public boolean isAutoSizeSupported ()
```

Returns

true if the field can set its height, false if it cannot.

Usage

This directly corresponds to the editor setting for the height of the extended field. If it is set to "auto", this will return true. If it's set to a number of rows for height, then this will return false.

This works in conjunction with `FieldAdapter.autosizeBehavior()`. This method allows the extension to ask if the editor definitions support autosizing.

`FieldAdapter.autosizeBehavior()` is Agentry's way of asking the extension how to handle autosizing.

isEnabled() method

Returns whether the field is currently enabled based on current rule evaluation.

Syntax

```
public boolean isEnabled ()
```

Returns

true if the field is enabled, false if it is disabled

isHidden() method

Returns whether or not the field is currently hidden based on current rule evaluations.

Syntax

```
public boolean isHidden ()
```

Returns

true if the field is hidden, false if the field is visible

isHyperlinkEnabled() method

Returns whether or not the label hyperlink action is enabled.

Syntax

```
public boolean isHyperlinkEnabled ()
```

Returns

true if the label hyperlink action is enabled, false if it is disabled.

launchActivity(Intent, int) method

If the extension needs to launch an new activity, it has to call through this method to do it.

Syntax

```
public void launchActivity ( Intent intent , int requestCode )
```

Parameters

- **intent** – defines the activity to launch
- **requestCode** – what code to return in `onActivityResult()`.

Usage

It needs to pass in the intent and requestCode. Agentry will handle launching the activity. `FieldAdapter.onActivityResult` will be called when the activity is dismissed.

requestLayoutHeight(int) method

This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

Syntax

```
public void requestLayoutHeight ( int newHeight )
```

Parameters

- **newHeight** – the new height requested by the extension.

Usage

This is used by auto-sizing fields to tell the layout manager what their actual height is. This should only be called if `isAutoSizeSupported()` returns true.

IntegerDisplayModel interface

Interface given to an integer display extension object so it can call back into the host.

Syntax

```
public interface IntegerDisplayModel extends FieldModel
```

Derived classes

- *IntegerEditModel* on page 1094

Members

All members of `IntegerDisplayModel`, including inherited members. **Methods**

Modifier and Type	Method	Description
public int	<i>getValue()</i> on page 1094	Gets the current integer value from the model.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutoSizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.

Modifier and Type	Member	Description
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch a new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Gets the current integer value from the model.

Syntax

```
public int getValue ()
```

Returns

the current value of the field

IntegerEditModel interface

Interface given to an integer edit extension object so it can call back into the host.

Syntax

```
public interface IntegerEditModel extends IntegerDisplayModel
```

Members

All members of IntegerEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public int	<i>getMaximumValue()</i> on page 1096	Returns the maximum integer value that will be accepted.

Modifier and Type	Method	Description
public int	<i>getMinimumValue()</i> on page 1096	Returns the minimum integer value that will be accepted.
public ProcessInputReturn	<i>processIntegerInput(int)</i> on page 1096	Processes the input of the field.

Inherited members from IntegerDisplayModel

Modifier and Type	Member	Description
public int	<i>getValue()</i> on page 1094	Gets the current integer value from the model.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgencyAction(String)</i> on page 1089	Asks Agency to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agency to run the field's hyperlink action.
public ActionEnableType	<i>getAgencyActionEnableState(String)</i> on page 1090	Asks Agency what the current enable state is for the action specified by name.
public String	<i>getAgencyString(String)</i> on page 1090	Asks Agency for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.

Modifier and Type	Member	Description
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getMaximumValue() method

Returns the maximum integer value that will be accepted.

Syntax

```
public int getMaximumValue ()
```

Returns

the maximum integer value that will be accepted

getMinimumValue() method

Returns the minimum integer value that will be accepted.

Syntax

```
public int getMinimumValue ()
```

Returns

the minimum integer value that will be accepted

processIntegerInput(int) method

Processes the input of the field.

Syntax

```
public ProcessInputReturn processIntegerInput ( int value )
```

Parameters

- **value** – the value to process

Returns

result based on the value passed in

Usage

Returns a ProcessInputReturn representing the result of processing the input.

LabelDisplayModel interface

Interface given to a label display extension object so it can call back into the host.

Syntax

```
public interface LabelDisplayModel extends FieldModel
```

Members

All members of LabelDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public String	<i>getValue()</i> on page 1098	Returns the text the label should display.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.

Modifier and Type	Member	Description
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the text the label should display.

Syntax

```
public String getValue ()
```

Returns

the label text

LocationDisplayModel interface

Interface given to a location display extension object so it can call back into the host.

Syntax

```
public interface LocationDisplayModel extends FieldModel
```

Derived classes

- *LocationEditModel* on page 1100

Members

All members of LocationDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public AgentryLocation	<i>getValue()</i> on page 1099	Returns the location that should be displayed.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.

Modifier and Type	Member	Description
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the location that should be displayed.

Syntax

```
public AgentryLocation getValue ()
```

Returns

the location

LocationEditModel interface

Interface given to a location edit extension object so it can call back into the host.

Syntax

```
public interface LocationEditModel extends
LocationDisplayModel
```

Members

All members of LocationEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public ProcessInputReturn	<i>processInput(AgentryLocation)</i> on page 1101	Processes the selected location.

Inherited members from LocationDisplayModel

Modifier and Type	Member	Description
public AgentryLocation	<i>getValue()</i> on page 1099	Returns the location that should be displayed.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.

Modifier and Type	Member	Description
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

processInput(AgentryLocation) method

Processes the selected location.

Syntax

```
public ProcessInputReturn processInput ( AgentryLocation
location )
```

Parameters

- **location** – object

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

StringDisplayModel interface

Interface given to a string display extension object so it can call back into the host.

Syntax

```
public interface StringDisplayModel extends FieldModel
```

Derived classes

- *StringEditModel* on page 1104

Members

All members of *StringDisplayModel*, including inherited members. **Methods**

Modifier and Type	Method	Description
public String	<i>getValue()</i> on page 1103	Returns the current value of the field.
public boolean	<i>isCarriageReturnAllowed()</i> on page 1103	Returns whether the field allows carriage returns.
public boolean	<i>isWordWrapAllowed()</i> on page 1103	Returns whether the field allows word wrap.

Inherited members from *FieldModel*

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutoSizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.

Modifier and Type	Member	Description
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the current value of the field.

Syntax

```
public String getValue ()
```

Returns

the field value

isCarriageReturnAllowed() method

Returns whether the field allows carriage returns.

Syntax

```
public boolean isCarriageReturnAllowed ()
```

Returns

whether or not carriage return is allowed

isWordWrapAllowed() method

Returns whether the field allows word wrap.

Syntax

```
public boolean isWordWrapAllowed ()
```

Returns

whether or not word wrap is allowed

StringEditModel interface

Interface given to a string edit extension object so it can call back into the host.

Syntax

```
public interface StringEditModel extends StringDisplayModel
```

Members

All members of StringEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public int	<i>getMaximumLength()</i> on page 1105	This retrieves the maximum number of characters that the edit text field will allow upon field validation.
public int	<i>getMinimumLength()</i> on page 1106	This retrieves the minimum number of characters that the edit text field will allow upon field validation.
public boolean	<i>isPasswordInput()</i> on page 1106	This is used to determine if the edit field should obscure its input, as would be the case if it were being used to retrieve a password.
public ProcessInputReturn	<i>processInput(String)</i> on page 1106	Processes the input of the field.

Inherited members from StringDisplayModel

Modifier and Type	Member	Description
public String	<i>getValue()</i> on page 1103	Returns the current value of the field.
public boolean	<i>isCarriageReturnAllowed()</i> on page 1103	Returns whether the field allows carriage returns.
public boolean	<i>isWordWrapAllowed()</i> on page 1103	Returns whether the field allows word wrap.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getMaximumLength() method

This retrieves the maximum number of characters that the edit text field will allow upon field validation.

Syntax

```
public int getMaximumLength ()
```

Returns

the maximum number of characters to enter

getMinimumLength() method

This retrieves the minimum number of characters that the edit text field will allow upon field validation.

Syntax

```
public int getMinimumLength ()
```

Returns

the minimum number of characters to enter

isPasswordInput() method

This is used to determine if the edit field should obscure its input, as would be the case if it were being used to retrieve a password.

Syntax

```
public boolean isPasswordInput ()
```

Returns

true if the input should be hidden from the user, in whatever password-entry style is standard for the platform; false if not

processInput(String) method

Processes the input of the field.

Syntax

```
public ProcessInputReturn processInput (String value )
```

Parameters

- **value** – the value to process

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

TimeDisplayModel interface

Interface given to a time display extension object so it can call back into the host.

Syntax

```
public interface TimeDisplayModel extends FieldModel
```

Derived classes

- *TimeEditModel* on page 1108

Members

All members of TimeDisplayModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public GregorianCalendar	<i>getValue()</i> on page 1108	Returns the field's current time value.

Inherited members from FieldModel

Modifier and Type	Member	Description
public ActionResult	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public ActionResult	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public ActionEnableType	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public String	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public String	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public boolean	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public boolean	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.

Modifier and Type	Member	Description
public boolean	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public boolean	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public void	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public void	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

getValue() method

Returns the field's current time value.

Syntax

```
public GregorianCalendar getValue ()
```

Returns

the time

TimeEditModel interface

Interface given to a time edit extension object so it can call back into the host.

Syntax

```
public interface TimeEditModel extends TimeDisplayModel
```

Members

All members of TimeEditModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public ProcessInputReturn	<i>processInput(GregorianCalendar)</i> on page 1110	Processes the current time input.

Inherited members from TimeDisplayModel

Modifier and Type	Member	Description
public <code>GregorianCalendar</code>	<i>getValue()</i> on page 1108	Returns the field's current time value.

Inherited members from `FieldModel`

Modifier and Type	Member	Description
public <code>ActionResult</code>	<i>executeAgentryAction(String)</i> on page 1089	Asks Agentry to execute the action specified by name.
public <code>ActionResult</code>	<i>executeHyperlinkAction()</i> on page 1090	Asks Agentry to run the field's hyperlink action.
public <code>ActionEnableType</code>	<i>getAgentryActionEnableState(String)</i> on page 1090	Asks Agentry what the current enable state is for the action specified by name.
public <code>String</code>	<i>getAgentryString(String)</i> on page 1090	Asks Agentry for a specific string value.
public <code>String</code>	<i>getLabel()</i> on page 1091	Returns the label text for the field.
public <code>boolean</code>	<i>isAutosizeSupported()</i> on page 1091	Checks whether the the field is allowed to automatically decide its own height.
public <code>boolean</code>	<i>isEnabled()</i> on page 1091	Returns whether the field is currently enabled based on current rule evaluation.
public <code>boolean</code>	<i>isHidden()</i> on page 1092	Returns whether or not the field is currently hidden based on current rule evaluations.
public <code>boolean</code>	<i>isHyperlinkEnabled()</i> on page 1092	Returns whether or not the label hyperlink action is enabled.
public <code>void</code>	<i>launchActivity(Intent, int)</i> on page 1092	If the extension needs to launch an new activity, it has to call through this method to do it.
public <code>void</code>	<i>requestLayoutHeight(int)</i> on page 1092	This is called by a field's UI extension to tell the model's layout manager that the field needs to have a specific pixel height.

processInput(GregorianCalendar) method

Processes the current time input.

Syntax

```
public ProcessInputReturn processInput ( GregorianCalendar
time )
```

Parameters

- **time** – the time value to process

Returns

result based on the value passed in

Usage

Returns a `ProcessInputReturn` representing the result of processing the input.

core package

openui package

AgentryImage class

This class is the Java implementation to support Agentry images.

Syntax

```
public class AgentryImage extends OpenUIImage
```

Members

All members of `AgentryImage`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>AgentryImage(String, Image-Type, ImagePresentation, ImagePosition, int, int, int)</i> on page 1113	Constructs an <code>AgentryImage</code> object.

Methods

Modifier and Type	Method	Description
public byte[]	<i>getBitmapData()</i> on page 1113	Returns the bitmap data for the image.

Modifier and Type	Method	Description
public String	<i>getImageName()</i> on page 1113	Retrieves the imageName.
public ImagePosition	<i>getImagePosition()</i> on page 1114	Retrieves the image position.
public ImagePresentation	<i>getImagePresentation()</i> on page 1114	Retrieves the image presentation and scaling mode.
public ImageType	<i>getImageType()</i> on page 1114	Retrieves the image type.
public MaskColor	<i>getMaskColor()</i> on page 1114	Retrieves the image's transparency color.
public boolean	<i>isValid()</i> on page 1114	Returns whether the image represented by this object is valid.
public boolean	<i>needsBitmapData()</i> on page 1115	Returns true if the bitmap data has been cached.
public void	<i>setBitmapData(byte[])</i> on page 1115	Sets the bitmap data and caches it for next time.

AgentryImage.ImageType enum

The ImageType enum represents the different image types that Agentry stores.

Members

All members of ImageType, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>ImageType_Bitmap</i> on page 1112	.bmp file
public	<i>ImageType_GIF</i> on page 1112	.gif file
public	<i>ImageType_JPEG</i> on page 1112	.jpg or .jpeg file
public	<i>ImageType_PNG</i> on page 1112	.png file
public	<i>ImageType_Unknown</i> on page 1112	We don't know the image type.

Methods

Modifier and Type	Method	Description
public int	<i>getValue()</i> on page 1112	

getValue() method

Syntax

```
public int getValue ()
```

ImageType_Bitmap variable

.bmp file

Syntax

```
public ImageType_Bitmap
```

ImageType_GIF variable

.gif file

Syntax

```
public ImageType_GIF
```

ImageType_JPEG variable

.jpg or .jpeg file

Syntax

```
public ImageType_JPEG
```

ImageType_PNG variable

.png file

Syntax

```
public ImageType_PNG
```

ImageType_Unknown variable

We don't know the image type.

Syntax

```
public ImageType_Unknown
```


AgentryImage(String, ImageType, ImagePresentation, ImagePosition, int, int, int)
constructor

Constructs an AgentryImage object.

Syntax

```
public AgentryImage ( String imageName , ImageType type ,
ImagePresentation presentation , ImagePosition position , int maskRed ,
int maskGreen , int maskBlue )
```

Parameters

- **imageName** – The name of the image
- **type** – The image type.
- **presentation** – The image presentation.
- **position** – The image position.
- **maskRed** – The red component of the masking color, or -1 if there is no masking color.
- **maskGreen** – The green component of the masking color, or -1 if there is no masking color.
- **maskBlue** – The blue component of the masking color, or -1 if there is no masking color.

Usage

This does not set the actual bitmap data and should be followed by a call to needsBitmapData and setBitmapData (if appropriate).

getBitmapData() method

Returns the bitmap data for the image.

Syntax

```
public byte[] getBitmapData ()
```

Returns

the bitmap data

getImageName() method

Retrieves the imageName.

Syntax

```
public String getImageName ()
```

Returns

the image name

getImagePosition() method

Retrieves the image position.

Syntax

```
public ImagePosition getImagePosition ()
```

Returns

The image position.

getImagePresentation() method

Retrieves the image presentation and scaling mode.

Syntax

```
public ImagePresentation getImagePresentation ()
```

Returns

The image presentation (scaling mode).

getImageType() method

Retrieves the image type.

Syntax

```
public ImageType getImageType ()
```

Returns

The image type

getMaskColor() method

Retrieves the image's transparency color.

Syntax

```
public MaskColor getMaskColor ()
```

Returns

the image's transparency masking color, or null if there is no mask color. This only applies to BMP-format images.

isValid() method

Returns whether the image represented by this object is valid.

Syntax

```
public boolean isValid ()
```

Returns

true if the image is valid, else false

needsBitmapData() method

Returns true if the bitmap data has been cached.

Syntax

```
public boolean needsBitmapData ()
```

Returns

true if cached data was found, false otherwise

Usage

If so, there is no need to call `setBitmapData`. If cached data was not found, `setBitmapData` should be called.

setBitmapData(byte[]) method

Sets the bitmap data and caches it for next time.

Syntax

```
public void setBitmapData (byte[] bitmap )
```

Parameters

- **bitmap** – byte array of bitmap data

AgentryLocation class

Gives the location details.

Syntax

```
public class AgentryLocation
```

Members

All members of `AgentryLocation`, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>AgentryLocation(boolean, double, double, int, double)</i> on page 1116	Constructs a new <code>AgentryLocation</code> object.

Methods

Modifier and Type	Method	Description
public double	<i>getDilution()</i> on page 1116	Retrieves the dilution.
public double	<i>getLatitude()</i> on page 1117	Retrieves the latitude.
public double	<i>getLongitude()</i> on page 1117	Retrieves the longitude.
public int	<i>getSatellites()</i> on page 1117	Retrieves the number of satellites.
public boolean	<i>isValid()</i> on page 1117	Returns whether the location is valid.
public void	<i>setDilution(double)</i> on page 1118	Sets the dilution.
public void	<i>setLatitude(double)</i> on page 1118	Sets the latitude.
public void	<i>setLongitude(double)</i> on page 1118	Sets the longitude.
public void	<i>setSatellites(int)</i> on page 1118	Sets the number of satellites.
public void	<i>setValid(boolean)</i> on page 1118	Sets whether the location is valid.

AgentryLocation(boolean, double, double, int, double) constructor

Constructs a new AgentryLocation object.

Syntax

```
public AgentryLocation ( boolean valid , double lat , double lon ,
int sats , double decDilution )
```

Parameters

- **valid** – true if the location is valid, false otherwise
- **lat** – latitude
- **lon** – longitude
- **sats** – number of satellites
- **decDilution** – dilution

getDilution() method

Retrieves the dilution.

Syntax

```
public double getDilution ()
```

Returns

the dilution

getLatitude() method

Retrieves the latitude.

Syntax

```
public double getLatitude ()
```

Returns

the latitude

getLongitude() method

Retrieves the longitude.

Syntax

```
public double getLongitude ()
```

Returns

the longitude

getSatellites() method

Retrieves the number of satellites.

Syntax

```
public int getSatellites ()
```

Returns

the number of satellites

isValid() method

Returns whether the location is valid.

Syntax

```
public boolean isValid ()
```

Returns

true if the location is valid, false otherwise

setDilution(double) method

Sets the dilution.

Syntax

```
public void setDilution ( double dilution )
```

Parameters

- **dilution** – the new dilution

setLatitude(double) method

Sets the latitude.

Syntax

```
public void setLatitude ( double latitude )
```

Parameters

- **latitude** – the new latitude

setLongitude(double) method

Sets the longitude.

Syntax

```
public void setLongitude ( double longitude )
```

Parameters

- **longitude** – the new longitude

setSatellites(int) method

Sets the number of satellites.

Syntax

```
public void setSatellites ( int satellites )
```

Parameters

- **satellites** – the new number of satellites

setValid(boolean) method

Sets whether the location is valid.

Syntax

```
public void setValid ( boolean isValid )
```

Parameters

- **isValid** – true if the location is valid, false otherwise

MaskColor class

This encapsulates a masking color that's used by AgentryImage.

Syntax

```
public class MaskColor
```

Members

All members of MaskColor, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>MaskColor(short, short, short)</i> on page 1119	Constructs a new MaskColor object.
public	<i>MaskColor(int, int, int)</i> on page 1120	Constructs a new MaskColor object.

Methods

Modifier and Type	Method	Description
public short	<i>getBlue()</i> on page 1120	Returns the blue component of the color.
public short	<i>getGreen()</i> on page 1120	Returns the green component of the color.
public short	<i>getRed()</i> on page 1120	Returns the red component of the color.
public boolean	<i>isValid()</i> on page 1121	Evaluates the mask color and returns if it is valid.

MaskColor(short, short, short) constructor

Constructs a new MaskColor object.

Syntax

```
public MaskColor ( short red , short green , short blue )
```

Parameters

- **red** – Red component value, 0-255.
- **green** – Green component value, 0-255.

- **blue** – Blue component value, 0-255.

MaskColor(int, int, int) constructor

Constructs a new MaskColor object.

Syntax

```
public    MaskColor ( int red ,    int green ,    int blue )
```

Parameters

- **red** – Red component value, 0-255.
- **green** – Green component value, 0-255.
- **blue** – Blue component value, 0-255.

getBlue() method

Returns the blue component of the color.

Syntax

```
public short  getBlue ()
```

Returns

The blue component of the color, 0-255.

getGreen() method

Returns the green component of the color.

Syntax

```
public short  getGreen ()
```

Returns

The green component of the color, 0-255.

getRed() method

Returns the red component of the color.

Syntax

```
public short  getRed ()
```

Returns

The red component of the color, 0-255.

isValid() method

Evaluates the mask color and returns if it is valid.

Syntax

```
public boolean isValid ()
```

Returns

Whether or not the mask color is valid.

ProcessInputReturn class

Contains the result of calling to process input.

Syntax

```
public class ProcessInputReturn
```

Members

All members of ProcessInputReturn, including inherited members. **Constructors**

Modifier and Type	Constructor	Description
public	<i>ProcessInputReturn(boolean, boolean, boolean)</i> on page 1122	Constructs a new ProcessInputReturn object.

Methods

Modifier and Type	Method	Description
public boolean	<i>getChanged()</i> on page 1122	Returns whether or not the processInput method received a value different than what it already had stored.
public boolean	<i>getMunged()</i> on page 1122	Returns whether or not the processInput method "munged" the value.
public boolean	<i>getValid()</i> on page 1122	Returns whether or not the processInput method accepted the value as valid.

ProcessInputReturn(boolean, boolean, boolean) constructor

Constructs a new ProcessInputReturn object.

Syntax

```
public ProcessInputReturn ( boolean valid , boolean munged ,  
boolean changed )
```

Parameters

- **valid** – is the result valid?
- **munged** – is the result a munged value?
- **changed** – is the result a changed value?

getChanged() method

Returns whether or not the processInput method received a value different than what it already had stored.

Syntax

```
public boolean getChanged ()
```

Returns

true if changed, false if not changed

getMunged() method

Returns whether or not the processInput method "munged" the value.

Syntax

```
public boolean getMunged ()
```

Returns

true if munged, false if not munged

Usage

Munged means that the value needed to be changed, but the logical value was not affected. For example, if the lowercase attribute is set, and an uppercase character was typed in, the stored value gets changed to all lowercase and the UI needs to be updated.

getValid() method

Returns whether or not the processInput method accepted the value as valid.

Syntax

```
public boolean getValid ()
```

Returns

true if valid, false if invalid

ActionEnableType enum

The enable states that an action can have.

Members

All members of ActionEnableType, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>ActionDisable</i> on page 1123	Action is disabled.
public	<i>ActionEnable</i> on page 1123	Action is enabled.
public	<i>ActionError</i> on page 1123	Action is found but is invalid.
public	<i>ActionNoOperation</i> on page 1123	Action cannot be found.

ActionDisable variable

Action is disabled.

Syntax

```
public ActionDisable
```

ActionEnable variable

Action is enabled.

Syntax

```
public ActionEnable
```

ActionError variable

Action is found but is invalid.

Syntax

```
public ActionError
```

ActionNoOperation variable

Action cannot be found.

Syntax

```
public ActionNoOperation
```

ActionResult enum

The result states that running an action can return.

Members

All members of ActionResult, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>Action_BackUp</i> on page 1124	The action was backed out of by the user.
public	<i>Action_Cancel</i> on page 1124	The action was canceled by user.
public	<i>Action_Complete</i> on page 1124	The action completed successfully.
public	<i>Action_Error</i> on page 1125	There was an error when running the action.
public	<i>Action_Pending</i> on page 1125	The action is still in progress and has not yet completed.

Action_BackUp variable

The action was backed out of by the user.

Syntax

```
public Action_BackUp
```

Action_Cancel variable

The action was canceled by user.

Syntax

```
public Action_Cancel
```

Action_Complete variable

The action completed successfully.

Syntax

```
public Action_Complete
```

Action_Error variable

There was an error when running the action.

Syntax

```
public Action_Error
```

Action_Pending variable

The action is still in progress and has not yet completed.

Syntax

```
public Action_Pending
```

AutosizeBehavior enum

Values for autosize behavior for Agentry fields.

Members

All members of AutosizeBehavior, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>Autosize_FillVisible</i> on page 1125	Field should take up the remaining visible area on the screen.
public	<i>Autosize_None</i> on page 1125	Field does not autosize.
public	<i>Autosize_WrapContent</i> on page 1126	Field should size itself so all of its content is visible The layout manager will invoke the FieldAdapter.getContentHeightForAutosizing(int width) method to find the field's content size.

Autosize_FillVisible variable

Field should take up the remaining visible area on the screen.

Syntax

```
public Autosize_FillVisible
```

Autosize_None variable

Field does not autosize.

Syntax

```
public Autosize_None
```

Autosize_WrapContent variable

Field should size itself so all of its content is visible The layout manager will invoke the FieldAdapter.getContentHeightForAutosizing(int width) method to find the field's content size.

Syntax

```
public Autosize_WrapContent
```

ButtonType enum

This enum has the 3 different types of buttons an Agentry Button Widget can be set to.

Members

All members of ButtonType, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>ButtonStyleCheckbox</i> on page 1126	Check box style button.
public	<i>ButtonStylePush</i> on page 1126	Push Button.
public	<i>ButtonStyleRadio</i> on page 1126	Radio Button.

ButtonStyleCheckbox variable

Check box style button.

Syntax

```
public ButtonStyleCheckbox
```

ButtonStylePush variable

Push Button.

Syntax

```
public ButtonStylePush
```

ButtonStyleRadio variable

Radio Button.

Syntax

```
public ButtonStyleRadio
```

DurationDisplayFormat enum

This is a list of possible duration display formats.

Members

All members of DurationDisplayFormat, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>DecHour</i> on page 1127	HH.XX.
public	<i>HourMin</i> on page 1127	HH:MM.
public	<i>HourMinSec</i> on page 1127	HH:MM:SS where : will be localized.
public	<i>MinSec</i> on page 1127	MM:SS.

DecHour variable

HH.XX.

Syntax

```
public    DecHour
```

HourMin variable

HH:MM.

Syntax

```
public    HourMin
```

HourMinSec variable

HH:MM:SS where : will be localized.

Syntax

```
public    HourMinSec
```

MinSec variable

MM:SS.

Syntax

```
public    MinSec
```

ImagePosition enum

The ImagePosition enum represents the different ways that an image can be positioned in the available space.

Members

All members of ImagePosition, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>ImagePosition_Center</i> on page 1129	Image positioned at the center.
public	<i>ImagePosition_LowerLeft</i> on page 1129	Image positioned at the bottom left.
public	<i>ImagePosition_LowerMiddle</i> on page 1129	Image positioned at the bottom middle.
public	<i>ImagePosition_LowerRight</i> on page 1129	Image positioned at the bottom right.
public	<i>ImagePosition_MiddleLeft</i> on page 1129	Image positioned at the middle left.
public	<i>ImagePosition_MiddleRight</i> on page 1129	Image positioned at the middle right.
public	<i>ImagePosition_Unknown</i> on page 1129	We don't know the image position.
public	<i>ImagePosition_UpperLeft</i> on page 1130	Image positioned at the top left.
public	<i>ImagePosition_UpperMiddle</i> on page 1130	Image positioned at the top middle.
public	<i>ImagePosition_UpperRight</i> on page 1130	Image positioned at the top right.

Methods

Modifier and Type	Method	Description
public int	<i>getValue()</i> on page 1128	

getValue() method

Syntax

```
public int getValue ()
```


ImagePosition_Center variable

Image positioned at the center.

Syntax

```
public    ImagePosition_Center
```

ImagePosition_LowerLeft variable

Image positioned at the bottom left.

Syntax

```
public    ImagePosition_LowerLeft
```

ImagePosition_LowerMiddle variable

Image positioned at the bottom middle.

Syntax

```
public    ImagePosition_LowerMiddle
```

ImagePosition_LowerRight variable

Image positioned at the bottom right.

Syntax

```
public    ImagePosition_LowerRight
```

ImagePosition_MiddleLeft variable

Image positioned at the middle left.

Syntax

```
public    ImagePosition_MiddleLeft
```

ImagePosition_MiddleRight variable

Image positioned at the middle right.

Syntax

```
public    ImagePosition_MiddleRight
```

ImagePosition_Unknown variable

We don't know the image position.

Syntax

```
public    ImagePosition_Unknown
```

ImagePosition_UpperLeft variable

Image positioned at the top left.

Syntax

```
public    ImagePosition_UpperLeft
```

ImagePosition_UpperMiddle variable

Image positioned at the top middle.

Syntax

```
public    ImagePosition_UpperMiddle
```

ImagePosition_UpperRight variable

Image positioned at the top right.

Syntax

```
public    ImagePosition_UpperRight
```

ImagePresentation enum

The ImagePresentation enum represents the different ways that an image can be displayed.

Members

All members of ImagePresentation, including inherited members. **Variables**

Modifier and Type	Variable	Description
public	<i>ImagePresentation_CropToFit</i> on page 1131	Image will be cropped to fit available area if it is larger than the available area.
public	<i>ImagePresentation_FullSize</i> on page 1131	The image should be presented full-sized.
public	<i>ImagePresentation_LockAspectRatio</i> on page 1131	Lock the aspect ratio.
public	<i>ImagePresentation_StretchToFit</i> on page 1131	Image will be stretched to fit available area.
public	<i>ImagePresentation_Unknown</i> on page 1131	We don't know the image presentation type.

Methods

Modifier and Type	Method	Description
public int	<i>getValue()</i> on page 1131	

getValue() method

Syntax

```
public int getValue ()
```

ImagePresentation_CropToFit variable

Image will be cropped to fit available area if it is larger than the available area.

Syntax

```
public ImagePresentation_CropToFit
```

ImagePresentation_FullSize variable

The image should be presented full-sized.

Syntax

```
public ImagePresentation_FullSize
```

ImagePresentation_LockAspectRatio variable

Lock the aspect ratio.

Syntax

```
public ImagePresentation_LockAspectRatio
```

Usage

Image will be resized to fit in the available area but maintains its aspect ratio.

ImagePresentation_StretchToFit variable

Image will be stretched to fit available area.

Syntax

```
public ImagePresentation_StretchToFit
```

ImagePresentation_Unknown variable

We don't know the image presentation type.

Syntax

```
public ImagePresentation_Unknown
```

OpenUIImage interface

This is the interface used for Open UI images.

Syntax

```
public interface OpenUIImage
```

Derived classes

- *AgentryImage* on page 1110

Members

All members of OpenUIImage, including inherited members. **Methods**

Modifier and Type	Method	Description
public byte[]	<i>getBitmapData()</i> on page 1132	Returns the bitmap data for the image.
public String	<i>getImageName()</i> on page 1132	Retrieves the imageName.
public ImageType	<i>getImageType()</i> on page 1133	Retrieves the image type.
public MaskColor	<i>getMaskColor()</i> on page 1133	Retrieves the image's transparency color.
public boolean	<i>isValid()</i> on page 1133	Returns whether the image represented by this object is valid.

getBitmapData() method

Returns the bitmap data for the image.

Syntax

```
public byte[] getBitmapData ()
```

Returns

the bitmap data

getImageName() method

Retrieves the imageName.

Syntax

```
public String getImageName ()
```

Returns

the image name

getImageType() method

Retrieves the image type.

Syntax

```
public ImageType getImageType ()
```

Returns

The image type

getMaskColor() method

Retrieves the image's transparency color.

Syntax

```
public MaskColor getMaskColor ()
```

Returns

the image's transparency masking color, or null if there is no mask color. This only applies to BMP-format images.

isValid() method

Returns whether the image represented by this object is valid.

Syntax

```
public boolean isValid ()
```

Returns

true if the image is valid, else false

Agentry OpenUI API for iOS

Use the OpenUI API for iOS to add custom controls to Agentry applications.

iOSDataAPI

iOSDataAPIExternal

The iOS DataAPI exposed interfaces.

Usage

This module contains a grouping of all exposed interfaces of DataAPI to provide easy access to all the protocols available via the API.

For a detailed overview of DataAPI, please visit the `smpdataapi_ios` documentation landing page.

SMPDataAPILocationProtocol protocol

SMPDataAPILocationProtocol Protocol - Defines a interface that all SAP Mobile Platform Location objects must adhere to.

Syntax

```
@protocol SMPDataAPILocationProtocol
```

Derived classes

- *SMPOpenUILocation* on page 1143

initWithCLLocation: method

Initializer for the SMPOpenUILocation object from a CLLocation.

Syntax

```
- (id) initWithCLLocation : (CLLocation *) location
```

Parameters

- **location** – the CLLocation.

initWithLatitude:andLongitude:andSatellites:andDilution: method

Initializer for the SMPOpenUILocation object.

Syntax

```
- (id) initWithLatitude : (double) latitude andLongitude : (double) longitude andSatellites : (int) satellites andDilution : (double) dilution
```

Parameters

- **latitude** – the latitude.
- **longitude** – the longitude.
- **satellites** – the number of satellites used.
- **dilution** – the horizontal accuracy of the position.

locationWithCLLocation: method

Get an autoreleased SMPOpenUILocation object from a CLLocation.

Syntax

```
+ (id) locationWithCLLocation : (CLLocation *) location
```

Parameters

- **location** – the CLLocation.

locationWithLatitude:andLongitude:andSatellites:andDilution: method

Get an autoreleased SMPOpenUILocation object.

Syntax

```
+ (id) locationWithLatitude : (double) latitude andLongitude :  
  (double) longitude andSatellites : (int) satellites andDilution : (  
double) dilution
```

Parameters

- **latitude** – the latitude.
- **longitude** – the longitude.
- **satellites** – the number of satellites used.
- **dilution** – the horizontal accuracy of the position.

dilution property

The dilution of the location.

Syntax

```
@property (nonatomic , readonly) double dilution
```

latitude property

The latitude of the location.

Syntax

```
@property (nonatomic , readonly) double latitude
```

location property

This location object as an auto release CLLocation object.

Syntax

```
@property (nonatomic , readonly) CLLocation * location
```

longitude property

The longitude of the location.

Syntax

```
@property (nonatomic , readonly) double longitude
```

satellites property

The number of satellites used in the reading of the location.

Syntax

```
@property (nonatomic, readonly) NSInteger satellites
```

valid property

A Boolean value representing whether the location object is valid.

Syntax

```
@property (nonatomic, readonly) BOOL valid
```

SMPDataAPIPropertyProtocol protocol

SMPDataAPIPropertyProtocol - Defines an interface that all SAP Mobile Platform Data objects of type Property (SMPDataAPIProperty) must adhere to.

Syntax

```
@protocol SMPDataAPIPropertyProtocol
```

asBool method

Evaluates the value of the property object as a bool.

Syntax

```
- (BOOL) asBool
```

Returns

BOOL value

asDate method

Evaluates the value of the property object as an NSDate object.

Syntax

```
- (NSDate *) asDate
```

Returns

NSDate value

asDateAndTime method

Evaluates the value of the property object as an NSDate object.

Syntax

```
- (NSDate *) asDateAndTime
```


Returns

NSDate value

asDecimal method

Evaluates the value of the property object as an double.

Syntax

```
- (double) asDecimal
```

Returns

double value

asLocation method

Evaluates the value of the property object as an SMPDataAPILocationProtocol object.

Syntax

```
- (id< SMPDataAPILocationProtocol >) asLocation
```

Returns

An object conforming to the SMPDataAPILocationProtocol protocol

asLong method

Evaluates the value of the property object as an integer.

Syntax

```
- (NSInteger) asLong
```

Returns

NSInteger value

asString method

Evaluates the value of the property object as a string.

Syntax

```
- (NSString *) asString
```

Returns

NSString* value

asTime method

Evaluates the value of the property object as an NSDate object.

Syntax

```
- (NSDate *) asTime
```

Returns

NSDate value

log method

Optional debug function to help debug the code.

Syntax

```
- (void) log
```

Usage

This function will print data to NSLog().

propertyType method

A value that identifies the property type of a property data object.

Syntax

```
- (enum SMPDataAPIPropertyType) propertyType
```

Returns

The property type of the object

SMPDataAPIProtocol protocol

SMPDataAPI Protocol - Defines a interface that all SAP Mobile Platform Data objects must adhere to.

Syntax

```
@protocol SMPDataAPIProtocol
```

Derived classes

- *SMPDataAPIPropertyProtocol* on page 1136

ancestor method

The ancestor object (parent object)

Syntax

```
- (id< SMPDataAPIProtocol >) ancestor
```

Returns

Ancestor (parent) object

dataIdentifier method

A unique id that identifies the data object.

Syntax

```
- (NSUInteger) dataIdentifier
```

Returns

The unique id value

dataType method

A value that identifies the data type of data object.

Syntax

```
- (enum SMPDataAPIDataType) dataType
```

Returns

The data type of the object

descendant: method

Retrieves a descendant (child) data object.

Syntax

```
- (id< SMPDataAPIProtocol >) descendant : (NSUInteger)
position
```

Parameters

- **position** – Index of data object to retrieve

Returns

Descendant data object

descendantCount method

The number of descendant (child) data objects.

Syntax

```
- (NSInteger) descendantCount
```

Returns

Number of descendant data objects

displayName method

The display name of the Agentry data object.

Syntax

```
- (NSString *) displayName
```

Returns

The display name of the object

log method

Optional debug function to help debug the code.

Syntax

```
- (void) log
```

Usage

This function will print data to NSLog().

name method

The internal name of the Agentry data object.

Syntax

```
- (NSString *) name
```

Returns

The display name

root method

The root object in the data tree for an Agentry Module.

Syntax

```
- (id< SMPDataAPIProtocol >) root
```

Returns

The root object

SMPDataAPIDataType enumeration

enum List of Data Types

Enum Constant Summary

- **SMPDataAPIUnknown** – The model is invalid and the data type can't be determined.
- **SMPDataAPIObject** – An Object.
- **SMPDataAPIProperty** – A Property.
- **SMPDataAPICollection** – A Collection of Objects.

SMPDataAPIPropertyType enumeration

enum List of Property Types

Enum Constant Summary

- **SMPDataAPIUnknownProperty** – The model is invalid and the property type can't be determined.
- **SMPDataAPIIdentifierProperty** – Identifier property.
- **SMPDataAPIStringProperty** – String property.
- **SMPDataAPIIntegerNumber** – Integer property.
- **SMPDataAPIBooleanProperty** – Boolean property.
- **SMPDataAPIDateProperty** – Date property.
- **SMPDataAPITimeProperty** – Time property.
- **SMPDataAPIDurationProperty** – Duration property.
- **SMPDataAPIListSelectionProperty** – List Selection property.
- **SMPDataAPIDataTableSelectionProperty** – Data Table Selection property.
- **SMPDataAPIComplexTableSelectionProperty** – Complex Table Selection property.
- **SMPDataAPISignatureProperty** – Signature property.
- **SMPDataAPIDateAndTime** – Date and time property.
- **SMPDataAPIDecimalNumber** – Decimal number property.
- **SMPDataAPIExternalData** – External data property.
- **SMPDataAPIImage** – Image property.
- **SMPDataAPILocationProperty** – Location property.

iOSOpenUI

iOSOpenUIExternal

The iOS OpenUI exposed interfaces.

Usage

This module contains a grouping of all exposed interfaces of the OpenUI framework to provide easy access to all the protocols available via the API.

For a detailed overview of OpenUI as well as installation instructions, known issues, and other documentation resources, please visit the `smpopenui_ios_overview` documentation landing page.

SMPOpenUIImage class

An immutable object that represents and Agentry image.

Syntax

```
@interface SMPOpenUIImage
```

image property

The actual image created from the image definition.

Syntax

```
@property (nonatomic, readonly) UIImage * image
```

name property

The name of the image defined in the editor.

Syntax

```
@property (nonatomic, readonly) NSString * name
```

Usage

Can be used for accessibility features.

position property

The position of the image.

Syntax

```
@property (nonatomic, readonly) SMPOpenUIImagePosition  
position
```

presentation property

The presentation of the image.

Syntax

```
@property (nonatomic, readonly) SMPOpenUIImagePresentation
presentation
```

SMPOpenUILocation class

An immutable object that represents and Agentry location.

Syntax

```
@interface SMPOpenUILocation :
<SMPLocationProtocol>
```

Usage

It provides utility constructors to ease working with CLLocation objects as well as to get this object as a CLLocation object.

initWithCLLocation: method

Initializer for the SMPOpenUILocation object from a CLLocation.

Syntax

```
- (id) initWithCLLocation : (CLLocation *) location
```

Parameters

- **location** – the CLLocation.

initWithLatitude:andLongitude:andSatellites:andDilution: method

Initializer for the SMPOpenUILocation object.

Syntax

```
- (id) initWithLatitude : (double) latitude andLongitude : (
double) longitude andSatellites : (int) satellites andDilution : (
double) dilution
```

Parameters

- **latitude** – the latitude.
- **longitude** – the longitude.
- **satellites** – the number of satellites used.
- **dilution** – the horizontal accuracy of the position.

locationWithCLLocation: method

Get an autoreleased SMPOpenUILocation object from a CLLocation.

Syntax

```
+ (id) locationWithCLLocation : (CLLocation *) location
```

Parameters

- **location** – the CLLocation.

locationWithLatitude:andLongitude:andSatellites:andDilution: method

Get an autoreleased SMPOpenUILocation object.

Syntax

```
+ (id) locationWithLatitude : (double) latitude andLongitude :  
(double) longitude andSatellites : (int) satellites andDilution : (  
double ) dilution
```

Parameters

- **latitude** – the latitude.
- **longitude** – the longitude.
- **satellites** – the number of satellites used.
- **dilution** – the horizontal accuracy of the position.

dilution property

The dilution of the location.

Syntax

```
@property (nonatomic , readonly) double dilution
```

latitude property

The latitude of the location.

Syntax

```
@property (nonatomic , readonly) double latitude
```

location property

This location object as an auto release CLLocation object.

Syntax

```
@property (nonatomic , readonly) CLLocation * location
```


longitude property

The longitude of the location.

Syntax

```
@property (nonatomic, readonly) double longitude
```

satellites property

The number of satellites used in the reading of the location.

Syntax

```
@property (nonatomic, readonly) NSInteger satellites
```

valid property

A Boolean value representing whether the location object is valid.

Syntax

```
@property (nonatomic, readonly) BOOL valid
```

SMPOpenUIBooleanDisplayAdapter protocol

Protocol for a field extension representing a display-only Boolean field.

Syntax

```
@protocol SMPOpenUIBooleanDisplayAdapter
```

initWithBooleanDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIBooleanDisplayAdapter > )  
initWithBooleanDisplayModel : (id<  
SMPOpenUIBooleanDisplayModel > ) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeBoolean: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIBooleanDisplayModel >) model  
didChangeBoolean : (BOOL) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIBooleanDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only Boolean.

Syntax

```
@protocol SMPOpenUIBooleanDisplayModel
```

Derived classes

- *SMPOpenUIBooleanEditModel* on page 1147

value property

The current Boolean value.

Syntax

```
@property (nonatomic , readonly) BOOL value
```

SMPOpenUIBooleanEditAdapter protocol

Protocol for a field extension representing an editable Boolean field.

Syntax

```
@protocol SMPOpenUIBooleanEditAdapter
```

initWithBooleanEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIBooleanEditAdapter >)  
initWithBooleanEditModel : (id< SMPOpenUIBooleanEditModel >)  
model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeBoolean: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIBooleanEditModel >) model
didChangeBoolean : (BOOL) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIBooleanEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable Boolean.

Syntax

```
@protocol SMPOpenUIBooleanEditModel
```

processInputBoolean: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputBoolean :
(BOOL) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

SMPOpenUIButtonDisplayAdapter protocol

Adapter protocol for an extension field that represents a button.

Syntax

```
@protocol SMPOpenUIButtonDisplayAdapter
```

initWithButtonDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIButtonDisplayAdapter > )  
initWithButtonDisplayModel : (id<  
SMPOpenUIButtonDisplayModel > ) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeButtonImage: method

Called to inform the adapter that the button's image has changed.

Syntax

```
- (void) model : (id< SMPOpenUIButtonDisplayModel > ) model  
didChangeButtonImage : (SMPOpenUIImage *) image
```

Parameters

- **model** – the model for the field.
- **image** – the new image for the button.

model:didChangeSelected: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIButtonDisplayModel > ) model  
didChangeSelected : (BOOL) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIButtonDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a button.

Syntax

```
@protocol SMPOpenUIButtonDisplayModel
```

processInput method

Called to process a button push.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInput
```

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

buttonImage property

The image associated with the button.

Syntax

```
@property (nonatomic, readonly) SMPOpenUIImage *  
buttonImage
```

buttonText property

The text that the button should display.

Syntax

```
@property (nonatomic, readonly) NSString * buttonText
```

buttonType property

The button type.

Syntax

```
@property (nonatomic, readonly) SMPOpenUIButtonType  
buttonType
```

Usage

Possible types are checkbox, radio, and push buttons. See ButtonStyle enum.

selected property

The selected state of the button.

Syntax

```
@property (nonatomic , readonly) BOOL selected
```

supportsAction property

Whether or not there is an action tied to the button.

Syntax

```
@property (nonatomic , readonly) BOOL supportsAction
```

value property

Gets the current value.

Syntax

```
@property (nonatomic , readonly) BOOL value
```

Usage

For a button, this is synonymous with "selected".

SMPOpenUICollectionDisplayAdapter protocol

Protocol for a field extension representing a collection.

Syntax

```
@protocol SMPOpenUICollectionDisplayAdapter
```

allObjectsChanged: method

Called to inform the adapter that the collection has changed enough that it needs to be completely refreshed.

Syntax

```
- (void) allObjectsChanged : (id<  
SMPOpenUICollectionDisplayModel >) model
```

Parameters

- **model** – the model.

initWithCollectionDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUICollectionDisplayAdapter > )
initWithCollectionDisplayModel : (id<
SMPOpenUICollectionDisplayModel > ) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didSelectObjectAtIndex: method

Called to inform the adapter that the selection index has changed.

Syntax

```
- (void) model : (id< SMPOpenUICollectionDisplayModel > )
model didSelectObjectAtIndex : (NSIndexPath *) indexPath
```

Parameters

- **model** – the model.
- **indexPath** – the index path of the object that now has the selection.

Usage

This is not called when the user selects a different index. This is called when something in Agentry causes the selection to change. This can happen through update rules and retargetting that Agentry handles. This can also happen if the currently selected item gets deleted.

model:objectAddedAtIndex: method

Called to inform the adapter that an object has been added to the collection at the specified position.

Syntax

```
- (void) model : (id< SMPOpenUICollectionDisplayModel > )
model objectAddedAtIndex : (NSIndexPath *) indexPath
```

Parameters

- **model** – the model.
- **indexPath** – the index path of the added object.

model:objectChangedAtIndex: method

Called to inform the adapter that the object at the specified position has changed enough that it needs to be completely refreshed.

Syntax

```
- (void) model : (id< SMPOpenUICollectionDisplayModel >)
model objectChangedAtIndex : (NSIndexPath *) indexPath
```

Parameters

- **model** – the model.
- **indexPath** – the index path of the changed object.

model:objectDeletedAtIndex: method

Called to inform the adapter that the object at the specified position has been deleted and needs to be removed.

Syntax

```
- (void) model : (id< SMPOpenUICollectionDisplayModel >)
model objectDeletedAtIndex : (NSIndexPath *) indexPath
```

Parameters

- **model** – the model.
- **indexPath** – the index path of the object that needs to be removed.

SMPOpenUICollectionDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a collection.

Syntax

```
@protocol SMPOpenUICollectionDisplayModel
```

collection method

Returns the collection.

Syntax

```
- (id< SMPDataAPIProtocol >) collection
```


Returns

The collection

displayedObjectAtIndex: method

Syntax

```
- (id< SMPDataAPIProtocol >) displayedObjectAtIndex :
(NSUInteger) index
```

processInputSelection: method

Processes the selection of a descendant object of the collection.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputSelection :
(NSInteger) selection
```

Parameters

- **selection** – the position of the selected descendant object.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

displayedObjectCount property

Returns the number of displayable objects.

Syntax

```
@property (nonatomic , readonly) NSUInteger
displayedObjectCount
```

selection property

The current selected child object.

Syntax

```
@property (nonatomic , readonly) NSIndexPath * selection
```

SMPOpenUIDateAndTimeDisplayAdapter protocol

Protocol for a field extension representing a display-only date and time field.

Syntax

```
@protocol SMPOpenUIDateAndTimeDisplayAdapter
```

initWithDateAndTimeDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDateAndTimeDisplayAdapter >)  
initWithDateAndTimeDisplayModel : (id<  
SMPOpenUIDateAndTimeDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDateAndTime: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDateAndTimeDisplayModel >)  
model didChangeDateAndTime : (NSDate *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIDateAndTimeDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only date and time.

Syntax

```
@protocol SMPOpenUIDateAndTimeDisplayModel
```

Derived classes

- *SMPOpenUIDateAndTimeEditModel* on page 1156

value property

The current date value.

Syntax

```
@property (nonatomic, readonly) NSDate * value
```

SMPOpenUIDateAndTimeEditAdapter protocol

Protocol for a field extension representing an editable date and time field.

Syntax

```
@protocol SMPOpenUIDateAndTimeEditAdapter
```

initWithDateAndTimeEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDateAndTimeEditAdapter >) initWithDateAndTimeEditModel : (id< SMPOpenUIDateAndTimeEditModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDateAndTime: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDateAndTimeEditModel >) model didChangeDateAndTime : (NSDate *) value
```

Parameters

- **model** – the model.

- **value** – the updated value the field should display.

SMPOpenUIDateAndTimeEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable date and time.

Syntax

```
@protocol SMPOpenUIDateAndTimeEditModel
```

processInputDateAndTime: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputDateAndTime : (NSDate *) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

SMPOpenUIDateDisplayAdapter protocol

Protocol for a field extension representing a display-only date field.

Syntax

```
@protocol SMPOpenUIDateDisplayAdapter
```

initWithDateDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDateDisplayAdapter >) initWithDateDisplayModel : (id< SMPOpenUIDateDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDate: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDateDisplayModel >) model
didChangeDate : (NSDate *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIDateDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only date.

Syntax

```
@protocol SMPOpenUIDateDisplayModel
```

Derived classes

- *SMPOpenUIDateEditModel* on page 1158

value property

The current date value.

Syntax

```
@property (nonatomic, readonly) NSDate * value
```

Usage

The time portion of the NSDate will be midnight.

SMPOpenUIDateEditAdapter protocol

Protocol for a field extension representing an editable date field.

Syntax

```
@protocol SMPOpenUIDateEditAdapter
```

initWithDateEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDateEditAdapter >) initWithDateEditModel : (  
id< SMPOpenUIDateEditModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDate: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDateEditModel >) model  
didChangeDate : (NSDate *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIDateEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable date.

Syntax

```
@protocol SMPOpenUIDateEditModel
```

processInputDate: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputDate : (NSDate  
*) value
```

Parameters

- **value** – the value to process. The time portion of this date will be set to midnight.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

SMPOpenUIDecimalDisplayAdapter protocol

Protocol for a field extension representing a display-only decimal field.

Syntax

```
@protocol SMPOpenUIDecimalDisplayAdapter
```

initWithDecimalDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDecimalDisplayAdapter >)  
initWithDecimalDisplayModel : (id<  
SMPOpenUIDecimalDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDecimal: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDecimalDisplayModel >) model  
didChangeDecimal : (double) value
```

Parameters

- **model** – the model.

- **value** – the updated value the field should display.

SMPOpenUIDecimalDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only decimal.

Syntax

```
@protocol SMPOpenUIDecimalDisplayModel
```

Derived classes

- *SMPOpenUIDecimalEditModel* on page 1161

value property

The current decimal value.

Syntax

```
@property (nonatomic, readonly) double value
```

SMPOpenUIDecimalEditAdapter protocol

Protocol for a field extension representing an editable decimal field.

Syntax

```
@protocol SMPOpenUIDecimalEditAdapter
```

initWithDecimalEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDecimalEditAdapter > )  
initWithDecimalEditModel : (id< SMPOpenUIDecimalEditModel > )  
model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDecimal: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDecimalEditModel >) model
didChangeDecimal : (double) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIDecimalEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable decimal.

Syntax

```
@protocol SMPOpenUIDecimalEditModel
```

processInputDecimal: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputDecimal :
(double) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

maximumValue property

The maximum value accepted for the field.

Syntax

```
@property (nonatomic, readonly) double maximumValue
```

Usage

If there is no maximum value defined, it returns DBL_MAX.

minimumValue property

The minimum value accepted for the field.

Syntax

```
@property (nonatomic, readonly) double minimumValue
```

Usage

If there is no minimum value defined, it returns DBL_MIN.

SMPOpenUIDurationDisplayAdapter protocol

Protocol for a field extension representing a display-only duration field.

Syntax

```
@protocol SMPOpenUIDurationDisplayAdapter
```

initWithDurationDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDurationDisplayAdapter >)  
initWithDurationDisplayModel : (id<  
SMPOpenUIDurationDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDuration: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDurationDisplayModel >) model  
didChangeDuration : (NSInteger) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display. This value is in seconds.

model:didChangeFractionalHour: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDurationDisplayModel >) model
didChangeFractionalHour : (double) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display. This value is in fractional hours.

Usage

This will be called when the display mode for the duration is fractional hours.

SMPOpenUIDurationDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only duration.

Syntax

```
@protocol SMPOpenUIDurationDisplayModel
```

Derived classes

- *SMPOpenUIDurationEditModel* on page 1165

displayFormat property

The display format specified for the duration.

Syntax

```
@property (nonatomic , readonly)
SMPOpenUIDurationDisplayFormat displayFormat
```

fractionalHourValue property

Gets the current value for the duration in decimal hours.

Syntax

```
@property (nonatomic , readonly) double fractionalHourValue
```

value property

The current value for the duration in seconds.

Syntax

```
@property (nonatomic, readonly) NSInteger value
```

SMPOpenUIDurationEditAdapter protocol

Protocol for a field extension representing an editable duration field.

Syntax

```
@protocol SMPOpenUIDurationEditAdapter
```

initWithDurationEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIDurationEditAdapter >)  
initWithDurationEditModel : (id< SMPOpenUIDurationEditModel  
>) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeDuration: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDurationEditModel >) model  
didChangeDuration : (NSInteger) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display. This value is in seconds.

model:didChangeFractionalHour: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIDurationEditModel >) model
didChangeFractionalHour : (double) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display. This value is in fractional hours.

Usage

This will be called when the display mode for the duration is set to fractional hours.

SMPOpenUIDurationEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable duration.

Syntax

```
@protocol SMPOpenUIDurationEditModel
```

processInputDuration: method

Process the current input.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputDuration :
(NSInteger) value
```

Parameters

- **value** – input value of duration in seconds.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

processInputFractionalHour: method

Process the current input.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputFractionalHour :  
  (double) fractionalHourValue
```

Parameters

- **fractionalHourValue** – input value of duration in fractional hours.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

This should be used when SMPOpenUIDurationDisplayFormatDecHour is the display format.

maximumFractionalHourValue property

The maximum value accepted for the field in fractional hours.

Syntax

```
@property (nonatomic, readonly) double  
maximumFractionalHourValue
```

Usage

This should be used when SMPOpenUIDurationDisplayFormatDecHour is the display format. If no maximum value is setup for this field, DBL_MAX will be returned.

maximumValue property

The maximum value accepted for the field in seconds.

Syntax

```
@property (nonatomic, readonly) NSInteger maximumValue
```

Usage

If no maximum value is setup for this field, NSIntegerMax will be returned.

minimumFractionalHourValue property

The minimum value accepted for the field in fractional hours.

Syntax

```
@property (nonatomic, readonly) double
minimumFractionalHourValue
```

Usage

This should be used when SMPOpenUIDurationDisplayFormatDecHour is the display format. If no minimum value is setup for this field, DBL_MIN will be returned.

minimumValue property

The minimum value accepted for the field in seconds.

Syntax

```
@property (nonatomic, readonly) NSInteger minimumValue
```

Usage

If no minimum value is setup for this field, NSIntegerMin will be returned.

SMPOpenUIEmbeddedImageDisplayAdapter protocol

Protocol for a field extension representing an embedded image field.

Syntax

```
@protocol SMPOpenUIEmbeddedImageDisplayAdapter
```

initWithEmbeddedImageModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIEmbeddedImageDisplayAdapter >)
initWithEmbeddedImageModel : (id<
SMPOpenUIEmbeddedImageDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeImage: method

Called to inform the adapter that the field's underlying image has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIEmbeddedImageDisplayModel >)  
model didChangeImage : ( SMPOpenUIImage * ) image
```

Parameters

- **model** – the model.
- **image** – the new image the field should display.

modelDidChangeImageCellSelection: method

Called to inform the adapter that image selection has changed.

Syntax

```
- (void) modelDidChangeImageCellSelection : (id<  
SMPOpenUIEmbeddedImageDisplayModel > ) model
```

Parameters

- **model** – the model.

SMPOpenUIEmbeddedImageDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing an Embedded Image.

Syntax

```
@protocol SMPOpenUIEmbeddedImageDisplayModel
```

Usage

An Embedded Image can be sectioned into a grid of rows and columns, and each cell can be selected or not. Agentry Actions might also be launched as a result of selecting a cell.

imageCellClickedAtRow:andColumn: method

Call this to inform agentry that a cell at a given position has been clicked.

Syntax

```
- (void) imageCellClickedAtRow : (NSUInteger) row andColumn  
: (NSUInteger) column
```


Parameters

- **row** – the row of the clicked cell.
- **column** – the column of the clicked cell.

isImageCellSelectedAtRow:andColumn: method

Call this to find out whether a cell at a given position has been clicked.

Syntax

```
- ( BOOL ) isImageCellSelectedAtRow : ( NSUInteger ) row
andColumn : ( NSUInteger ) column
```

Parameters

- **row** – the row of the cell.
- **column** – the column of the cell.

Returns

YES if the specified cell is selected. NO otherwise.

columns property

The number of columns defined for the image.

Syntax

```
@property ( nonatomic , readonly ) NSUInteger columns
```

highlightSelectedColor property

The highlight selected color to use for the selected cells.

Syntax

```
@property ( nonatomic , readonly ) UIColor *
highlightSelectedColor
```

image property

The image to display.

Syntax

```
@property ( nonatomic , readonly ) SMPOpenUIImage * image
```

rows property

The number of rows defined for the image.

Syntax

```
@property ( nonatomic , readonly ) NSUInteger rows
```

SMPOpenUIExternalDataDisplayAdapter protocol

Protocol for a field extension representing a display-only external data field.

Syntax

```
@protocol SMPOpenUIExternalDataDisplayAdapter
```

initWithExternalDataDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIExternalDataDisplayAdapter >)  
initWithExternalDataDisplayModel : (id<  
SMPOpenUIExternalDataDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeExternalData: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIExternalDataDisplayModel >)  
model didChangeExternalData : (NSString *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIExternalDataDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only external data.

Syntax

```
@protocol SMPOpenUIExternalDataDisplayModel
```

Derived classes

- *SMPOpenUIExternalDataEditModel* on page 1172

value property

The current filename value.

Syntax

```
@property (nonatomic, readonly) NSString * value
```

SMPOpenUIExternalDataEditAdapter protocol

Protocol for a field extension representing an editable date field.

Syntax

```
@protocol SMPOpenUIExternalDataEditAdapter
```

initWithExternalDataEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIExternalDataEditAdapter >)  
initWithExternalDataEditModel : (id<  
SMPOpenUIExternalDataEditModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeExternalData: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIExternalDataEditModel >) model  
didChangeExternalData : (NSString *) value
```

Parameters

- **model** – the model.

- **value** – the updated value the field should display.

SMPOpenUIExternalDataEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable external data.

Syntax

```
@protocol SMPOpenUIExternalDataEditModel
```

processInputExternalData: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputExternalData : (NSString *) value
```

Parameters

- **value** – the value to process. This is the path to the external data file.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

SMPOpenUIFieldAdapter protocol

The base class for the protocols that must be implemented by all Open UI field extension classes.

Syntax

```
@protocol SMPOpenUIFieldAdapter
```

Derived classes

- *SMPOpenUIBooleanDisplayAdapter* on page 1145
- *SMPOpenUIBooleanEditAdapter* on page 1146
- *SMPOpenUIButtonDisplayAdapter* on page 1148
- *SMPOpenUICollectionDisplayAdapter* on page 1150
- *SMPOpenUIDateAndTimeDisplayAdapter* on page 1154
- *SMPOpenUIDateAndTimeEditAdapter* on page 1155
- *SMPOpenUIDateDisplayAdapter* on page 1156

- *SMPOpenUIDateEditAdapter* on page 1157
- *SMPOpenUIDecimalDisplayAdapter* on page 1159
- *SMPOpenUIDecimalEditAdapter* on page 1160
- *SMPOpenUIDurationDisplayAdapter* on page 1162
- *SMPOpenUIDurationEditAdapter* on page 1164
- *SMPOpenUIEmbeddedImageDisplayAdapter* on page 1167
- *SMPOpenUIExternalDataDisplayAdapter* on page 1170
- *SMPOpenUIExternalDataEditAdapter* on page 1171
- *SMPOpenUIIntegerDisplayAdapter* on page 1181
- *SMPOpenUIIntegerEditAdapter* on page 1183
- *SMPOpenUILabelDisplayAdapter* on page 1184
- *SMPOpenUILocationDisplayAdapter* on page 1186
- *SMPOpenUILocationEditAdapter* on page 1187
- *SMPOpenUIStringDisplayAdapter* on page 1188
- *SMPOpenUIStringEditAdapter* on page 1190
- *SMPOpenUITimeDisplayAdapter* on page 1192
- *SMPOpenUITimeEditAdapter* on page 1193
- *SMPOpenUIUnsignedIntegerDisplayAdapter* on page 1194
- *SMPOpenUIUnsignedIntegerEditAdapter* on page 1196

Usage

This is an "abstract" protocol, in that you need to implement one of its child protocols so that there is an `initWithXxxModel` method. The client host will create an instance of the specified class, and call its `initWithXxxModel` method. When the extension control is to be displayed, the `viewForFrame:` method will be called and the returned view will be added as a subview of the Agentry Screen. Each adapter will have a host view and view controller that will determine the space dedicated for the adapter.

agentryShouldDisplayLabel method

Called to ask the adapter if Agentry should handle displaying the label for the field or leave it to the extension.

Syntax

```
- (BOOL) agentryShouldDisplayLabel
```

Returns

YES if Agentry should handle the label, NO if extension handles the label.

Usage

If this method returns YES, Agentry will handle displaying the label, including hyperlink functionality. If this method returns NO, the extension takes responsibility for the label (and is free to just not bother with it).

This is optional. If not present, assumes NO.

agentryShouldDisplayValidationFailure method

Called to ask the adapter if Agentry should handle displaying validation failure text or leave it to the extension.

Syntax

```
- ( BOOL ) agentryShouldDisplayValidationFailure
```

Returns

YES if Agentry should handle the validation failure text, NO if the extension handles the validation failure text.

Usage

If this method returns YES, Agentry will handle displaying the field validation failure text, and do the necessary layout adjustments for it. If this method returns NO, the extension takes responsibility for displaying the field validation failure text.

This is optional. If not present, assumes YES and Agentry displays the validation failure message.

autosizeBehavior method

Called to ask the adapter what its desired autosize behavior is.

Syntax

```
- ( SMPOpenUIAutosizeBehavior ) autosizeBehavior
```

Returns

The desired autosize behavior. If this method is not implemented or the return value is unknown, SMPOpenUIAutosizeBehaviorNone will be used.

Usage

See SMPOpenUIAutosizeBehavior for possible values.

model:didSetEnabled: method

Called to inform the adapter that the host widget has been enabled or disabled.

Syntax

```
- (void) model : (id< SMPOpenUIFieldModel >) model
didSetEnabled : (BOOL) enabled
```

Parameters

- **model** – the model
- **enabled** – YES to indicate it is enabled, NO to indicate it is disabled

Usage

The extension should give some kind of indication to the user that it is disabled. Optional.

model:didSetHyperlinkEnabled: method

Called to inform the adapter that the enable state of the hyperlink has changed.

Syntax

```
- (void) model : (id< SMPOpenUIFieldModel >) model
didSetHyperlinkEnabled : (BOOL) enabled
```

Parameters

- **model** – the model
- **enabled** – YES if hyperlink is enabled, NO if it hyperlink is disabled

Usage

Only called if the extension is handling the label functionality, and a hyperlink is defined
Optional.

model:didSetValid:withValidationFailureText: method

Called to inform the adapter that the field's valid state has changed.

Syntax

```
- (void) model : (id< SMPOpenUIFieldModel >) model
didSetValid : (BOOL) valid withValidationFailureText :
(NSString *) text
```

Parameters

- **model** – the model
- **valid** – YES if the field value is valid, NO for invalid.
- **text** – the message to display to the user if the field is invalid.

Usage

The field has either become invalid and the user needs to be informed with the validation message or valid and any previous validation failure text needs to be hidden. The validation message will contain information that tells the user why their field is invalid.

model:didSetVisible: method

Called to inform the adapter that the host widget has been shown or hidden.

Syntax

```
- (void) model : (id< SMPOpenUIFieldModel >) model  
didSetVisible : (BOOL) visible
```

Parameters

- **model** – the model
- **visible** – YES to indicate it is visible, NO to indicate it is hidden

Usage

The UIView for the extension will be show or hidden automatically. Optional.

model:didUpdateLabel: method

Called to inform the adapter that the text of the label has changed.

Syntax

```
- (void) model : (id< SMPOpenUIFieldModel >) model  
didUpdateLabel : (NSString *) label
```

Parameters

- **model** – the model
- **label** – The new value for the label

Usage

Only called if the extension is handling the label functionality, and the label is defined with a rule Optional.

model:wantsExtensionString: method

Called by the Agentry to get the value for the specified string.

Syntax

```
- (NSString *) model : (id< SMPOpenUIFieldModel >) model
wantsExtensionString : (NSString *) stringName
```

Parameters

- **model** – the model
- **stringName** – The string that Agentry is requesting

Returns

The value the extension determines based on the specified key

Usage

In the definitions, there are specified keys. The string passed in is a key, the value is returned from the extension.

model:wantsViewHeightForWidth: method

Called to ask the adapter the height needed for its view for a given width for layout calculations.

Syntax

```
- (NSUInteger) model : (id< SMPOpenUIFieldModel >) model
wantsViewHeightForWidth : (NSUInteger) width
```

Parameters

- **model** – the model
- **width** – the width for the field

Usage

This method will only be called if the height of the field is set to Auto in the Editor and the adapter has reported that its desired autoSizeBehavior is SMPOpenUIAutoSizeBehaviorWrapContent.

If the adapter reports its desired autoSize behavior is SMPOpenUIAutoSizeBehaviorNone, or if this method is not implemented, standard Agentry layout rules will be used to determine the height of the field.

viewForFrame: method

Returns the UIView that will be added as a subview to the host's UIView This will be called one time after initWithXxxModel: has been called.

Syntax

```
- (UIView *) viewForFrame : (CGRect) frame
```

Parameters

- **frame** – the frame.

SMPOpenUIFieldModel protocol

This is the protocol implemented by all model objects that are given to an adapter extension so it can interface with Agency.

Syntax

```
@protocol SMPOpenUIFieldModel
```

Derived classes

- *SMPOpenUIBooleanDisplayModel* on page 1146
- *SMPOpenUIButtonDisplayModel* on page 1149
- *SMPOpenUICollectionDisplayModel* on page 1152
- *SMPOpenUIDateAndTimeDisplayModel* on page 1154
- *SMPOpenUIDateDisplayModel* on page 1157
- *SMPOpenUIDecimalDisplayModel* on page 1160
- *SMPOpenUIDurationDisplayModel* on page 1163
- *SMPOpenUIEmbeddedImageDisplayModel* on page 1168
- *SMPOpenUIExternalDataDisplayModel* on page 1170
- *SMPOpenUIIntegerDisplayModel* on page 1182
- *SMPOpenUILabelDisplayModel* on page 1185
- *SMPOpenUILocationDisplayModel* on page 1186
- *SMPOpenUIStringDisplayModel* on page 1189
- *SMPOpenUITimeDisplayModel* on page 1193
- *SMPOpenUIUnsignedIntegerDisplayModel* on page 1195

Usage

See its derived protocols for specific data types.

agentryActionEnableState: method

Asks Agentry what the current enable state is for the action specified by name.

Syntax

```
- ( SMPOpenUIActionEnableType ) agentryActionEnableState :  
  ( NSString * ) actionName
```

Parameters

- **actionName** – The action name as a string

Returns

The enable state

Usage

It will either be enabled, disabled, no-op(action not found), or error.

agentryString: method

Asks Agentry for a specific string value.

Syntax

```
- ( NSString * ) agentryString : ( NSString * ) stringName
```

Parameters

- **stringName** – The string the extension is requesting.

Returns

The value paired with that string.

Usage

In the definitions there are key/value pairs. The String passed in is a key, the value is returned. If no key exists for the specified string, an empty String will be returned.

executeAgentryAction: method

Asks Agentry to execute the action specified by name.

Syntax

```
- ( SMPOpenUIActionResult ) executeAgentryAction : ( NSString  
  * ) actionName
```

Parameters

- **actionName** – The action name as a string

Returns

The result of trying to run the action

Usage

This should only be called if `agentryActionEnableState` returns `ActionEnable` for the specified action.

executeHyperlinkAction method

Executes the field's hyperlink action (if the hyperlink action is enabled).

Syntax

```
- ( SMPOpenUIActionResult ) executeHyperlinkAction
```

Returns

The action result

requestLayoutHeigh: method

Used to inform Agentry that a new height is desired for an autosizing field.

Syntax

```
- ( void ) requestLayoutHeigh : ( NSInteger ) newHeight
```

Parameters

- **newHeight** – the desired height for the extension.

Usage

If the field is not autosizing, this request will be ignored. If it is autosizing and the extension can handle autosizing, Agentry will fire layout calculations and it might query the extension for the size again letting it know what its final width will be. See `model:wantsViewHeightForWidth: (SMPOpenUIFieldAdapter-p)` in the `SMPOpenUIFieldAdapter` protocol.

autosizing property

A Boolean value representing whether the field is set to Auto height in the Editor.

Syntax

```
@property ( nonatomic , readonly ) BOOL autosizing
```

Usage

The extension may choose to respond to this by providing the height of the field whenever requested by Agentry depending on what it wants to display. See `model:wantsViewHeightForWidth:` (`SMPOpenUIFieldAdapter-p`) in the `SMPOpenUIFieldAdapter` protocol.

The extension may also choose to notify Agentry that it wants a new height for an autosizing field via the method `requestLayoutHeight:` in this protocol.

enabled property

A Boolean value representing whether the field is currently enabled based on current rule evaluation.

Syntax

```
@property (nonatomic, readonly) BOOL enabled
```

hidden property

A Boolean value representing whether or not the field is currently hidden based on current rule evaluations.

Syntax

```
@property (nonatomic, readonly) BOOL hidden
```

hyperlinkEnabled property

A Boolean value representing whether or not the hyperlink action is enabled.

Syntax

```
@property (nonatomic, readonly) BOOL hyperlinkEnabled
```

label property

The label the field would like the extension to display.

Syntax

```
@property (nonatomic, readonly) NSString * label
```

SMPOpenUIIntegerDisplayAdapter protocol

Protocol for a field extension representing a display-only integer field.

Syntax

```
@protocol SMPOpenUIIntegerDisplayAdapter
```

initWithIntegerDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIIntegerDisplayAdapter >)  
initWithIntegerDisplayModel : (id<  
SMPOpenUIIntegerDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeInteger: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIIntegerDisplayModel >) model  
didChangeInteger : (NSInteger) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIIntegerDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only integer.

Syntax

```
@protocol SMPOpenUIIntegerDisplayModel
```

Derived classes

- *SMPOpenUIIntegerEditModel* on page 1183

value property

The current integer value.

Syntax

```
@property (nonatomic, readonly) NSInteger value
```

SMPOpenUIIntegerEditAdapter protocol

Protocol for a field extension representing an editable integer field.

Syntax

```
@protocol SMPOpenUIIntegerEditAdapter
```

initWithIntegerEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIIntegerEditAdapter >)
initWithIntegerEditModel : (id< SMPOpenUIIntegerEditModel >)
model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeInteger: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIIntegerEditModel >) model
didChangeInteger : (NSInteger) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIIntegerEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable integer.

Syntax

```
@protocol SMPOpenUIIntegerEditModel
```

processInputInteger: method

Processes the input of the field.

Syntax

```
- ( SMPOpenUIProcessInputReturn ) processInputInteger :  
  ( NSInteger ) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

maximumValue property

The maximum integer value that will be accepted.

Syntax

```
@property ( nonatomic , readonly ) NSInteger maximumValue
```

Usage

If no maximum value is set up for this field, NSIntegerMax will be returned.

minimumValue property

The minimum integer value that will be accepted.

Syntax

```
@property ( nonatomic , readonly ) NSInteger minimumValue
```

Usage

If no minimum value is set up for this field, NSIntegerMin will be returned.

SMPOpenUILabelDisplayAdapter protocol

Protocol for a field extension representing a label field.

Syntax

```
@protocol SMPOpenUILabelDisplayAdapter
```


initWithLabelDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUILabelDisplayAdapter >)
initWithLabelDisplayModel : (id< SMPOpenUILabelDisplayModel
>) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeLabel: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUILabelDisplayModel >) model
didChangeLabel : (NSString *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUILabelDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a label.

Syntax

```
@protocol SMPOpenUILabelDisplayModel
```

value property

The text the label should display.

Syntax

```
@property (nonatomic, readonly) NSString * value
```

SMPOpenUILocationDisplayAdapter protocol

Protocol for a field extension representing a display-only location field.

Syntax

```
@protocol SMPOpenUILocationDisplayAdapter
```

initWithLocationDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUILocationDisplayAdapter > )  
initWithLocationDisplayModel : (id<  
SMPOpenUILocationDisplayModel > ) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeLocation: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUILocationDisplayModel > ) model  
didChangeLocation : ( SMPOpenUILocation * ) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUILocationDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only location.

Syntax

```
@protocol SMPOpenUILocationDisplayModel
```

Derived classes

- *SMPOpenUILocationEditModel* on page 1188

value property

The current location value as an autoreleased SMPOpenUILocation.

Syntax

```
@property (nonatomic, readonly) SMPOpenUILocation * value
```

SMPOpenUILocationEditAdapter protocol

Protocol for a field extension representing an editable location field.

Syntax

```
@protocol SMPOpenUILocationEditAdapter
```

initWithLocationEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUILocationEditAdapter >)
initWithLocationEditModel : (id< SMPOpenUILocationEditModel
>) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeLocation: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUILocationEditModel >) model
didChangeLocation : (SMPOpenUILocation *) value
```

Parameters

- **model** – the model.

- **value** – the updated value the field should display.

SMPOpenUILocationEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable location.

Syntax

```
@protocol SMPOpenUILocationEditModel
```

processInputLocation: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputLocation :  
  (SMPOpenUILocation *) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

SMPOpenUIStringDisplayAdapter protocol

Protocol for a field extension representing a display-only string field.

Syntax

```
@protocol SMPOpenUIStringDisplayAdapter
```

initWithStringDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIStringDisplayAdapter > )  
initWithStringDisplayModel : (id<  
SMPOpenUIStringDisplayModel > ) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeString: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIStringDisplayModel >) model
didChangeString : (NSString *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIStringDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only string.

Syntax

```
@protocol SMPOpenUIStringDisplayModel
```

Derived classes

- *SMPOpenUIStringEditModel* on page 1191

allowsCarriageReturn property

A Boolean value representing whether this string display field allows carriage return.

Syntax

```
@property (nonatomic , readonly) BOOL allowsCarriageReturn
```

usesWordWrap property

A Boolean value representing whether this string display field uses word wrap.

Syntax

```
@property (nonatomic , readonly) BOOL usesWordWrap
```

value property

The current string value.

Syntax

```
@property (nonatomic, readonly) NSString * value
```

SMPOpenUIStringEditAdapter protocol

Protocol for a field extension representing an editable string field.

Syntax

```
@protocol SMPOpenUIStringEditAdapter
```

initWithStringEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIStringEditAdapter >) initWithStringEditModel  
: (id< SMPOpenUIStringEditModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeString: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIStringEditModel >) model  
didChangeString : (NSString *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIStringEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable string.

Syntax

```
@protocol SMPOpenUIStringEditModel
```

processInputString: method

Processes the input of the field.

Syntax

```
- ( SMPOpenUIProcessInputReturn ) processInputString :  
  ( NSString * ) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

isPasswordInput property

A Boolean value representing whether the edit field should obscure its input, as would be the case if it were being used to retrieve a password.

Syntax

```
@property ( nonatomic , readonly ) BOOL isPasswordInput
```

maxLength property

The maximum number of characters that the edit text field will allow upon field validation.

Syntax

```
@property ( nonatomic , readonly ) NSInteger maxLength
```

Usage

If no maximum length is setup for this field, 0 will be returned.

minimumLength property

The minimum number of characters that the edit text field will allow upon field validation.

Syntax

```
@property (nonatomic, readonly) NSUInteger minimumLength
```

Usage

If no minimum length is setup for this field, 0 will be returned.

SMPOpenUITimeDisplayAdapter protocol

Protocol for a field extension representing a display-only time field.

Syntax

```
@protocol SMPOpenUITimeDisplayAdapter
```

initWithTimeDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUITimeDisplayAdapter >)  
initWithTimeDisplayModel : (id< SMPOpenUITimeDisplayModel >)  
model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeTime: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUITimeDisplayModel >) model  
didChangeTime : (NSDate *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUITimeDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only time.

Syntax

```
@protocol SMPOpenUITimeDisplayModel
```

Derived classes

- *SMPOpenUITimeEditModel* on page 1194

value property

The current time value.

Syntax

```
@property (nonatomic, readonly) NSDate * value
```

Usage

The date portion will be set to the reference date (Jan 1, 2001)

SMPOpenUITimeEditAdapter protocol

Protocol for a field extension representing an editable time field.

Syntax

```
@protocol SMPOpenUITimeEditAdapter
```

initWithTimeEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUITimeEditAdapter >) initWithTimeEditModel : (
id< SMPOpenUITimeEditModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeTime: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUITimeEditModel >) model  
didChangeTime : (NSDate *) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUITimeEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable time.

Syntax

```
@protocol SMPOpenUITimeEditModel
```

processInputTime: method

Processes the input of the field.

Syntax

```
- (SMPOpenUIProcessInputReturn) processInputTime : (NSDate  
*) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

SMPOpenUIUnsignedIntegerDisplayAdapter protocol

Protocol for a field extension representing a display-only unsigned integer field.

Syntax

```
@protocol SMPOpenUIUnsignedIntegerDisplayAdapter
```

initWithUnsignedIntegerDisplayModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIUnsignedIntegerDisplayAdapter >)
initWithUnsignedIntegerDisplayModel : (id<
SMPOpenUIUnsignedIntegerDisplayModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeUnsignedInteger: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIUnsignedIntegerDisplayModel
>) model didChangeUnsignedInteger : (NSUInteger) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIUnsignedIntegerDisplayModel protocol

Model protocol of object provided to an adapter used for an extension field representing a display-only unsigned integer.

Syntax

```
@protocol SMPOpenUIUnsignedIntegerDisplayModel
```

Derived classes

- *SMPOpenUIUnsignedIntegerEditModel* on page 1196

value property

The current unsigned integer value.

Syntax

```
@property (nonatomic , readonly) NSUInteger value
```

SMPOpenUIUnsignedIntegerEditAdapter protocol

Protocol for a field extension representing an editable unsigned integer field.

Syntax

```
@protocol SMPOpenUIUnsignedIntegerEditAdapter
```

initWithUnsignedIntegerEditModel: method

Called to initialize the extension with its model.

Syntax

```
- (id< SMPOpenUIUnsignedIntegerEditAdapter >)  
initWithUnsignedIntegerEditModel : (id<  
SMPOpenUIUnsignedIntegerEditModel >) model
```

Parameters

- **model** – The model for this adapter to use.

Returns

An initialized object that implements the protocol.

model:didChangeUnsignedInteger: method

Called to inform the adapter that the field's underlying value has changed, and it needs to be updated to display the correct value.

Syntax

```
- (void) model : (id< SMPOpenUIUnsignedIntegerEditModel >)  
model didChangeUnsignedInteger : (NSUInteger) value
```

Parameters

- **model** – the model.
- **value** – the updated value the field should display.

SMPOpenUIUnsignedIntegerEditModel protocol

Model protocol of object provided to an adapter used for an extension field representing an editable unsigned integer.

Syntax

```
@protocol SMPOpenUIUnsignedIntegerEditModel
```

processInputUnsignedInteger: method

Processes the input of the field.

Syntax

```
- ( SMPOpenUIProcessInputReturn )
processInputUnsignedInteger : ( NSUInteger ) value
```

Parameters

- **value** – the value to process.

Returns

SMPOpenUIProcessInputReturn result based on the value passed in.

Usage

Returns the SMPOpenUIProcessInputReturn mask from processing the input.

maximumValue property

The maximum unsigned integer value that will be accepted.

Syntax

```
@property ( nonatomic , readonly ) NSUInteger maximumValue
```

Usage

If no maximum value is setup for this field, NSUIntegerMax will be returned.

minimumValue property

The minimum unsigned integer value that will be accepted.

Syntax

```
@property ( nonatomic , readonly ) NSUInteger minimumValue
```

Usage

If no minimum value is setup for this field, 0 will be returned.

SMPOpenUIActionEnableType enumeration

An enum used to represent the possible enabled states of an action.

Enum Constant Summary

- **SMPOpenUIActionEnableTypeUnknown** – The model is invalid and the action enable type cannot be determined.

- **SMPOpenUIActionEnable** – The action is enabled.
- **SMPOpenUIActionDisable** – The action is disabled.
- **SMPOpenUIActionNoOperation** – The action cannot be found.
- **SMPOpenUIActionError** – The action is found but is invalid.

SMPOpenUIActionResult enumeration

An enum used to represent the possible results of executing an action.

Enum Constant Summary

- **SMPOpenUIActionResultUnknown** – The model is invalid and the action is not being processed at all.
- **SMPOpenUIActionResultBackup** – The action was backed out of by the user.
- **SMPOpenUIActionResultError** – There was an error when running the action.
- **SMPOpenUIActionResultCancel** – The action was canceled by user.
- **SMPOpenUIActionResultPending** – The action is still in progress and has not yet completed.
- **SMPOpenUIActionResultComplete** – The action completed successfully.

SMPOpenUIAutosizeBehavior enumeration

An enum used to tell Agentry what the autosize behavior for the extension should be.

Enum Constant Summary

- **SMPOpenUIAutosizeBehaviorNone** – The adapter view will not be autosized.
- **SMPOpenUIAutosizeBehaviorFillVisible** – The adapter view will be autosized to take up the visible area of the screen.
- **SMPOpenUIAutosizeBehaviorWrapContent** – The adapter will be queried via "wantsViewHeightForWidth:" to determine the height it wants its view to be depending on its content.

SMPOpenUIButtonType enumeration

An enum used to represent the different button types an Agentry Button Field can be set to.

Enum Constant Summary

- **SMPOpenUIButtonTypeUnknown** – The type of the button could not be determined.
- **SMPOpenUIButtonTypeCheckbox** – Check box button.
- **SMPOpenUIButtonTypeRadio** – Radio button.
- **SMPOpenUIButtonTypePush** – Push button.

SMPOpenUIDurationDisplayFormat enumeration

An enum used to represent the different display formats an Agentry Duration field can be set to.

Enum Constant Summary

- **SMPOpenUIDurationDisplayFormatUnknown** – The model is invalid and the display format cannot be queried.
- **SMPOpenUIDurationDisplayFormatHourMinSec** – HH:MM:SS.
- **SMPOpenUIDurationDisplayFormatHourMin** – HH:MM.
- **SMPOpenUIDurationDisplayFormatMinSec** – MM:SS.
- **SMPOpenUIDurationDisplayFormatDecHour** – HH.XX.

SMPOpenUIImagePosition enumeration

An enum used to represent the possible presentation positions for an Agentry image.

Enum Constant Summary

- **SMPOpenUIImagePositionUnknown** – We don't know the image position.
- **SMPOpenUIImagePositionCenter** – Image positioned at the center.
- **SMPOpenUIImagePositionUpperLeft** – Image positioned at the top left.
- **SMPOpenUIImagePositionUpperMiddle** – Image positioned at the top middle.
- **SMPOpenUIImagePositionUpperRight** – Image positioned at the top right.
- **SMPOpenUIImagePositionMiddleLeft** – Image positioned at the middle left.
- **SMPOpenUIImagePositionMiddleRight** – Image positioned at the middle right.
- **SMPOpenUIImagePositionLowerLeft** – Image positioned at the bottom left.
- **SMPOpenUIImagePositionLowerMiddle** – Image positioned at the bottom middle.
- **SMPOpenUIImagePositionLowerRight** – Image positioned at the bottom right.

SMPOpenUIImagePresentation enumeration

An enum used to represent the possible presentation styles for an Agentry image.

Enum Constant Summary

- **SMPOpenUIImagePresentationUnknown** – We don't know the image presentation type.
- **SMPOpenUIImagePresentationLockAspectRatio** – Image should be resized to fit within the field area while maintaining its aspect ratio.
- **SMPOpenUIImagePresentationStretchToFit** – Image should be stretched to fit within the field area.
- **SMPOpenUIImagePresentationCropToFit** – Image should be cropped to fit within the field area.

- **SMPOpenUIImagePresentationFullSize** – The image should be presented full-sized (nothing is selected in the Editor)

SMPOpenUIProcessInputReturn enumeration

An options enum used to represent the return of processInputXxx: on each model.

Enum Constant Summary

- **SMPOpenUIProcessInputReturnNone** – There were no changes to the state of the model or the model could not be accessed and the value is not being processed at all.
- **SMPOpenUIProcessInputReturnValid** – The value passed in is valid (or it was made valid).
- **SMPOpenUIProcessInputReturnMunged** – The value has been adjusted from what the user did, but not in a way that affects its logical value (for example, if the lowercase attribute is set, and an uppercase character was typed in).
- **SMPOpenUIProcessInputReturnChanged** – The value passed in is not the same as the value the model already had.

Agentry OpenUI API for WPF

Use the OpenUI API for WPF to add custom controls to Agentry applications.

IAgentryCollection interface

Placeholder interface for future development.

Visual Basic syntax

```
Public Interface IAgentryCollection Implements IAgentryData
```

C# syntax

```
public interface IAgentryCollection : IAgentryData
```

Members

All members of IAgentryCollection, including inherited members. **Inherited members from IAgentryData**

Modifier and Type	Member	Description
public IAgentryData	<i>Ancestor</i> on page 1247	The parent object of this object.
public List< IAgentryCollection >	<i>Collections()</i> on page 1246	Return a list of collections contained by this object.
public AgentryDataType	<i>DataType</i> on page 1248	Return the type of this object as defined in the Editor.

Modifier and Type	Member	Description
public IAgentryData	<i>Descendant(int)</i> on page 1246	Return a specific data item that's owned by this object.
public int	<i>DescendantCount</i> on page 1248	Return the number of data items owned by this object.
public string	<i>DisplayName</i> on page 1248	Return the display name of this object as specified in the Editor.
public string	<i>InternalName</i> on page 1248	Return the internal name of this object as specified in the Editor.
public List< IAgentryObject >	<i>Objects()</i> on page 1247	Return a list of objects contained by this object.
public List< IAgentryProperty >	<i>Properties()</i> on page 1247	Return a list of properties owned by this object.
public IAgentryData	<i>Root</i> on page 1249	The root object in the data tree for an Agentry module.

IAgentryControlViewModel interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModel Implements
INotifyPropertyChanged
```

C# syntax

```
public interface IAgentryControlViewModel:
INotifyPropertyChanged
```

Derived classes

- *IAgentryControlViewModelCollectionDisplay* on page 1207
- *IAgentryControlViewModelDateTimeDisplay* on page 1212
- *IAgentryControlViewModelDurationDisplay* on page 1223
- *IAgentryControlViewModelFileDisplay* on page 1228
- *IAgentryControlViewModelImage* on page 1230
- *IAgentryControlViewModelLabel* on page 1233
- *IAgentryControlViewModelNumberDisplay< T>* on page 1237
- *IAgentryControlViewModelStringDisplay* on page 1239

Members

All members of IAgentryControlViewModel, including inherited members. **Methods**

Modifier and Type	Method	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Properties

Modifier and Type	Property	Description
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

DoesAgentryActionExist(string) method

Ask Agentry if an action with the specified name exists.

Visual Basic syntax

```
Public Function DoesAgentryActionExist (ByVal actionName As String) As Boolean
```

C# syntax

```
public bool DoesAgentryActionExist (string actionName)
```

Parameters

- **actionName** – the name of the action

Returns

True if the action exists; false if not

ExecuteAgentryAction(string) method

Ask Agentry to execute the action with the specified name.

Visual Basic syntax

```
Public Function ExecuteAgentryAction (ByVal actionName As String) As SMPActionResult
```

C# syntax

```
public SMPActionResult ExecuteAgentryAction (string actionName)
```

Parameters

- **actionName** – the name of the action

Returns

Enum indicating the result of the action

Usage

Only actions defined for this control in the Agentry Editor can be executed.

ExecuteHyperlinkAction() method

Direct the Agentry client to invoke the control's hyperlink action.

Visual Basic syntax

```
Public Function ExecuteHyperlinkAction () As SMPActionResult
```

C# syntax

```
public SMPActionResult ExecuteHyperlinkAction ()
```

Returns

Enum indicating the result of the action

GetAgentryString(string) method

Asks Agentry for a specific string value.

Visual Basic syntax

```
Public Function GetAgentryString (ByVal key As String) As  
String
```

C# syntax

```
public string GetAgentryString (string key)
```

Parameters

- **key** – the key associated with the desired value

Returns

The value associated with the specified key

Usage

In the definitions there are key/value pairs. If the specified string matches a key, its value is returned. Otherwise, an empty string is returned.

IsAgentryActionEnabled(string) method

Ask Agentry if an action with the specified name exists and is enabled.

Visual Basic syntax

```
Public Function IsAgentryActionEnabled (ByVal actionName As  
String) As Boolean
```

C# syntax

```
public bool IsAgentryActionEnabled (string actionName)
```

Parameters

- **actionName** – the name of the action

Returns

True if the action exists and is enabled; false if not

OnPropertyChanged(string) method

The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Visual Basic syntax

```
Public Sub OnPropertyChanged (ByVal strPropertyName As String)
```

C# syntax

```
public void OnPropertyChanged (string strPropertyName)
```

Usage

Property Name: Label IsEnabled IsVisible IsHyperlinkEnabled Error (IDataErrorInfo)

Each type of control has its own value property, which raises the PropertyChanged event when it changes.

String: StringValue Label: Label Integer: NumberValue Identifier: NumberValue Decimal: NumberValue, StringValue Duration: DurationValue Date: Value, DateValue Time: Value, TimeValue Date/Time: Value, DateValue, TimeValue Image: Image Data: FilePath

IsAutoSize property

Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.

Visual Basic syntax

```
Public ReadOnly Property IsAutoSize As Boolean
```

C# syntax

```
public bool IsAutoSize {get;}
```

Usage

The WPF/.NET client doesn't need to do that because of WPF's built-in support for automatically sizing its controls.

IsEnabled property

Determine if the control should be enabled.

Visual Basic syntax

```
Public ReadOnly Property IsEnabled As Boolean
```

C# syntax

```
public bool IsEnabled {get;}
```

IsHyperlinkEnabled property

Determine if the control's hyperlink should be enabled.

Visual Basic syntax

```
Public ReadOnly Property IsHyperlinkEnabled As Boolean
```

C# syntax

```
public bool IsHyperlinkEnabled {get;}
```

IsVisible property

Determine if the control should be visible.

Visual Basic syntax

```
Public ReadOnly Property IsVisible As Boolean
```

C# syntax

```
public bool IsVisible {get;}
```

Label property

Returns the text of this control's label.

Visual Basic syntax

```
Public Property Label As String
```

C# syntax

```
public string Label {get;set;}
```

Usage

(The set method ignores the passed value. It merely determines if the view-model should raise a change event for this property, in case it's changed in the model.)

IAgentryControlViewModelCollectionDisplay interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelCollectionDisplay
    Implements IAgentryControlViewModel, IEnumerable< IAgentryData
>
```

C# syntax

```
public interface IAgentryControlViewModelCollectionDisplay:
    IAgentryControlViewModel, IEnumerable< IAgentryData >
```

Members

All members of IAgentryControlViewModelCollectionDisplay, including inherited members. **Methods**

Modifier and Type	Method	Description
public IAgentryData	<i>DisplayedItemAt(int)</i> on page 1208	Return the displayed item at the passed index.
public SMPProcessInputReturn	<i>SelectItem(int)</i> on page 1209	Select the item at the passed index into the displayed items.

Properties

Modifier and Type	Property	Description
public uint	<i>DisplayedItemCount</i> on page 1209	Get number of items in collection of displayed objects.
public IAgentryData	<i>SelectedItem</i> on page 1209	Return the selected item in the list.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.

Modifier and Type	Member	Description
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a collection control, such as a list-view or tile list.

Note that IEnumerable<> implements IEnumerable, too.

DisplayedItemAt(int) method

Return the displayed item at the passed index.

Visual Basic syntax

```
Public Function DisplayedItemAt (ByVal index As Integer) As
IAgentryData
```


C# syntax

```
public IAgentryData DisplayedItemAt (int index)
```

Parameters

- **index** – Index of desired displayed item

Returns

Item displayed at the passed index

SelectItem(int) method

Select the item at the passed index into the displayed items.

Visual Basic syntax

```
Public Function SelectItem (ByVal index As Integer) As  
SMPPProcessInputReturn
```

C# syntax

```
public SMPPProcessInputReturn SelectItem (int index)
```

Parameters

- **index** – Index of the displayed item to select

Returns

Result of selection

DisplayedItemCount property

Get number of items in collection of displayed objects.

Visual Basic syntax

```
Public ReadOnly Property DisplayedItemCount As UInteger
```

C# syntax

```
public uint DisplayedItemCount {get;}
```

SelectedItem property

Return the selected item in the list.

Visual Basic syntax

```
Public ReadOnly Property SelectedItem As IAgentryData
```

C# syntax

```
public IAgentryData SelectedItem {get;}
```

IAgentryControlViewModelDateTime interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelDateTime Implements
IAgentryControlViewModelDateTimeDisplay
```

C# syntax

```
public interface IAgentryControlViewModelDateTime :
IAgentryControlViewModelDateTimeDisplay
```

Members

All members of IAgentryControlViewModelDateTime, including inherited members.

Methods

Modifier and Type	Method	Description
public void	<i>ProcessInput(DateTime)</i> on page 1211	Set the value of the control's backing property to the passed value.

Inherited members from IAgentryControlViewModelDateTimeDisplay

Modifier and Type	Member	Description
public DateTime	<i>Date Value</i> on page 1214	Return the Date property of the DateTime value.
public TimeSpan	<i>Time Value</i> on page 1214	Return the TimeOfDay property of the DateTime value.
public DateTime	<i>Value</i> on page 1214	Return the DateTime value of the control's backing property.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.

Modifier and Type	Member	Description
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "Date and Time," "Date," or "Time," edit control.

ProcessInput(DateTime) method

Set the value of the control's backing property to the passed value.

Visual Basic syntax

```
Public Sub ProcessInput (ByVal value As Date)
```

C# syntax

```
public void ProcessInput (DateTime value)
```

Parameters

- **value** – New value of this control's backing property

IAgentryControlViewModelDateTimeDisplay interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelDateTimeDisplay  
Implements IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelDateTimeDisplay :  
IAgentryControlViewModel
```

Derived classes

- *IAgentryControlViewModelDateTime* on page 1210

Members

All members of IAgentryControlViewModelDateTimeDisplay, including inherited members.

Properties

Modifier and Type	Property	Description
public DateTime	<i>Date Value</i> on page 1214	Return the Date property of the DateTime value.
public TimeSpan	<i>Time Value</i> on page 1214	Return the TimeOfDay property of the DateTime value.
public DateTime	<i>Value</i> on page 1214	Return the DateTime value of the control's backing property.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.

Modifier and Type	Member	Description
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "Date and Time," "Date," or "Time," display control.

DateValue property

Return the Date property of the DateTime value.

Visual Basic syntax

```
Public Property DateValue As Date
```

C# syntax

```
public DateTime DateValue {get;set;}
```

Usage

If it's null, this returns Today.

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelDateTime interface that derives from this one.)

TimeValue property

Return the TimeOfDay property of the DateTime value.

Visual Basic syntax

```
Public Property TimeValue As TimeSpan
```

C# syntax

```
public TimeSpan TimeValue {get;set;}
```

Usage

If it's null, this returns a zero TimeSpan.

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelDateTime interface that derives from this one.)

Value property

Return the DateTime value of the control's backing property.

Visual Basic syntax

```
Public Property Value As Date
```

C# syntax

```
public DateTime Value {get;set;}
```

Usage

The value may be null.

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelDateTime interface that derives from this one.)

IAgencyControlViewModelDecimal interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgencyControlViewModelDecimal Implements
IAgencyControlViewModelDecimalDisplay
```

C# syntax

```
public interface IAgencyControlViewModelDecimal :
IAgencyControlViewModelDecimalDisplay
```

Members

All members of IAgencyControlViewModelDecimal, including inherited members.

Methods

Modifier and Type	Method	Description
public void	<i>ProcessInput(double)</i> on page 1217	Set the value of the control's backing property to the passed value.

Inherited members from IAgencyControlViewModelNumberDisplay< T >

Modifier and Type	Member	Description
public T	<i>NumberValue</i> on page 1239	Return the value of the control's backing property.

Inherited members from IAgencyControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.

Modifier and Type	Member	Description
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Inherited members from IAgentryControlViewModelStringDisplay

Modifier and Type	Member	Description
public string	<i>StringValue</i> on page 1241	Return the value of the control's backing property.
public bool	<i>WordWrap</i> on page 1241	Determine if word-wrapping is enabled.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.

Modifier and Type	Member	Description
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agency client to invoke the control's hyperlink action.
public string	<i>GetAgencyString(string)</i> on page 1204	Asks Agency for a specific string value.
public bool	<i>IsAgencyActionEnabled(string)</i> on page 1204	Ask Agency if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agency clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agency client's view-model.

This particular interface is only implemented by the view-model of a "Decimal" edit control.

ProcessInput(double) method

Set the value of the control's backing property to the passed value.

Visual Basic syntax

```
Public Sub ProcessInput (ByVal value As Double)
```

C# syntax

```
public void ProcessInput (double value)
```

Parameters

- **value** – New value of this control's backing property

IAgentryControlViewModelDecimalDisplay interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelDecimalDisplay
Implements IAgentryControlViewModelNumberDisplay< T >,
IAgentryControlViewModelStringDisplay
```

C# syntax

```
public interface IAgentryControlViewModelDecimalDisplay:
IAgentryControlViewModelNumberDisplay< T >,
IAgentryControlViewModelStringDisplay
```

Derived classes

- *IAgentryControlViewModelDecimal* on page 1215

Members

All members of IAgentryControlViewModelDecimalDisplay, including inherited members.

Inherited members from IAgentryControlViewModelNumberDisplay< T >

Modifier and Type	Member	Description
public T	<i>NumberValue</i> on page 1239	Return the value of the control's backing property.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.

Modifier and Type	Member	Description
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Inherited members from **IAgentryControlViewModelStringDisplay**

Modifier and Type	Member	Description
public string	<i>StringValue</i> on page 1241	Return the value of the control's backing property.
public bool	<i>WordWrap</i> on page 1241	Determine if word-wrapping is enabled.

Inherited members from **IAgentryControlViewModel**

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.

Modifier and Type	Member	Description
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "Decimal" display control.

IAgentryControlViewModelDuration interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelDuration Implements
IAgentryControlViewModelDurationDisplay
```

C# syntax

```
public interface IAgentryControlViewModelDuration:
IAgentryControlViewModelDurationDisplay
```

Members

All members of IAgentryControlViewModelDuration, including inherited members.

Methods

Modifier and Type	Method	Description
public void	<i>ProcessInput(TimeSpan)</i> on page 1223	Set the value of the control's backing property to the passed value.

Properties

Modifier and Type	Property	Description
public TimeSpan	<i>Maximum Value</i> on page 1223	Return the maximum value permitted for this control.
public TimeSpan	<i>Minimum Value</i> on page 1223	Return the minimum value permitted for this control.

Inherited members from IAgentryControlViewModelDurationDisplay

Modifier and Type	Member	Description
public SMPDurationFormat	<i>DurationFormat</i> on page 1225	Return the format that is set in the Agentry Editor for this control.
public TimeSpan	<i>Duration Value</i> on page 1225	Return the value of the control's backing property.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "Duration" edit control.

ProcessInput(TimeSpan) method

Set the value of the control's backing property to the passed value.

Visual Basic syntax

```
Public Sub ProcessInput (ByVal value As TimeSpan)
```

C# syntax

```
public void ProcessInput (TimeSpan value)
```

Parameters

- **value** – New value of this control's backing property

MaximumValue property

Return the maximum value permitted for this control.

Visual Basic syntax

```
Public ReadOnly Property MaximumValue As TimeSpan
```

C# syntax

```
public TimeSpan MaximumValue {get;}
```

MinimumValue property

Return the minimum value permitted for this control.

Visual Basic syntax

```
Public ReadOnly Property MinimumValue As TimeSpan
```

C# syntax

```
public TimeSpan MinimumValue {get;}
```

IAgentryControlViewModelDurationDisplay interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelDurationDisplay  
Implements IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelDurationDisplay:  
IAgentryControlViewModel
```

Derived classes

- *IAgentryControlViewModelDuration* on page 1221

Members

All members of *IAgentryControlViewModelDurationDisplay*, including inherited members.

Properties

Modifier and Type	Property	Description
public SMPDurationFormat	<i>DurationFormat</i> on page 1225	Return the format that is set in the Agentry Editor for this control.
public TimeSpan	<i>DurationValue</i> on page 1225	Return the value of the control's backing property.

Inherited members from *IAgentryControlViewModel*

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.

Modifier and Type	Member	Description
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "Duration" display control.

DurationFormat property

Return the format that is set in the Agentry Editor for this control.

Visual Basic syntax

```
Public ReadOnly Property DurationFormat As SMPDurationFormat
```

C# syntax

```
public SMPDurationFormat DurationFormat {get;}
```

DurationValue property

Return the value of the control's backing property.

Visual Basic syntax

```
Public Property DurationValue As TimeSpan
```

C# syntax

```
public TimeSpan DurationValue {get;set;}
```

Usage

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelDuration interface that derives from this one.)

IAgentryControlViewModelFile interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelFile Implements
IAgentryControlViewModelFileDisplay
```

C# syntax

```
public interface IAgentryControlViewModelFile:
IAgentryControlViewModelFileDisplay
```

Members

All members of IAgentryControlViewModelFile, including inherited members. **Methods**

Modifier and Type	Method	Description
public void	<i>ProcessInput(string)</i> on page 1227	Set the value of the control's backing property to the passed value.

Inherited members from IAgentryControlViewModelFileDisplay

Modifier and Type	Member	Description
public string	<i>FilePath</i> on page 1229	Return the full path of the file.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.

Modifier and Type	Member	Description
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's `DataContext` property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of an "External File" edit control.

ProcessInput(string) method

Set the value of the control's backing property to the passed value.

Visual Basic syntax

```
Public Sub ProcessInput (ByVal value As String)
```

C# syntax

```
public void ProcessInput (string value)
```

Parameters

- **value** – New value of this control's backing property

IAgentryControlViewModelFileDisplay interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelFileDisplay
Implements IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelFileDisplay:
IAgentryControlViewModel
```

Derived classes

- *IAgentryControlViewModelFile* on page 1226

Members

All members of IAgentryControlViewModelFileDisplay, including inherited members.

Properties

Modifier and Type	Property	Description
public string	<i>FilePath</i> on page 1229	Return the full path of the file.

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.

Modifier and Type	Member	Description
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of an "External File" display control.

FilePath property

Return the full path of the file.

Visual Basic syntax

```
Public Property FilePath As String
```

C# syntax

```
public string FilePath {get;set;}
```

Usage

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelFile interface that derives from this one.)

IAgentryControlViewModelImage interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelImage Implements
IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelImage :
IAgentryControlViewModel
```

Members

All members of IAgentryControlViewModelImage, including inherited members. **Methods**

Modifier and Type	Method	Description
public bool	<i>IsSelected(int, int)</i> on page 1232	
public void	<i>SelectCell(int, int)</i> on page 1232	

Properties

Modifier and Type	Property	Description
public int	<i>Columns</i> on page 1232	This method returns
public System.Windows.Media.ImageSource	<i>Image</i> on page 1232	
public int	<i>Rows</i> on page 1233	
public System.Windows.Media.Color	<i>SelectColor</i> on page 1233	

Inherited members from IAgentryControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.

Modifier and Type	Member	Description
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of an "Embedded Image" control.

IsSelected(int, int) method

Visual Basic syntax

```
Public Function IsSelected (ByVal x As Integer, ByVal y As Integer) As Boolean
```

C# syntax

```
public bool IsSelected (int x, int y)
```

Parameters

- **x** – Zero-based column of the cell to test
- **y** – Zero-based row of the cell to test

Returns

flag indicating if the cell at the specified coordinates is selected

SelectCell(int, int) method

Visual Basic syntax

```
Public Sub SelectCell (ByVal x As Integer, ByVal y As Integer)
```

C# syntax

```
public void SelectCell (int x, int y)
```

Parameters

- **x** – Zero-based column of the cell to select
- **y** – Zero-based row of the cell to select

Columns property

This method returns

Visual Basic syntax

```
Public ReadOnly Property Columns As Integer
```

C# syntax

```
public int Columns {get;}
```

Image property

Visual Basic syntax

```
Public Property Image As System.Windows.Media.ImageSource
```


C# syntax

```
public System.Windows.Media.ImageSource Image {get;set;}
```

Rows property

Visual Basic syntax

```
Public ReadOnly Property Rows As Integer
```

C# syntax

```
public int Rows {get;}
```

SelectColor property

Visual Basic syntax

```
Public ReadOnly Property SelectColor As  
System.Windows.Media.Color
```

C# syntax

```
public System.Windows.Media.Color SelectColor {get;}
```

IAgentryControlViewModelLabel interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelLabel Implements  
IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelLabel :  
IAgentryControlViewModel
```

Members

All members of IAgentryControlViewModelLabel, including inherited members. **Inherited members from IAgentryControlViewModel**

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.

Modifier and Type	Member	Description
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agency client to invoke the control's hyperlink action.
public string	<i>GetAgencyString(string)</i> on page 1204	Asks Agency for a specific string value.
public bool	<i>IsAgencyActionEnabled(string)</i> on page 1204	Ask Agency if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agency clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agency client's view-model.

This particular interface is only implemented by the view-model of a "Label" control.

IAgencyControlViewModelNumber< T > interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgencyControlViewModelNumber< T >
Implements IAgencyControlViewModelNumberDisplay< T >
```

C# syntax

```
public interface IAgencyControlViewModelNumber< T >:
IAgencyControlViewModelNumberDisplay< T >
```

Members

All members of IAgencyControlViewModelNumber< T >, including inherited members.

Methods

Modifier and Type	Method	Description
public void	<i>ProcessInput(T)</i> on page 1237	Set the value of the control's backing property to the passed value.

Properties

Modifier and Type	Property	Description
public T	<i>Maximum</i> on page 1237	Return the maximum value permitted for this control.
public T	<i>Minimum</i> on page 1237	Return the minimum value permitted for this control.

Inherited members from IAgencyControlViewModelNumberDisplay< T >

Modifier and Type	Member	Description
public T	<i>NumberValue</i> on page 1239	Return the value of the control's backing property.

Inherited members from IAgencyControlViewModel

Modifier and Type	Member	Description
public bool	<i>DoesAgencyActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgencyAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.

Modifier and Type	Member	Description
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agency client to invoke the control's hyperlink action.
public string	<i>GetAgencyString(string)</i> on page 1204	Asks Agency for a specific string value.
public bool	<i>IsAgencyActionEnabled(string)</i> on page 1204	Ask Agency if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agency clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agency client's view-model.

This particular interface is only implemented by the view-model of an "Integral" or "Identifier" edit control.

ProcessInput(T) method

Set the value of the control's backing property to the passed value.

Visual Basic syntax

```
Public Sub ProcessInput (ByVal value As T)
```

C# syntax

```
public void ProcessInput (T value)
```

Parameters

- **value** – New value of this control's backing property

Maximum property

Return the maximum value permitted for this control.

Visual Basic syntax

```
Public ReadOnly Property Maximum As T
```

C# syntax

```
public T Maximum {get;}
```

Minimum property

Return the minimum value permitted for this control.

Visual Basic syntax

```
Public ReadOnly Property Minimum As T
```

C# syntax

```
public T Minimum {get;}
```

IAgentryControlViewModelNumberDisplay< T > interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelNumberDisplay< T >  
Implements IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelNumberDisplay< T >:  
IAgentryControlViewModel
```

Derived classes

- *IAgentryControlViewModelDecimalDisplay* on page 1218
- *IAgentryControlViewModelNumber< T >* on page 1235

Members

All members of *IAgentryControlViewModelNumberDisplay< T >*, including inherited members. **Properties**

Modifier and Type	Property	Description
public T	<i>NumberValue</i> on page 1239	Return the value of the control's backing property.

Inherited members from *IAgentryControlViewModel*

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.

Modifier and Type	Member	Description
public bool	<i>Is Visible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of an "Integral" or "Identifier" display control.

The template parameter is either an 'int' or 'uint.'

NumberValue property

Return the value of the control's backing property.

Visual Basic syntax

```
Public Property NumberValue As T
```

C# syntax

```
public T NumberValue {get;set;}
```

Usage

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelNumber interface that derives from this one.)

IAgentryControlViewModelStringDisplay interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelStringDisplay
    Implements IAgentryControlViewModel
```

C# syntax

```
public interface IAgentryControlViewModelStringDisplay :
    IAgentryControlViewModel
```

Derived classes

- *IAgentryControlViewModelDecimalDisplay* on page 1218
- *IAgentryControlViewModelStringEdit* on page 1242

Members

All members of *IAgentryControlViewModelStringDisplay*, including inherited members.

Properties

Modifier and Type	Property	Description
public string	<i>StringValue</i> on page 1241	Return the value of the control's backing property.
public bool	<i>WordWrap</i> on page 1241	Determine if word-wrapping is enabled.

Inherited members from *IAgentryControlViewModel*

Modifier and Type	Member	Description
public bool	<i>DoesAgentryActionExist(string)</i> on page 1203	Ask Agentry if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgentryAction(string)</i> on page 1203	Ask Agentry to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agentry client to invoke the control's hyperlink action.
public string	<i>GetAgentryString(string)</i> on page 1204	Asks Agentry for a specific string value.
public bool	<i>IsAgentryActionEnabled(string)</i> on page 1204	Ask Agentry if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agentry clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.

Modifier and Type	Member	Description
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "String" display control.

StringValue property

Return the value of the control's backing property.

Visual Basic syntax

```
Public Property StringValue As String
```

C# syntax

```
public string StringValue {get;set;}
```

Usage

(The set method serves no purpose in this interface. It's provided solely for the IAgentryControlViewModelString interface that derives from this one.)

WordWrap property

Determine if word-wrapping is enabled.

Visual Basic syntax

```
Public ReadOnly Property WordWrap As Boolean
```

C# syntax

```
public bool WordWrap {get;}
```

IAgentryControlViewModelStringEdit interface

This interface is implemented by the Agentry client.

Visual Basic syntax

```
Public Interface IAgentryControlViewModelStringEdit Implements  
IAgentryControlViewModelStringDisplay
```

C# syntax

```
public interface IAgentryControlViewModelStringEdit :  
IAgentryControlViewModelStringDisplay
```

Members

All members of IAgentryControlViewModelStringEdit, including inherited members.

Methods

Modifier and Type	Method	Description
public void	<i>ProcessInput(string)</i> on page 1244	Set the value of the control's backing property to the passed value.

Properties

Modifier and Type	Property	Description
public bool	<i>AcceptReturn</i> on page 1244	Return a flag indicating if the control should accept the Return key as input.
public bool	<i>IsPassword</i> on page 1244	Return a flag indicating if the control is a password.
public int	<i>MaximumLength</i> on page 1245	Return the maximum length of the string value (in characters).
public int	<i>MinimumLength</i> on page 1245	Return the minimum length of the string value (in characters).

Inherited members from IAgentryControlViewModelStringDisplay

Modifier and Type	Member	Description
public string	<i>StringValue</i> on page 1241	Return the value of the control's backing property.
public bool	<i>WordWrap</i> on page 1241	Determine if word-wrapping is enabled.

Inherited members from **IAgencyControlViewModel**

Modifier and Type	Member	Description
public bool	<i>DoesAgencyActionExist(string)</i> on page 1203	Ask Agency if an action with the specified name exists.
public SMPActionResult	<i>ExecuteAgencyAction(string)</i> on page 1203	Ask Agency to execute the action with the specified name.
public SMPActionResult	<i>ExecuteHyperlinkAction()</i> on page 1204	Direct the Agency client to invoke the control's hyperlink action.
public string	<i>GetAgencyString(string)</i> on page 1204	Asks Agency for a specific string value.
public bool	<i>IsAgencyActionEnabled(string)</i> on page 1204	Ask Agency if an action with the specified name exists and is enabled.
public bool	<i>IsAutoSize</i> on page 1205	Some Agency clients depend on the internal layout manager to determine the size of a control by asking the control to determine its own size.
public bool	<i>IsEnabled</i> on page 1206	Determine if the control should be enabled.
public bool	<i>IsHyperlinkEnabled</i> on page 1206	Determine if the control's hyperlink should be enabled.
public bool	<i>IsVisible</i> on page 1206	Determine if the control should be visible.
public string	<i>Label</i> on page 1206	Returns the text of this control's label.

Modifier and Type	Member	Description
public void	<i>OnPropertyChanged(string)</i> on page 1205	The consumer of this class can listen for the PropertyChanged event in order to handle changes to any of the properties.

Usage

The third-party's custom control's DataContext property is set to an object that implements this interface. The control can use this to request certain information from the Agentry client's view-model.

This particular interface is only implemented by the view-model of a "String" edit control.

ProcessInput(string) method

Set the value of the control's backing property to the passed value.

Visual Basic syntax

```
Public Sub ProcessInput (ByVal value As String)
```

C# syntax

```
public void ProcessInput (string value)
```

Parameters

- **value** – New value of this control's backing property

AcceptReturn property

Return a flag indicating if the control should accept the Return key as input.

Visual Basic syntax

```
Public ReadOnly Property AcceptReturn As Boolean
```

C# syntax

```
public bool AcceptReturn {get;}
```

IsPassword property

Return a flag indicating if the control is a password.

Visual Basic syntax

```
Public ReadOnly Property IsPassword As Boolean
```

C# syntax

```
public bool IsPassword {get;}
```

MaxLength property

Return the maximum length of the string value (in characters).

Visual Basic syntax

```
Public ReadOnly Property MaxLength As Integer
```

C# syntax

```
public int MaxLength {get;}
```

MinLength property

Return the minimum length of the string value (in characters).

Visual Basic syntax

```
Public ReadOnly Property MinLength As Integer
```

C# syntax

```
public int MinLength {get;}
```

IAgentryData interface

Visual Basic syntax

```
Public Interface IAgentryData
```

C# syntax

```
public interface IAgentryData
```

Derived classes

- *IAgentryCollection* on page 1200
- *IAgentryObject* on page 1249
- *IAgentryProperty* on page 1250

Members

All members of IAgentryData, including inherited members. **Methods**

Modifier and Type	Method	Description
public List< IAgentryCollection >	<i>Collections()</i> on page 1246	Return a list of collections contained by this object.
public IAgentryData	<i>Descendant(int)</i> on page 1246	Return a specific data item that's owned by this object.

Modifier and Type	Method	Description
public List< IAgentryObject >	<i>Objects()</i> on page 1247	Return a list of objects contained by this object.
public List< IAgentryProperty >	<i>Properties()</i> on page 1247	Return a list of properties owned by this object.

Properties

Modifier and Type	Property	Description
public IAgentryData	<i>Ancestor</i> on page 1247	The parent object of this object.
public AgentryDataType	<i>DataType</i> on page 1248	Return the type of this object as defined in the Editor.
public int	<i>DescendantCount</i> on page 1248	Return the number of data items owned by this object.
public string	<i>DisplayName</i> on page 1248	Return the display name of this object as specified in the Editor.
public string	<i>InternalName</i> on page 1248	Return the internal name of this object as specified in the Editor.
public IAgentryData	<i>Root</i> on page 1249	The root object in the data tree for an Agentry module.

Collections() method

Return a list of collections contained by this object.

Visual Basic syntax

```
Public Function Collections () As List< IAgentryCollection >
```

C# syntax

```
public List< IAgentryCollection > Collections ()
```

Returns

A list of only the child data items that are collections.

Descendant(int) method

Return a specific data item that's owned by this object.

Visual Basic syntax

```
Public Function Descendant (ByVal index As Integer) As IAgentryData
```

C# syntax

```
public IAgentryData Descendant (int index)
```

Parameters

- **index** – The index of the requested child data object

Returns

The data item at the specified index or nullptr if the index is out of range.

Objects() method

Return a list of objects contained by this object.

Visual Basic syntax

```
Public Function Objects () As List< IAgentryObject >
```

C# syntax

```
public List< IAgentryObject > Objects ()
```

Returns

A list of only the child data items that are objects.

Properties() method

Return a list of properties owned by this object.

Visual Basic syntax

```
Public Function Properties () As List< IAgentryProperty >
```

C# syntax

```
public List< IAgentryProperty > Properties ()
```

Returns

A list of only the child data items that are properties.

Ancestor property

The parent object of this object.

Visual Basic syntax

```
Public ReadOnly Property Ancestor As IAgentryData
```

C# syntax

```
public IAgentryData Ancestor {get;}
```

Usage

Each data object has an ancestor, except the root data object (main object).

DataType property

Return the type of this object as defined in the Editor.

Visual Basic syntax

```
Public ReadOnly Property DataType As AgentryDataType
```

C# syntax

```
public AgentryDataType DataType {get;}
```

DescendantCount property

Return the number of data items owned by this object.

Visual Basic syntax

```
Public ReadOnly Property DescendantCount As Integer
```

C# syntax

```
public int DescendantCount {get;}
```

DisplayName property

Return the display name of this object as specified in the Editor.

Visual Basic syntax

```
Public ReadOnly Property DisplayName As String
```

C# syntax

```
public string DisplayName {get;}
```

InternalName property

Return the internal name of this object as specified in the Editor.

Visual Basic syntax

```
Public ReadOnly Property InternalName As String
```

C# syntax

```
public string InternalName {get;}
```


Root property

The root object in the data tree for an Agentry module.

Visual Basic syntax

```
Public ReadOnly Property Root As IAgentryData
```

C# syntax

```
public IAgentryData Root {get;}
```

Usage

For an Agentry module, the root is the module's Main Object.

IAgentryObject interface

Placeholder interface for future development.

Visual Basic syntax

```
Public Interface IAgentryObject Implements IAgentryData
```

C# syntax

```
public interface IAgentryObject : IAgentryData
```

Members

All members of IAgentryObject, including inherited members. **Inherited members from IAgentryData**

Modifier and Type	Member	Description
public IAgentryData	<i>Ancestor</i> on page 1247	The parent object of this object.
public List< IAgentryCollection >	<i>Collections()</i> on page 1246	Return a list of collections contained by this object.
public AgentryDataType	<i>DataType</i> on page 1248	Return the type of this object as defined in the Editor.
public IAgentryData	<i>Descendant(int)</i> on page 1246	Return a specific data item that's owned by this object.
public int	<i>DescendantCount</i> on page 1248	Return the number of data items owned by this object.
public string	<i>DisplayName</i> on page 1248	Return the display name of this object as specified in the Editor.
public string	<i>InternalName</i> on page 1248	Return the internal name of this object as specified in the Editor.

Modifier and Type	Member	Description
public List< IAgentryObject >	<i>Objects()</i> on page 1247	Return a list of objects contained by this object.
public List< IAgentryProperty >	<i>Properties()</i> on page 1247	Return a list of properties owned by this object.
public IAgentryData	<i>Root</i> on page 1249	The root object in the data tree for an Agentry module.

IAgentryProperty interface

This interface represents a single property of a data object.

Visual Basic syntax

```
Public Interface IAgentryProperty Implements IAgentryData
```

C# syntax

```
public interface IAgentryProperty : IAgentryData
```

Members

All members of IAgentryProperty, including inherited members. **Methods**

Modifier and Type	Method	Description
public bool	<i>ToBoolean()</i> on page 1251	Convert this property's value to a boolean.
public DateTime	<i>ToDate()</i> on page 1252	Convert this property's value to a date.
public DateTime	<i>ToDateTime()</i> on page 1252	Convert this property's value to a date/time.
public double	<i>ToDouble()</i> on page 1252	Convert this property's value to a double.
public int	<i>ToInt()</i> on page 1253	Convert this property's value to an integer.
public string	<i>ToString()</i> on page 1253	Convert this property's value to a string.
public TimeSpan	<i>Time()</i> on page 1253	Convert this property's value to a time.
public uint	<i>ToUInt()</i> on page 1253	Convert this property's value to an unsigned integer.

Properties

Modifier and Type	Property	Description
public AgencyPropertyType	<i>PropertyType</i> on page 1254	The type of property this is (e.g., string or integer).

Inherited members from IAgencyData

Modifier and Type	Member	Description
public IAgencyData	<i>Ancestor</i> on page 1247	The parent object of this object.
public List< IAgencyCollection >	<i>Collections()</i> on page 1246	Return a list of collections contained by this object.
public AgencyDataType	<i>DataType</i> on page 1248	Return the type of this object as defined in the Editor.
public IAgencyData	<i>Descendant(int)</i> on page 1246	Return a specific data item that's owned by this object.
public int	<i>DescendantCount</i> on page 1248	Return the number of data items owned by this object.
public string	<i>DisplayName</i> on page 1248	Return the display name of this object as specified in the Editor.
public string	<i>InternalName</i> on page 1248	Return the internal name of this object as specified in the Editor.
public List< IAgencyObject >	<i>Objects()</i> on page 1247	Return a list of objects contained by this object.
public List< IAgencyProperty >	<i>Properties()</i> on page 1247	Return a list of properties owned by this object.
public IAgencyData	<i>Root</i> on page 1249	The root object in the data tree for an Agency module.

ToBoolean() method

Convert this property's value to a boolean.

Visual Basic syntax

```
Public Function ToBoolean () As Boolean
```

C# syntax

```
public bool ToBoolean ()
```

Returns

This property's value as a boolean

ToDate() method

Convert this property's value to a date.

Visual Basic syntax

```
Public Function ToDate () As Date
```

C# syntax

```
public DateTime ToDate ()
```

Returns

This property's value as a date

ToDateTime() method

Convert this property's value to a date/time.

Visual Basic syntax

```
Public Function ToDateTime () As Date
```

C# syntax

```
public DateTime ToDateTime ()
```

Returns

This property's value as a date/time

ToDouble() method

Convert this property's value to a double.

Visual Basic syntax

```
Public Function ToDouble () As Double
```

C# syntax

```
public double ToDouble ()
```

Returns

This property's value as a double

ToInt() method

Convert this property's value to an integer.

Visual Basic syntax

```
Public Function ToInt () As Integer
```

C# syntax

```
public int ToInt ()
```

Returns

This property's value as an integer

ToString() method

Convert this property's value to a string.

Visual Basic syntax

```
Public Function ToString () As String
```

C# syntax

```
public string ToString ()
```

Returns

This property's value as a string

ToTime() method

Convert this property's value to a time.

Visual Basic syntax

```
Public Function ToTime () As TimeSpan
```

C# syntax

```
public TimeSpan ToTime ()
```

Returns

This property's value as a time

ToUInt() method

Convert this property's value to an unsigned integer.

Visual Basic syntax

```
Public Function ToUInt () As UInteger
```

C# syntax

```
public uint ToUInt ()
```

Returns

This property's value as an unsigned integer

PropertyType property

The type of property this is (e.g., string or integer).

Visual Basic syntax

```
Public ReadOnly Property PropertyType As AgentryPropertyType
```

C# syntax

```
public AgentryPropertyType PropertyType {get;}
```

ICustomAgentryControl interface

The third-party custom control must implement this interface in order to provide the Agentry client with specific information about how it should operate.

Visual Basic syntax

```
Public Interface ICustomAgentryControl
```

C# syntax

```
public interface ICustomAgentryControl
```

Members

All members of ICustomAgentryControl, including inherited members. **Methods**

Modifier and Type	Method	Description
public string	<i>GetExtensionString(string)</i> on page 1255	Return the value of the passed key for this custom control.

Properties

Modifier and Type	Property	Description
public bool	<i>ClientDisplaysLabel</i> on page 1255	Return a flag indicating if the client should display this control's label text.
public bool	<i>ClientDisplaysValidationError</i> on page 1255	Return a flag indicating if the client should display this control's validation error message.

GetExtensionString(string) method

Return the value of the passed key for this custom control.

Visual Basic syntax

```
Public Function GetExtensionString (ByVal key As String) As String
```

C# syntax

```
public string GetExtensionString (string key)
```

ClientDisplaysLabel property

Return a flag indicating if the client should display this control's label text.

Visual Basic syntax

```
Public ReadOnly Property ClientDisplaysLabel As Boolean
```

C# syntax

```
public bool ClientDisplaysLabel {get;}
```

ClientDisplaysValidationError property

Return a flag indicating if the client should display this control's validation error message.

Visual Basic syntax

```
Public ReadOnly Property ClientDisplaysValidationError As Boolean
```

C# syntax

```
public bool ClientDisplaysValidationError {get;}
```

IEnumerable< IAgentryData > class

Visual Basic syntax

```
Public Class IEnumerable< IAgentryData >
```

C# syntax

```
public class IEnumerable< IAgentryData >
```

Derived classes

- *IAgentryControlViewModelCollectionDisplay* on page 1207

AgentryDataType enumeration

Enum Constant Summary

- **Unknown** –
- **Object** –
- **Property** –
- **Collection** –

AgentryPropertyType enumeration

Enum Constant Summary

- **Unknown** –
- **String** –
- **Identifier** –
- **Integer** –
- **Decimal** –
- **Boolean** –
- **Date** –
- **Time** –
- **DateTime** –
- **Duration** –
- **ListSelection** –
- **DataTableSelection** –
- **ComplexTableSelection** –
- **Signature** –
- **ExternalData** –
- **Image** –
- **Location** –

SMPActionResult enumeration

Enum Constant Summary

- **UserBackedOut** –
- **Error** –
- **UserCanceled** –
- **Pending** –
- **Complete** –

SMPActionState enumeration

Enum Constant Summary

- **Enable** –
- **Disable** –
- **NoOperation** –
- **Error** –

SMPDurationFormat enumeration

Enum Constant Summary

- **HMS** –
- **HM** –
- **MS** –
- **FractionalHour** –

SMPPProcessInputReturn enumeration

Enum Constant Summary

- **None** –
- **Valid** –
- **Munged** –
- **Changed** –

Index

- `_clientLastDataUpdateTime` variable
 - `ComplexTable< CTOBJ >` class [Agentry Java System Connection API API] 685
 - `DataTable< DTOBJ extends DataTableObject >` class [Agentry Java System Connection API API] 694
- `_defaultFormatter` variable
 - `AgentryHandler` class [Agentry Java System Connection API API] 650
- `_name` variable
 - `User` class [Agentry Java System Connection API API] 805
- `_rebuilding` variable
 - `ComplexTable< CTOBJ >` class [Agentry Java System Connection API API] 686
- `_server` variable
 - `ServiceEvent` class [Agentry Java System Connection API API] 770
- `_session` variable
 - `ComplexTable< CTOBJ >` class [Agentry Java System Connection API API] 686
 - `DataTable< DTOBJ extends DataTableObject >` class [Agentry Java System Connection API API] 694
 - `Steplet` class [Agentry Java System Connection API API] 785
- `_sessionData` variable
 - `ServiceEvent` class [Agentry Java System Connection API API] 770
- [Agentry Open UI Android SDK API]
 - `Action_BackUp` variable 1124
 - `Action_Cancel` variable 1124
 - `Action_Complete` variable 1124
 - `Action_Error` variable 1125
 - `Action_Pending` variable 1125
 - `ActionDisable` variable 1123
 - `ActionEnable` variable 1123
 - `ActionError` variable 1123
 - `ActionNoOperation` variable 1123
 - `AgentryImage(String, ImageType, ImagePresentation, ImagePosition, int, int, int)` constructor 1113
 - `AgentryLocation(boolean, double, double, int, double)` constructor 1116
 - `Autosize_FillVisible` variable 1125
 - `Autosize_None` variable 1125
 - `Autosize_WrapContent` variable 1126
 - `buttonImageChanged(AgentryImage)` method 1003
 - `ButtonStyleCheckbox` variable 1126
 - `ButtonStylePush` variable 1126
 - `ButtonStyleRadio` variable 1126
 - `DecHour` variable 1127
 - `executeAgentryAction(String)` method 1089
 - `executeHyperlinkAction()` method 1090
 - `fractionalHourValueChanged(double)` method 1019, 1022
 - `getAgentryActionEnableState(String)` method 1090
 - `getAgentryString(String)` method 1090
 - `getAutosizeBehavior()` method 1032
 - `getBitmapData()` method 1113, 1132
 - `getBlue()` method 1120
 - `getButtonImage()` method 1062
 - `getButtonText()` method 1063
 - `getButtonType()` method 1063
 - `getChanged()` method 1122
 - `getColumnCount()` method 1082
 - `getContentHeightForAutosizing(int)` method 1033
 - `getDilution()` method 1116
 - `getDurationDisplayFormat()` method 1076
 - `getExtensionString(String)` method 1033
 - `getFilePath()` method 1086
 - `getFractionalHourValue()` method 1076
 - `getGreen()` method 1120
 - `getHighlightColor()` method 1082
 - `getImage()` method 1083
 - `getImageName()` method 1113, 1132
 - `getImagePosition()` method 1083, 1114
 - `getImagePresentation()` method 1083, 1114
 - `getImageType()` method 1114, 1133
 - `getLabel()` method 1091
 - `getLatitude()` method 1117
 - `getLongitude()` method 1117
 - `getMaskColor()` method 1114, 1133
 - `getMaximumFractionalHour()` method 1078
 - `getMaximumLength()` method 1105
 - `getMaximumValue()` method 1074, 1079, 1096

- getMinimumFractionalHour() method 1079
- getMinimumLength() method 1106
- getMinimumValue() method 1074, 1079, 1096
- getMunged() method 1122
- getRed() method 1120
- getRowCount() method 1084
- getSatellites() method 1117
- getValid() method 1122
- getValue() method 1059, 1065, 1069, 1072, 1076, 1094, 1098, 1099, 1103, 1108, 1112, 1128, 1131
- getView() method 1034
- hasAction() method 1063
- HourMin variable 1127
- HourMinSec variable 1127
- imageChanged() method 1025
- ImagePosition_Center variable 1129
- ImagePosition_LowerLeft variable 1129
- ImagePosition_LowerMiddle variable 1129
- ImagePosition_LowerRight variable 1129
- ImagePosition_MiddleLeft variable 1129
- ImagePosition_MiddleRight variable 1129
- ImagePosition_Unknown variable 1129
- ImagePosition_UpperLeft variable 1130
- ImagePosition_UpperMiddle variable 1130
- ImagePosition_UpperRight variable 1130
- ImagePresentation_CropToFit variable 1131
- ImagePresentation_FullSize variable 1131
- ImagePresentation_LockAspectRatio variable 1131
- ImagePresentation_StretchToFit variable 1131
- ImagePresentation_Unknown variable 1131
- imageSelectionChanged() method 1025
- ImageType_Bitmap variable 1112
- ImageType_GIF variable 1112
- ImageType_JPEG variable 1112
- ImageType_PNG variable 1112
- ImageType_Unknown variable 1112
- initialize(BooleanDisplayModel, Context) method 998
- initialize(BooleanEditModel, Context) method 1000
- initialize(ButtonDisplayModel, Context) method 1003
- initialize(DateAndTimeDisplayModel, Context) method 1005
- initialize(DateAndTimeEditModel, Context) method 1007
- initialize(DateDisplayModel, Context) method 1010
- initialize(DateEditModel, Context) method 1012
- initialize(DecimalDisplayModel, Context) method 1014
- initialize(DecimalEditModel, Context) method 1017
- initialize(DurationDisplayModel, Context) method 1020
- initialize(DurationEditModel, Context) method 1023
- initialize(EmbeddedImageDisplayModel, Context) method 1025
- initialize(ExternalDataDisplayModel, Context) method 1027
- initialize(ExternalDataEditModel, Context) method 1030
- initialize(IntegerDisplayModel, Context) method 1038
- initialize(IntegerEditModel, Context) method 1041
- initialize(LabelDisplayModel, Context) method 1043
- initialize(LocationDisplayModel, Context) method 1045
- initialize(LocationEditModel, Context) method 1048
- initialize(StringDisplayModel, Context) method 1050
- initialize(StringEditModel, Context) method 1052
- initialize(TimeDisplayModel, Context) method 1055
- initialize(TimeEditModel, Context) method 1057
- isAgentryDisplayingLabel() method 1034
- isAgentryDisplayingValidationFailure() method 1034
- isAutosizeSupported() method 1091
- isButtonSelected() method 1063
- isCarriageReturnAllowed() method 1103
- isEnabled() method 1091
- isHidden() method 1092
- isHyperlinkEnabled() method 1092
- isImageCellSelected(long, long) method 1084
- isPasswordInput() method 1106

- isValid() method 1114, 1117, 1121, 1133
- isWrapAllowed() method 1103
- launchActivity(Intent, int) method 1092
- MaskColor(int, int, int) constructor 1120
- MaskColor(short, short, short) constructor 1119
- MinSec variable 1127
- needsBitmapData() method 1115
- onActivityResult(int, int, Intent) method 1035
- processDecimalInput(double) method 1080
- processInput() method 1064
- processInput(AgentryLocation) method 1101
- processInput(boolean) method 1061
- processInput(double) method 1074
- processInput(GregorianCalendar) method 1067, 1070, 1110
- processInput(int) method 1080
- processInput(String) method 1087, 1106
- ProcessInputReturn(boolean, boolean, boolean) constructor 1122
- processIntegerInput(int) method 1096
- requestLayoutHeight(int) method 1092
- selectedStateChanged(boolean) method 1003
- setBitmapData(byte[]) method 1115
- setDilution(double) method 1118
- setEnabled(boolean) method 1035
- setHyperlinkEnabled(boolean) method 1035
- setImageCellSelected(long, long) method 1084
- setLatitude(double) method 1118
- setLongitude(double) method 1118
- setSatellites(int) method 1118
- setValid(boolean, String) method 1036
- setValid(boolean) method 1118
- setVisible(boolean) method 1036
- updateLabel(String) method 1036
- valueChanged(AgentryLocation) method 1046, 1048
- valueChanged(boolean) method 998, 1001
- valueChanged(double) method 1015, 1017
- valueChanged(GregorianCalendar) method 1005, 1008, 1010, 1012, 1055, 1057
- valueChanged(int) method 1020, 1023, 1039, 1041
- valueChanged(String) method 1028, 1030, 1043, 1050, 1053
- [Agentry Open UI Windows SDK API]
 - AcceptReturn property 1244
 - AgentryDataType enumeration 1256
 - AgentryPropertyType enumeration 1256
 - Ancestor property 1247
 - ClientDisplaysLabel property 1255
 - ClientDisplaysValidationError property 1255
 - Collections() method 1246
 - Columns property 1232
 - DataType property 1248
 - DateValue property 1214
 - Descendant(int) method 1246
 - DescendantCount property 1248
 - DisplayedItemAt(int) method 1208
 - DisplayedItemCount property 1209
 - DisplayName property 1248
 - DoesAgentryActionExist(string) method 1203
 - DurationFormat property 1225
 - DurationValue property 1225
 - ExecuteAgentryAction(string) method 1203
 - ExecuteHyperlinkAction() method 1204
 - FilePath property 1229
 - GetAgentryString(string) method 1204
 - GetExtensionString(string) method 1255
 - Image property 1232
 - InternalName property 1248
 - IsAgentryActionEnabled(string) method 1204
 - IsAutoSize property 1205
 - IsEnabled property 1206
 - IsHyperlinkEnabled property 1206
 - IsPassword property 1244
 - IsSelected(int, int) method 1232
 - IsVisible property 1206
 - Label property 1206
 - Maximum property 1237
 - MaximumLength property 1245
 - MaximumValue property 1223
 - Minimum property 1237
 - MinimumLength property 1245
 - MinimumValue property 1223
 - NumberValue property 1239
 - Objects() method 1247
 - OnPropertyChanged(string) method 1205
 - ProcessInput(DateTime) method 1211
 - ProcessInput(double) method 1217
 - ProcessInput(string) method 1227, 1244
 - ProcessInput(T) method 1237
 - ProcessInput(TimeSpan) method 1223
 - Properties() method 1247
 - PropertyType property 1254

- Root property 1249
- Rows property 1233
- SelectCell(int, int) method 1232
- SelectColor property 1233
- SelectedItem property 1209
- SelectItem(int) method 1209
- SMPActionResult enumeration 1256
- SMPActionState enumeration 1257
- SMPDurationFormat enumeration 1257
- SMPProcessInputReturn enumeration 1257
- StringValue property 1241
- TimeValue property 1214
- ToBoolean() method 1251
- ToDate() method 1252
- ToDateTime() method 1252
- ToDouble() method 1252
- ToInt() method 1253
- ToString() method 1253
- ToTime() method 1253
- ToUInt() method 1253
- Value property 1214
- WordWrap property 1241

A

- AcceptReturn property
 - [Agentry Open UI Windows SDK API] 1244
- Action_BackUp variable
 - [Agentry Open UI Android SDK API] 1124
- Action_Cancel variable
 - [Agentry Open UI Android SDK API] 1124
- Action_Complete variable
 - [Agentry Open UI Android SDK API] 1124
- Action_Error variable
 - [Agentry Open UI Android SDK API] 1125
- Action_Pending variable
 - [Agentry Open UI Android SDK API] 1125
- ActionDisable variable
 - [Agentry Open UI Android SDK API] 1123
- ActionEnable variable
 - [Agentry Open UI Android SDK API] 1123
- ActionEnableType enum [Agentry Open UI
 - Android SDK API]
 - description 1123
- ActionError variable
 - [Agentry Open UI Android SDK API] 1123
- ActionNoOperation variable
 - [Agentry Open UI Android SDK API] 1123
- ActionResult enum [Agentry Open UI Android
 - SDK API]
 - description 1124

- adapters package [Agentry Open UI Android SDK
 - API]
 - description 996
- Agentry Java System Connection API API
 - AgentryAppender class 661
 - AgentryException class 668
 - AgentryHandler class 647
 - AgentryJavaLoggingConfigurator class 651
 - AgentryJavaLoggingConfigurator.ReallySimpleFormatter class 651
 - BusinessLogicException class 671
 - com.syclo.agentry package 647
 - ComplexTable< CTOBJ > class 675
 - ComplexTableSession class 673
 - DataTable< DTOBJ extends DataTableObject
 - > class 690
 - DataTableMapIterator< K, V > class 664
 - DataTableObject class 686
 - DataTableSession class 688
 - FatalTransactionException class 694
 - FatalTransactionExceptionStop class 697
 - FetchSession class 699
 - java_logging package 647
 - log4j package 661
 - Logger class 665
 - LoginBlockedException class 704
 - LoginException class 706
 - LoginSkippedException class 709
 - PasswordExpiredCannotChangeException
 - class 711
 - PasswordExpiredException class 713
 - PasswordInvalidException class 716
 - PasswordWarningCannotChangeException
 - class 718
 - PasswordWarningException class 720
 - PushSession class 723
 - PushUserSession class 729
 - RetryTransactionException class 735
 - RetryTransactionWithChangeException class
 - 737
 - Server class 739
 - Server.LoginEnumeration enum 746
 - Server.LoginFailureReason enum 749
 - ServiceEvent class 768
 - ServiceEventSession class 770
 - Session class 775
 - SessionData interface 805
 - Steplet class 778
 - StepletAbortException class 786

- StepletStopException class 787
- SycloCalendar class 788
- TransactionSession class 793
- User class 795
- User.ChangePasswordResult enum 797
- UserLogger class 659
- UserLogRecord class 652
- utility package 647
- Agentry Open UI Android SDK API
 - ActionEnableType enum 1123
 - ActionResult enum 1124
 - adapters package 996
 - AgentryImage class 1110
 - AgentryImage.ImageType enum 1111
 - AgentryLocation class 1115
 - AutosizeBehavior enum 1125
 - BooleanDisplayAdapter class 996
 - BooleanDisplayModel interface 1058
 - BooleanEditAdapter class 999
 - BooleanEditModel interface 1059
 - ButtonDisplayAdapter class 1001
 - ButtonDisplayModel interface 1061
 - ButtonType enum 1126
 - client package 996
 - com.sap.mobile.platform package 996
 - core package 1110
 - DateAndTimeDisplayAdapter class 1003
 - DateAndTimeDisplayModel interface 1064
 - DateAndTimeEditAdapter class 1006
 - DateAndTimeEditModel interface 1065
 - DateDisplayAdapter class 1008
 - DateDisplayModel interface 1067
 - DateEditAdapter class 1010
 - DateEditModel interface 1069
 - DecimalDisplayAdapter class 1013
 - DecimalDisplayModel interface 1071
 - DecimalEditAdapter class 1015
 - DecimalEditModel interface 1072
 - DurationDisplayAdapter class 1017
 - DurationDisplayFormat enum 1127
 - DurationDisplayModel interface 1074
 - DurationEditAdapter class 1020
 - DurationEditModel interface 1077
 - EmbeddedImageDisplayAdapter class 1023
 - EmbeddedImageDisplayModel interface 1080
 - ExternalDataDisplayAdapter class 1026
 - ExternalDataDisplayModel interface 1084
 - ExternalDataEditAdapter class 1028
 - ExternalDataEditModel interface 1086
 - FieldAdapter class 1030
 - FieldModel interface 1088
 - ImagePosition enum 1128
 - ImagePresentation enum 1130
 - IntegerDisplayAdapter class 1037
 - IntegerDisplayModel interface 1093
 - IntegerEditAdapter class 1039
 - IntegerEditModel interface 1094
 - LabelDisplayAdapter class 1041
 - LabelDisplayModel interface 1097
 - LocationDisplayAdapter class 1044
 - LocationDisplayModel interface 1098
 - LocationEditAdapter class 1046
 - LocationEditModel interface 1100
 - MaskColor class 1119
 - models package 1057
 - openui package 996, 1110
 - OpenUIImage interface 1132
 - ProcessInputReturn class 1121
 - StringDisplayAdapter class 1048
 - StringDisplayModel interface 1101
 - StringEditAdapter class 1051
 - StringEditModel interface 1104
 - TimeDisplayAdapter class 1053
 - TimeDisplayModel interface 1107
 - TimeEditAdapter class 1055
 - TimeEditModel interface 1108
- Agentry Open UI iOS SDK API iOSDataAPI 1133
- Agentry Open UI iOS SDK API
 - iOSDataAPIExternal 1133
- Agentry Open UI iOS SDK API iOSOpenUI 1141
- Agentry Open UI iOS SDK API
 - iOSOpenUIExternal 1142
- Agentry Open UI iOS SDK API
 - SMPDataAPILocationProtocol protocol 1134
- Agentry Open UI iOS SDK API
 - SMPDataAPIPropertyProtocol protocol 1136
- Agentry Open UI iOS SDK API
 - SMPDataAPIProtocol protocol 1138
- Agentry Open UI iOS SDK API
 - SMPOpenUIBooleanDisplayAdapter protocol 1145
- Agentry Open UI iOS SDK API
 - SMPOpenUIBooleanDisplayModel protocol 1146

Index

- Agency Open UI iOS SDK API
 SMPOpenUIBooleanEditAdapter
 protocol 1146
- Agency Open UI iOS SDK API
 SMPOpenUIBooleanEditModel protocol
 1147
- Agency Open UI iOS SDK API
 SMPOpenUIButtonDisplayAdapter
 protocol 1148
- Agency Open UI iOS SDK API
 SMPOpenUIButtonDisplayModel
 protocol 1149
- Agency Open UI iOS SDK API
 SMPOpenUICollectionDisplayAdapter
 protocol 1150
- Agency Open UI iOS SDK API
 SMPOpenUICollectionDisplayModel
 protocol 1152
- Agency Open UI iOS SDK API
 SMPOpenUIDateAndTimeDisplayAdapt
 er protocol 1154
- Agency Open UI iOS SDK API
 SMPOpenUIDateAndTimeDisplayMode
 l protocol 1154
- Agency Open UI iOS SDK API
 SMPOpenUIDateAndTimeEditAdapter
 protocol 1155
- Agency Open UI iOS SDK API
 SMPOpenUIDateAndTimeEditModel
 protocol 1156
- Agency Open UI iOS SDK API
 SMPOpenUIDateDisplayAdapter
 protocol 1156
- Agency Open UI iOS SDK API
 SMPOpenUIDateDisplayModel protocol
 1157
- Agency Open UI iOS SDK API
 SMPOpenUIDateEditAdapter protocol
 1157
- Agency Open UI iOS SDK API
 SMPOpenUIDateEditModel protocol
 1158
- Agency Open UI iOS SDK API
 SMPOpenUIDecimalDisplayAdapter
 protocol 1159
- Agency Open UI iOS SDK API
 SMPOpenUIDecimalDisplayModel
 protocol 1160
- Agency Open UI iOS SDK API
 SMPOpenUIDecimalEditAdapter
 protocol 1160
- Agency Open UI iOS SDK API
 SMPOpenUIDecimalEditModel protocol
 1161
- Agency Open UI iOS SDK API
 SMPOpenUIDurationDisplayAdapter
 protocol 1162
- Agency Open UI iOS SDK API
 SMPOpenUIDurationDisplayModel
 protocol 1163
- Agency Open UI iOS SDK API
 SMPOpenUIDurationEditAdapter
 protocol 1164
- Agency Open UI iOS SDK API
 SMPOpenUIDurationEditModel
 protocol 1165
- Agency Open UI iOS SDK API
 SMPOpenUIEmbeddedImageDisplayAd
 apter protocol 1167
- Agency Open UI iOS SDK API
 SMPOpenUIEmbeddedImageDisplayM
 odel protocol 1168
- Agency Open UI iOS SDK API
 SMPOpenUIExternalDataDisplayAdapt
 er protocol 1170
- Agency Open UI iOS SDK API
 SMPOpenUIExternalDataDisplayModel
 protocol 1170
- Agency Open UI iOS SDK API
 SMPOpenUIExternalDataEditAdapter
 protocol 1171
- Agency Open UI iOS SDK API
 SMPOpenUIExternalDataEditModel
 protocol 1172
- Agency Open UI iOS SDK API
 SMPOpenUIFieldAdapter protocol 1172
- Agency Open UI iOS SDK API
 SMPOpenUIFieldModel protocol 1178
- Agency Open UI iOS SDK API SMPOpenUIImage
 class 1142
- Agency Open UI iOS SDK API
 SMPOpenUIIntegerDisplayAdapter
 protocol 1181
- Agency Open UI iOS SDK API
 SMPOpenUIIntegerDisplayModel
 protocol 1182

- Agentry Open UI iOS SDK API
 - SMPOpenUIIntegerEditAdapter protocol 1183
- Agentry Open UI iOS SDK API
 - SMPOpenUIIntegerEditModel protocol 1183
- Agentry Open UI iOS SDK API
 - SMPOpenUILabelDisplayAdapter protocol 1184
- Agentry Open UI iOS SDK API
 - SMPOpenUILabelDisplayModel protocol 1185
- Agentry Open UI iOS SDK API
 - SMPOpenUILocation class 1143
- Agentry Open UI iOS SDK API
 - SMPOpenUILocationDisplayAdapter protocol 1186
- Agentry Open UI iOS SDK API
 - SMPOpenUILocationDisplayModel protocol 1186
- Agentry Open UI iOS SDK API
 - SMPOpenUILocationEditAdapter protocol 1187
- Agentry Open UI iOS SDK API
 - SMPOpenUILocationEditModel protocol 1188
- Agentry Open UI iOS SDK API
 - SMPOpenUIStringDisplayAdapter protocol 1188
- Agentry Open UI iOS SDK API
 - SMPOpenUIStringDisplayModel protocol 1189
- Agentry Open UI iOS SDK API
 - SMPOpenUIStringEditAdapter protocol 1190
- Agentry Open UI iOS SDK API
 - SMPOpenUIStringEditModel protocol 1191
- Agentry Open UI iOS SDK API
 - SMPOpenUITimeDisplayAdapter protocol 1192
- Agentry Open UI iOS SDK API
 - SMPOpenUITimeDisplayModel protocol 1193
- Agentry Open UI iOS SDK API
 - SMPOpenUITimeEditAdapter protocol 1193
- Agentry Open UI iOS SDK API
 - SMPOpenUITimeEditModel protocol 1194
- Agentry Open UI iOS SDK API
 - SMPOpenUIUnsignedIntegerDisplayAdapter protocol 1194
- Agentry Open UI iOS SDK API
 - SMPOpenUIUnsignedIntegerDisplayModel protocol 1195
- Agentry Open UI iOS SDK API
 - SMPOpenUIUnsignedIntegerEditAdapter protocol 1196
- Agentry Open UI iOS SDK API
 - SMPOpenUIUnsignedIntegerEditModel protocol 1196
- Agentry Open UI Windows SDK API
 - IAgencyCollection interface 1200
 - IAgencyControlViewModel interface 1201
 - IAgencyControlViewModelCollectionDisplay interface 1207
 - IAgencyControlViewModelDateTime interface 1210
 - IAgencyControlViewModelDateTimeDisplay interface 1212
 - IAgencyControlViewModelDecimal interface 1215
 - IAgencyControlViewModelDecimalDisplay interface 1218
 - IAgencyControlViewModelDuration interface 1221
 - IAgencyControlViewModelDurationDisplay interface 1223
 - IAgencyControlViewModelFile interface 1226
 - IAgencyControlViewModelFileDisplay interface 1228
 - IAgencyControlViewModelImage interface 1230
 - IAgencyControlViewModelLabel interface 1233
 - IAgencyControlViewModelNumber< T > interface 1235
 - IAgencyControlViewModelNumberDisplay< T > interface 1237
 - IAgencyControlViewModelStringDisplay interface 1239
 - IAgencyControlViewModelStringEdit interface 1242
 - IAgencyData interface 1245

- IAgencyObject interface 1249
- IAgencyProperty interface 1250
- ICustomAgencyControl interface 1254
- IEnumerable< IAgencyData > class 1255
- AGENTRY_USER_MDC_KEY variable
 - AgencyAppender class [Agency Java System Connection API API] 664
- agencyActionEnableState:
 - methodSMPOpenUIFieldModel protocol [Agency Open UI iOS SDK API] 1179
- AgencyAppender class [Agency Java System Connection API API]
 - AGENTRY_USER_MDC_KEY variable 664
 - AgencyAppender() constructor 663
 - append(LoggingEvent) method 663
 - close() method 663
 - description 661
 - mapLogLevel(Level) method 663
 - requiresLayout() method 663
- AgencyAppender() constructor
 - AgencyAppender class [Agency Java System Connection API API] 663
- AgencyDataType enumeration
 - [Agency Open UI Windows SDK API] 1256
- AgencyException class [Agency Java System Connection API API]
 - AgencyException(String, String, String, Throwable) constructor 670
 - AgencyException(String, String, String) constructor 670
 - AgencyException(String, Throwable) constructor 669
 - AgencyException(String) constructor 669
 - description 668
 - getNotificationText() method 671
 - getNotificationTitle() method 671
 - getOkButtonLabel() method 671
- AgencyException(String, String, String, Throwable) constructor
 - AgencyException class [Agency Java System Connection API API] 670
- AgencyException(String, String, String) constructor
 - AgencyException class [Agency Java System Connection API API] 670
- AgencyException(String, Throwable) constructor
 - AgencyException class [Agency Java System Connection API API] 669
- AgencyException(String) constructor
 - AgencyException class [Agency Java System Connection API API] 669
- AgencyExceptionHandler constructor
 - AgencyExceptionHandler class [Agency Java System Connection API API] 669
- AgencyHandler class [Agency Java System Connection API API]
 - _defaultFormatter variable 650
 - AgencyHandler() constructor 649
 - close() method 650
 - description 647
 - flush() method 650
 - mapLogLevel(Level) method 650
 - publish(LogRecord) method 650
- AgencyHandler() constructor
 - AgencyHandler class [Agency Java System Connection API API] 649
- AgencyImage class [Agency Open UI Android SDK API]
 - description 1110
- AgencyImage.ImageType enum [Agency Open UI Android SDK API]
 - description 1111
- AgencyImage(String, ImageType, ImagePresentation, ImagePosition, int, int, int) constructor
 - [Agency Open UI Android SDK API] 1113
- AgencyJavaLoggingConfigurator class [Agency Java System Connection API API]
 - AgencyJavaLoggingConfigurator() constructor 652
 - description 651
- AgencyJavaLoggingConfigurator.ReallySimpleFormatter class [Agency Java System Connection API API]
 - description 651
 - format(LogRecord) method 652
- AgencyJavaLoggingConfigurator() constructor
 - AgencyJavaLoggingConfigurator class [Agency Java System Connection API API] 652
- AgencyLocation class [Agency Open UI Android SDK API]
 - description 1115
- AgencyLocation(boolean, double, double, int, double) constructor
 - [Agency Open UI Android SDK API] 1116
- AgencyPropertyType enumeration
 - [Agency Open UI Windows SDK API] 1256
- agencyShouldDisplayLabel
 - methodSMPOpenUIFieldAdapter

- protocol [Agentry Open UI iOS SDK API] 1173
- agentryShouldDisplayValidationFailure
 - methodSMPOpenUIFieldAdapter
 - protocol [Agentry Open UI iOS SDK API] 1174
- agentryString: methodSMPOpenUIFieldModel
 - protocol [Agentry Open UI iOS SDK API] 1179
- allObjectsChanged:
 - methodSMPOpenUICollectionDisplayA
 - dapter protocol [Agentry Open UI iOS SDK API] 1150
- allowsCarriageReturn
 - propertySMPOpenUIStringDisplayMod
 - el protocol [Agentry Open UI iOS SDK API] 1189
- ancestor methodSMPDataAPIProtocol protocol
 - [Agentry Open UI iOS SDK API] 1139
- Ancestor property
 - [Agentry Open UI Windows SDK API] 1247
- append(LoggingEvent) method
 - AgentryAppender class [Agentry Java System
 - Connection API API] 663
- appendDebug(String) method
 - Logger class [deprecated] [Agentry Java
 - System Connection API API] 666
- asBool methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1136
- asDate methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1136
- asDateAndTime
 - methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1136
- asDecimal methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1137
- asLocation methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1137
- asLong methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1137
- asString methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1137

- asTime methodSMPDataAPIPropertyProtocol
 - protocol [Agentry Open UI iOS SDK API] 1138
- Autosize_FillVisible variable
 - [Agentry Open UI Android SDK API] 1125
- Autosize_None variable
 - [Agentry Open UI Android SDK API] 1125
- Autosize_WrapContent variable
 - [Agentry Open UI Android SDK API] 1126
- AutosizeBehavior enum [Agentry Open UI Android
 - SDK API]
 - description 1125
- autosizeBehavior
 - methodSMPOpenUIFieldAdapter
 - protocol [Agentry Open UI iOS SDK API] 1174
- autosizing propertySMPOpenUIFieldModel
 - protocol [Agentry Open UI iOS SDK API] 1180

B

- backendTimeAndDate() method [deprecated]
 - User class [Agentry Java System Connection
 - API API] 799
- beginChangePassword() method
 - User class [Agentry Java System Connection
 - API API] 800
- beginClientExchange() method
 - FetchSession class [Agentry Java System
 - Connection API API] 702
- beginDataAndUpdateSteps() method
 - ServiceEventSession class [Agentry Java
 - System Connection API API] 773
- beginDebug(String) method
 - Logger class [deprecated] [Agentry Java
 - System Connection API API] 667
- beginDisablePush() method
 - PushUserSession class [Agentry Java System
 - Connection API API] 733
- beginEnablePush() method
 - PushUserSession class [Agentry Java System
 - Connection API API] 733
- beginFetchObjectRead() method
 - FetchSession class [Agentry Java System
 - Connection API API] 702
- beginFetchRemoval() method
 - FetchSession class [Agentry Java System
 - Connection API API] 703

Index

- beginPushError() method
 - PushUserSession class [Agentry Java System Connection API API] 733
 - beginPushError() method [deprecated]
 - PushSession class [Agentry Java System Connection API API] 726
 - beginPushReadStep() method
 - PushSession class [Agentry Java System Connection API API] 727
 - beginPushRemoval() method
 - PushSession class [Agentry Java System Connection API API] 727
 - beginPushResponse() method
 - PushUserSession class [Agentry Java System Connection API API] 734
 - beginPushResponse() method [deprecated]
 - PushSession class [Agentry Java System Connection API API] 727
 - beginPushRetrieval() method
 - PushSession class [Agentry Java System Connection API API] 728
 - beginReadSteps() method
 - ServiceEventSession class [Agentry Java System Connection API API] 773
 - beginServerExchange() method
 - FetchSession class [Agentry Java System Connection API API] 703
 - beginServiceEventError() method
 - ServiceEventSession class [Agentry Java System Connection API API] 774
 - beginTransaction() method
 - TransactionSession class [Agentry Java System Connection API API] 795
 - BooleanDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 996
 - BooleanDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1058
 - BooleanEditAdapter class [Agentry Open UI Android SDK API]
 - description 999
 - BooleanEditModel interface [Agentry Open UI Android SDK API]
 - description 1059
 - build() method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 678
 - BusinessLogicException class [Agentry Java System Connection API API]
 - BusinessLogicException(String, Throwable) constructor 673
 - BusinessLogicException(String) constructor 672
 - description 671
 - BusinessLogicException(String, Throwable) constructor
 - BusinessLogicException class [Agentry Java System Connection API API] 673
 - BusinessLogicException(String) constructor
 - BusinessLogicException class [Agentry Java System Connection API API] 672
 - ButtonDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1001
 - ButtonDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1061
 - buttonImage
 - propertySMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] 1149
 - buttonImageChanged(AgentryImage) method [Agentry Open UI Android SDK API] 1003
 - ButtonStyleCheckbox variable [Agentry Open UI Android SDK API] 1126
 - ButtonStylePush variable [Agentry Open UI Android SDK API] 1126
 - ButtonStyleRadio variable [Agentry Open UI Android SDK API] 1126
 - buttonText
 - propertySMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] 1149
 - ButtonType enum [Agentry Open UI Android SDK API]
 - description 1126
 - buttonType
 - propertySMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] 1149
- ## C
- ChangePassword_Blocked variable
 - User.ChangePasswordResult enum [Agentry Java System Connection API API] 798

- ChangePassword_Failure variable
 - User.ChangePasswordResult enum [Agentry Java System Connection API API] 798
- ChangePassword_NotHandled variable
 - User.ChangePasswordResult enum [Agentry Java System Connection API API] 799
- ChangePassword_Success variable
 - User.ChangePasswordResult enum [Agentry Java System Connection API API] 799
- changePassword(String, String) method
 - User class [Agentry Java System Connection API API] 800
- changePasswordFailed(StringBuffer) method
 - User class [Agentry Java System Connection API API] 801
- changePasswordSessionAborted() method
 - User class [Agentry Java System Connection API API] 801
- checkForReload() method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 678
- client package [Agentry Open UI Android SDK API]
 - description 996
- ClientDisplaysLabel property
 - [Agentry Open UI Windows SDK API] 1255
- ClientDisplaysValidationError property
 - [Agentry Open UI Windows SDK API] 1255
- close() method
 - AgentryAppender class [Agentry Java System Connection API API] 663
 - AgentryHandler class [Agentry Java System Connection API API] 650
- code variable
 - DataTableObject class [Agentry Java System Connection API API] 688
- code() method [deprecated]
 - DataTableObject class [Agentry Java System Connection API API] 687
- collection
 - methodSMPOpenUICollectionDisplayModel protocol [Agentry Open UI iOS SDK API] 1152
- Collections() method
 - [Agentry Open UI Windows SDK API] 1246
- Columns property
 - [Agentry Open UI Windows SDK API] 1232
- columns
 - propertySMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] 1169
- com.sap.mobile.platform package [Agentry Open UI Android SDK API]
 - description 996
- com.syclo.agentry package [Agentry Java System Connection API API]
 - description 647
- ComplexTable(ComplexTableSession, GregorianCalendar) method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 679
- ComplexTable< CTOBJ > class [Agentry Java System Connection API API]
 - _clientLastDataUpdateTime variable 685
 - _rebuilding variable 686
 - _session variable 686
 - build() method 678
 - checkForReload() method 678
 - ComplexTable(ComplexTableSession, GregorianCalendar) method 679
 - dataIterator() method 679
 - deleteIterator() method 680
 - description 675
 - getClientLastDataUpdateTime() method 681
 - getNewDataUpdateTime() method 681
 - getSession() method 681
 - initialize() method [deprecated] 681
 - isRebuilding() method 682
 - lastUpdateDate() method 682
 - lastUpdateHours() method 682
 - lastUpdateMinutes() method 682
 - lastUpdateMonth() method 683
 - lastUpdateSeconds() method 683
 - lastUpdateYear() method 683
 - reload() method [deprecated] 684
 - setNewDataUpdateTime(GregorianCalendar) method 684
 - willRebuildTable() method 685
- ComplexTableSession class [Agentry Java System Connection API API]
 - ComplexTableSession(String, Server, SessionData, User) constructor 674
 - description 673

Index

ComplexTableSession(String, Server, SessionData, User) constructor
ComplexTableSession class [Agentry Java System Connection API API] 674
core package [Agentry Open UI Android SDK API] description 1110
createComplexTableSession(String, SessionData, User) method
Server class [Agentry Java System Connection API API] 750
createDataTableSession(String, SessionData, User) method
Server class [Agentry Java System Connection API API] 751
createFetchSession(String, Server, SessionData, User) method [deprecated]
Server class [Agentry Java System Connection API API] 752
createFetchSession(String, SessionData, User) method
Server class [Agentry Java System Connection API API] 752
createPushSession(String, Server, SessionData) method [deprecated]
Server class [Agentry Java System Connection API API] 753
createPushSession(String, SessionData) method
Server class [Agentry Java System Connection API API] 753
createPushUserSession(String, Server, SessionData, User) method [deprecated]
Server class [Agentry Java System Connection API API] 754
createPushUserSession(String, SessionData, User) method
Server class [Agentry Java System Connection API API] 754
createServiceEventSession(String, Server, SessionData) method [deprecated]
Server class [Agentry Java System Connection API API] 755
createServiceEventSession(String, SessionData) method
Server class [Agentry Java System Connection API API] 756
createTransactionSession(String, Server, SessionData, User) method [deprecated]
Server class [Agentry Java System Connection API API] 756

createTransactionSession(String, SessionData, User) method
Server class [Agentry Java System Connection API API] 757
createUser(String, int) method [deprecated]
Server class [Agentry Java System Connection API API] 757
createUser(String) method
Server class [Agentry Java System Connection API API] 758

D

dataIdentifier methodSMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] 1139
dataIterator() method
ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 679
dataReceived(Object) method
ServiceEvent class [Agentry Java System Connection API API] 769
DataTable(DataTableSession, GregorianCalendar) method
DataTable< DTOBJ extends DataTableObject > class [Agentry Java System Connection API API] 692
DataTable< DTOBJ extends DataTableObject > class [Agentry Java System Connection API API]
_clientLastDataUpdateTime variable 694
_session variable 694
DataTable(DataTableSession, GregorianCalendar) method 692
description 690
getClientLastDataUpdateTime() method 692
getSession() method 692
initialize() method 693
isOutOfDate() method 693
iterator() method 693
DataTableMapIterator(Map< K, V >) method
DataTableMapIterator< K, V > class [Agentry Java System Connection API API] 664
DataTableMapIterator< K, V > class [Agentry Java System Connection API API]
DataTableMapIterator(Map< K, V >) method 664
description 664
hasNext() method 665

- next() method 665
- remove() method 665
- DataTableObject class [Agentry Java System Connection API API]
 - code variable 688
 - code() method [deprecated] 687
 - DataTableObject(String, String) constructor 687
 - description 686
 - equals(Object) method 687
 - getKey() method 687
 - getValue() method 688
 - hashCode() method 688
 - value variable 688
 - value() method [deprecated] 688
- DataTableObject(String, String) constructor
 - DataTableObject class [Agentry Java System Connection API API] 687
- DataTableSession class [Agentry Java System Connection API API]
 - DataTableSession(String, Server, SessionData, User) constructor 690
 - description 688
- DataTableSession(String, Server, SessionData, User) constructor
 - DataTableSession class [Agentry Java System Connection API API] 690
- dataType methodSMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] 1139
- DataType property [Agentry Open UI Windows SDK API] 1248
- DateAndTimeDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1003
- DateAndTimeDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1064
- DateAndTimeEditAdapter class [Agentry Open UI Android SDK API]
 - description 1006
- DateAndTimeEditModel interface [Agentry Open UI Android SDK API]
 - description 1065
- DateDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1008
- DateDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1067
- DateEditAdapter class [Agentry Open UI Android SDK API]
 - description 1010
- DateEditModel interface [Agentry Open UI Android SDK API]
 - description 1069
- DateValue property [Agentry Open UI Windows SDK API] 1214
- debug(String, Map< String, String >, String) method
 - Logger class [deprecated] [Agentry Java System Connection API API] 667
- debug(String) method
 - Logger class [deprecated] [Agentry Java System Connection API API] 667
- Server class [Agentry Java System Connection API API] 758
- Session class [Agentry Java System Connection API API] 777
- User class [Agentry Java System Connection API API] 801
- DecHour variable [Agentry Open UI Android SDK API] 1127
- DecimalDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1013
- DecimalDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1071
- DecimalEditAdapter class [Agentry Open UI Android SDK API]
 - description 1015
- DecimalEditModel interface [Agentry Open UI Android SDK API]
 - description 1072
- decryptPassword(String) method
 - Server class [Agentry Java System Connection API API] 759
- deleteIterator() method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 680
- descendant: methodSMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] 1139
- Descendant(int) method [Agentry Open UI Windows SDK API] 1246
- descendantCount methodSMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] 1140

Index

DescendantCount property
 [Agentry Open UI Windows SDK API] 1248

dilution propertySMPDataAPILocationProtocol
 protocol [Agentry Open UI iOS SDK
 API] 1135

dilution propertySMPOpenUILocation class
 [Agentry Open UI iOS SDK API] 1144

disablePush() method
 PushUserSession class [Agentry Java System
 Connection API API] 734

DisplayedItemAt(int) method
 [Agentry Open UI Windows SDK API] 1208

DisplayedItemCount property
 [Agentry Open UI Windows SDK API] 1209

displayedObjectAtIndex:
 methodSMPOpenUICollectionDisplayM
 odel protocol [Agentry Open UI iOS SDK
 API] 1153

displayedObjectCount
 propertySMPOpenUICollectionDisplay
 Model protocol [Agentry Open UI iOS
 SDK API] 1153

displayFormat
 propertySMPOpenUIDurationDisplayM
 odel protocol [Agentry Open UI iOS SDK
 API] 1163

displayName methodSMPDataAPIProtocol
 protocol [Agentry Open UI iOS SDK
 API] 1140

DisplayName property
 [Agentry Open UI Windows SDK API] 1248

DoesAgentryActionExist(string) method
 [Agentry Open UI Windows SDK API] 1203

doSteplet() method
 Steplet class [Agentry Java System Connection
 API API] 782

DurationDisplayAdapter class [Agentry Open UI
 Android SDK API]
 description 1017

DurationDisplayFormat enum [Agentry Open UI
 Android SDK API]
 description 1127

DurationDisplayModel interface [Agentry Open UI
 Android SDK API]
 description 1074

DurationEditAdapter class [Agentry Open UI
 Android SDK API]
 description 1020

DurationEditModel interface [Agentry Open UI
 Android SDK API]
 description 1077

DurationFormat property
 [Agentry Open UI Windows SDK API] 1225

DurationValue property
 [Agentry Open UI Windows SDK API] 1225

E

EmbeddedImageDisplayAdapter class [Agentry
 Open UI Android SDK API]
 description 1023

EmbeddedImageDisplayModel interface [Agentry
 Open UI Android SDK API]
 description 1080

enabled propertySMPOpenUIFieldModel protocol
 [Agentry Open UI iOS SDK API] 1181

enablePush() method
 PushUserSession class [Agentry Java System
 Connection API API] 734

endChangePassword() method
 User class [Agentry Java System Connection
 API API] 802

endClientExchange() method
 FetchSession class [Agentry Java System
 Connection API API] 703

endDataAndUpdateSteps() method
 ServiceEventSession class [Agentry Java
 System Connection API API] 774

endDebug(String) method
 Logger class [deprecated] [Agentry Java
 System Connection API API] 667

endDisablePush() method
 PushUserSession class [Agentry Java System
 Connection API API] 734

endEnablePush() method
 PushUserSession class [Agentry Java System
 Connection API API] 735

endFetchObjectRead() method
 FetchSession class [Agentry Java System
 Connection API API] 703

endFetchRemoval() method
 FetchSession class [Agentry Java System
 Connection API API] 704

endPushError() method
 PushUserSession class [Agentry Java System
 Connection API API] 735

endPushError() method [deprecated]
 PushSession class [Agentry Java System
 Connection API API] 728
 endPushReadStep() method
 PushSession class [Agentry Java System
 Connection API API] 728
 endPushRemoval() method
 PushSession class [Agentry Java System
 Connection API API] 729
 endPushResponse() method
 PushUserSession class [Agentry Java System
 Connection API API] 735
 endPushResponse() method [deprecated]
 PushSession class [Agentry Java System
 Connection API API] 729
 endPushRetrieval() method
 PushSession class [Agentry Java System
 Connection API API] 729
 endReadSteps() method
 ServiceEventSession class [Agentry Java
 System Connection API API] 774
 endServerExchange() method
 FetchSession class [Agentry Java System
 Connection API API] 704
 endServiceEventError() method
 ServiceEventSession class [Agentry Java
 System Connection API API] 774
 endTransaction() method
 TransactionSession class [Agentry Java
 System Connection API API] 795
 equals(Object) method
 DataTableObject class [Agentry Java System
 Connection API API] 687
 UserLogRecord class [Agentry Java System
 Connection API API] 655
 eval(String) method
 SessionData interface [Agentry Java System
 Connection API API] 808
 executeAgentryAction:
 methodSMPOpenUIFieldModel protocol
 [Agentry Open UI iOS SDK API] 1179
 executeAgentryAction(String) method
 [Agentry Open UI Android SDK API] 1089
 ExecuteAgentryAction(string) method
 [Agentry Open UI Windows SDK API] 1203
 executeHyperlinkAction
 methodSMPOpenUIFieldModel protocol
 [Agentry Open UI iOS SDK API] 1180

executeHyperlinkAction() method
 [Agentry Open UI Android SDK API] 1090
 ExecuteHyperlinkAction() method
 [Agentry Open UI Windows SDK API] 1204
 ExternalDataDisplayAdapter class [Agentry Open
 UI Android SDK API]
 description 1026
 ExternalDataDisplayModel interface [Agentry
 Open UI Android SDK API]
 description 1084
 ExternalDataEditAdapter class [Agentry Open UI
 Android SDK API]
 description 1028
 ExternalDataEditModel interface [Agentry Open UI
 Android SDK API]
 description 1086

F

FatalTransactionException class [Agentry Java
 System Connection API API]
 description 694
 FatalTransactionException(String, String,
 String, Throwable) constructor 697
 FatalTransactionException(String, String,
 String) constructor 696
 FatalTransactionException(String,
 Throwable) constructor 696
 FatalTransactionException(String)
 constructor 696
 FatalTransactionException(String, String, String,
 Throwable) constructor
 FatalTransactionException class [Agentry Java
 System Connection API API] 697
 FatalTransactionException(String, String, String)
 constructor
 FatalTransactionException class [Agentry Java
 System Connection API API] 696
 FatalTransactionException(String, Throwable)
 constructor
 FatalTransactionException class [Agentry Java
 System Connection API API] 696
 FatalTransactionException(String) constructor
 FatalTransactionException class [Agentry Java
 System Connection API API] 696
 FatalTransactionExceptionStop class [Agentry Java
 System Connection API API]
 description 697
 FatalTransactionExceptionStop(String, String,
 String, Throwable) constructor 699

FatalTransactionExceptionStop(String, String, String) constructor 698
 FatalTransactionExceptionStop(String, String, String, Throwable) constructor
 FatalTransactionExceptionStop class [Agentry Java System Connection API API] 699
 FatalTransactionExceptionStop(String, String, String) constructor
 FatalTransactionExceptionStop class [Agentry Java System Connection API API] 698
 FetchSession class [Agentry Java System Connection API API]
 beginClientExchange() method 702
 beginFetchObjectRead() method 702
 beginFetchRemoval() method 703
 beginServerExchange() method 703
 description 699
 endClientExchange() method 703
 endFetchObjectRead() method 703
 endFetchRemoval() method 704
 endServerExchange() method 704
 FetchSession(String, Server, SessionData, User) constructor 702
 FetchSession(String, Server, SessionData, User) constructor
 FetchSession class [Agentry Java System Connection API API] 702
 FieldAdapter class [Agentry Open UI Android SDK API]
 description 1030
 FieldModel interface [Agentry Open UI Android SDK API]
 description 1088
 FilePath property
 [Agentry Open UI Windows SDK API] 1229
 findConfigurationFile(String) method
 Server class [Agentry Java System Connection API API] 759
 flush() method
 AgentryHandler class [Agentry Java System Connection API API] 650
 format(LogRecord) method
 AgentryJavaLoggingConfigurator.ReallySimpleFormatter class [Agentry Java System Connection API API] 652
 fractionalHourValue
 propertySMPOpenUIDurationDisplayM

 odel protocol [Agentry Open UI iOS SDK API] 1163
 fractionalHourValueChanged(double) method
 [Agentry Open UI Android SDK API] 1019, 1022

G

getAgentryActionEnableState(String) method
 [Agentry Open UI Android SDK API] 1090
 getAgentryString(String) method
 [Agentry Open UI Android SDK API] 1090
 GetAgentryString(string) method
 [Agentry Open UI Windows SDK API] 1204
 getAutosizeBehavior() method
 [Agentry Open UI Android SDK API] 1032
 getBitmapData() method
 [Agentry Open UI Android SDK API] 1113, 1132
 getBlue() method
 [Agentry Open UI Android SDK API] 1120
 getBoolean(String) method
 SessionData interface [Agentry Java System Connection API API] 808
 getButtonImage() method
 [Agentry Open UI Android SDK API] 1062
 getButtonText() method
 [Agentry Open UI Android SDK API] 1063
 getButtonType() method
 [Agentry Open UI Android SDK API] 1063
 getBytes(String) method
 SessionData interface [Agentry Java System Connection API API] 808
 getChanged() method
 [Agentry Open UI Android SDK API] 1122
 getClientLastDataUpdateTime() method
 ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 681
 DataTable< DTOBJ extends DataTableObject > class [Agentry Java System Connection API API] 692
 getColumnCount() method
 [Agentry Open UI Android SDK API] 1082
 getContentHeightForAutosizing(int) method
 [Agentry Open UI Android SDK API] 1033
 getDilution() method
 [Agentry Open UI Android SDK API] 1116
 getDouble(String) method
 SessionData interface [Agentry Java System Connection API API] 809

- getDurationDisplayFormat() method
 - [Agentry Open UI Android SDK API] 1076
- getExtensionString(String) method
 - [Agentry Open UI Android SDK API] 1033
- GetExtensionString(string) method
 - [Agentry Open UI Windows SDK API] 1255
- getFilePath() method
 - [Agentry Open UI Android SDK API] 1086
- getFloat(String) method
 - SessionData interface [Agentry Java System Connection API API] 809
- getFractionalHourValue() method
 - [Agentry Open UI Android SDK API] 1076
- getGreen() method
 - [Agentry Open UI Android SDK API] 1120
- getHighlightColor() method
 - [Agentry Open UI Android SDK API] 1082
- getImage() method
 - [Agentry Open UI Android SDK API] 1083
- getImageName() method
 - [Agentry Open UI Android SDK API] 1113, 1132
- getImagePosition() method
 - [Agentry Open UI Android SDK API] 1083, 1114
- getImagePresentation() method
 - [Agentry Open UI Android SDK API] 1083, 1114
- getImageType() method
 - [Agentry Open UI Android SDK API] 1114, 1133
- getImplementationVersion() method
 - Server class [Agentry Java System Connection API API] 759
- getInstance() method
 - Server class [Agentry Java System Connection API API] 760
- getInteger(String) method
 - SessionData interface [Agentry Java System Connection API API] 809
- getInvalidTimeAndDate() method
 - SycloCalendar class [Agentry Java System Connection API API] 792
- getKey() method
 - DataTableObject class [Agentry Java System Connection API API] 687
- getLabel() method
 - [Agentry Open UI Android SDK API] 1091
- getLatitude() method
 - [Agentry Open UI Android SDK API] 1117
- getLevel() method
 - UserLogRecord class [Agentry Java System Connection API API] 655
- getLoggerName() method
 - UserLogRecord class [Agentry Java System Connection API API] 655
- getLong(String) method
 - SessionData interface [Agentry Java System Connection API API] 809
- getLongitude() method
 - [Agentry Open UI Android SDK API] 1117
- getMaskColor() method
 - [Agentry Open UI Android SDK API] 1114, 1133
- getMaximumFractionalHour() method
 - [Agentry Open UI Android SDK API] 1078
- getMaximumLength() method
 - [Agentry Open UI Android SDK API] 1105
- getMaximumValue() method
 - [Agentry Open UI Android SDK API] 1074, 1079, 1096
- getMessage() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getMillis() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getMinimumFractionalHour() method
 - [Agentry Open UI Android SDK API] 1079
- getMinimumLength() method
 - [Agentry Open UI Android SDK API] 1106
- getMinimumValue() method
 - [Agentry Open UI Android SDK API] 1074, 1079, 1096
- getMunged() method
 - [Agentry Open UI Android SDK API] 1122
- getName() method
 - Session class [Agentry Java System Connection API API] 777
 - User class [Agentry Java System Connection API API] 802
- getNewDataUpdateTime() method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 681
- getNotificationText() method
 - AgentryException class [Agentry Java System Connection API API] 671

Index

- Steplet class [Agentry Java System Connection API API] 783
- getNotificationTitle() method
 - AgentryException class [Agentry Java System Connection API API] 671
- Steplet class [Agentry Java System Connection API API] 783
- getOkButtonLabel() method
 - AgentryException class [Agentry Java System Connection API API] 671
- Steplet class [Agentry Java System Connection API API] 783
- getParameters() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getRed() method
 - [Agentry Open UI Android SDK API] 1120
- getResourceBundle() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getResourceBundleName() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getReturnData() method
 - Steplet class [Agentry Java System Connection API API] 784
- getRowCount() method
 - [Agentry Open UI Android SDK API] 1084
- getSatellites() method
 - [Agentry Open UI Android SDK API] 1117
- getSequenceNumber() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getServer() method
 - Session class [Agentry Java System Connection API API] 777
- getSession() method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 681
 - DataTable< DTOBJ extends DataTableObject > class [Agentry Java System Connection API API] 692
 - Steplet class [Agentry Java System Connection API API] 784
- getSessionData() method
 - Session class [Agentry Java System Connection API API] 778
- getSourceClassName() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getSourceMethodName() method
 - UserLogRecord class [Agentry Java System Connection API API] 656
- getSpecificationVersion() method
 - Server class [Agentry Java System Connection API API] 760
- getString(String) method
 - SessionData interface [Agentry Java System Connection API API] 810
- getSystemConnectionTime() method
 - User class [Agentry Java System Connection API API] 802
- getThreadID() method
 - UserLogRecord class [Agentry Java System Connection API API] 657
- getThrown() method
 - UserLogRecord class [Agentry Java System Connection API API] 657
- getTimeAndDate(String, String) method
 - SessionData interface [Agentry Java System Connection API API] 810
- getTimeAndDate(String) method
 - SessionData interface [Agentry Java System Connection API API] 810
- getTimeZone() method
 - Server class [Agentry Java System Connection API API] 760
- getTimeZone(StringBuffer) method [deprecated]
 - User class [Agentry Java System Connection API API] 803
- getUser() method
 - Session class [Agentry Java System Connection API API] 778
 - UserLogger class [Agentry Java System Connection API API] 660
 - UserLogRecord class [Agentry Java System Connection API API] 657
- getUserLogger(String, String, User) method
 - UserLogger class [Agentry Java System Connection API API] 660
- getUserLogger(String, User) method
 - UserLogger class [Agentry Java System Connection API API] 660
- getValid() method
 - [Agentry Open UI Android SDK API] 1122

getValue() method
 [Agentry Open UI Android SDK API] 1059,
 1065, 1069, 1072, 1076, 1094, 1098,
 1099, 1103, 1108, 1112, 1128,
 1131
 DataTableObject class [Agentry Java System
 Connection API API] 688
 User.ChangePasswordResult enum [Agentry
 Java System Connection API API]
 798
 getView() method
 [Agentry Open UI Android SDK API] 1034

H

hasAction() method
 [Agentry Open UI Android SDK API] 1063
 hashCode() method
 DataTableObject class [Agentry Java System
 Connection API API] 688
 UserLogRecord class [Agentry Java System
 Connection API API] 657
 hasNext() method
 DataTableMapIterator< K, V > class [Agentry
 Java System Connection API API]
 665
 hidden propertySMPOpenUIFieldModel protocol
 [Agentry Open UI iOS SDK API] 1181
 highlightSelectedColor
 propertySMPOpenUIEmbeddedImageDi
 splayModel protocol [Agentry Open UI
 iOS SDK API] 1169
 HourMin variable
 [Agentry Open UI Android SDK API] 1127
 HourMinSec variable
 [Agentry Open UI Android SDK API] 1127
 hyperlinkEnabled
 propertySMPOpenUIFieldModel
 protocol [Agentry Open UI iOS SDK
 API] 1181

I

IAgentryCollection interface [Agentry Open UI
 Windows SDK API]
 description 1200
 IAgentryControlViewModel interface [Agentry
 Open UI Windows SDK API]
 description 1201

IAgentryControlViewModelCollectionDisplay
 interface [Agentry Open UI Windows
 SDK API]
 description 1207
 IAgentryControlViewModelDateTime interface
 [Agentry Open UI Windows SDK API]
 description 1210
 IAgentryControlViewModelDateTimeDisplay
 interface [Agentry Open UI Windows
 SDK API]
 description 1212
 IAgentryControlViewModelDecimal interface
 [Agentry Open UI Windows SDK API]
 description 1215
 IAgentryControlViewModelDecimalDisplay
 interface [Agentry Open UI Windows
 SDK API]
 description 1218
 IAgentryControlViewModelDuration interface
 [Agentry Open UI Windows SDK API]
 description 1221
 IAgentryControlViewModelDurationDisplay
 interface [Agentry Open UI Windows
 SDK API]
 description 1223
 IAgentryControlViewModelFile interface [Agentry
 Open UI Windows SDK API]
 description 1226
 IAgentryControlViewModelFileDisplay interface
 [Agentry Open UI Windows SDK API]
 description 1228
 IAgentryControlViewModelImage interface
 [Agentry Open UI Windows SDK API]
 description 1230
 IAgentryControlViewModelLabel interface
 [Agentry Open UI Windows SDK API]
 description 1233
 IAgentryControlViewModelNumber< T > interface
 [Agentry Open UI Windows SDK API]
 description 1235
 IAgentryControlViewModelNumberDisplay< T >
 interface [Agentry Open UI Windows
 SDK API]
 description 1237
 IAgentryControlViewModelStringDisplay
 interface [Agentry Open UI Windows
 SDK API]
 description 1239

Index

- IAgencyControlViewModelStringEdit interface
 - [Agency Open UI Windows SDK API]
 - description 1242
- IAgencyData interface [Agency Open UI Windows SDK API]
 - description 1245
- IAgencyObject interface [Agency Open UI Windows SDK API]
 - description 1249
- IAgencyProperty interface [Agency Open UI Windows SDK API]
 - description 1250
- ICustomAgencyControl interface [Agency Open UI Windows SDK API]
 - description 1254
- IEnumerable< IAgencyData > class [Agency Open UI Windows SDK API]
 - description 1255
- Image property
 - [Agency Open UI Windows SDK API] 1232
- image
 - propertySMPOpenUIEmbeddedImageDisplayModel protocol [Agency Open UI iOS SDK API] 1169
- image propertySMPOpenUIImage class [Agency Open UI iOS SDK API] 1142
- imageCellClickedAtRow:andColumn:
 - methodSMPOpenUIEmbeddedImageDisplayModel protocol [Agency Open UI iOS SDK API] 1168
- imageChanged() method
 - [Agency Open UI Android SDK API] 1025
- ImagePosition enum [Agency Open UI Android SDK API]
 - description 1128
- ImagePosition_Center variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_LowerLeft variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_LowerMiddle variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_LowerRight variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_MiddleLeft variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_MiddleRight variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_Unknown variable
 - [Agency Open UI Android SDK API] 1129
- ImagePosition_UpperLeft variable
 - [Agency Open UI Android SDK API] 1130
- ImagePosition_UpperMiddle variable
 - [Agency Open UI Android SDK API] 1130
- ImagePosition_UpperRight variable
 - [Agency Open UI Android SDK API] 1130
- ImagePresentation enum [Agency Open UI Android SDK API]
 - description 1130
- ImagePresentation_CropToFit variable
 - [Agency Open UI Android SDK API] 1131
- ImagePresentation_FullSize variable
 - [Agency Open UI Android SDK API] 1131
- ImagePresentation_LockAspectRatio variable
 - [Agency Open UI Android SDK API] 1131
- ImagePresentation_StretchToFit variable
 - [Agency Open UI Android SDK API] 1131
- ImagePresentation_Unknown variable
 - [Agency Open UI Android SDK API] 1131
- imageSelectionChanged() method
 - [Agency Open UI Android SDK API] 1025
- ImageType_Bitmap variable
 - [Agency Open UI Android SDK API] 1112
- ImageType_GIF variable
 - [Agency Open UI Android SDK API] 1112
- ImageType_JPEG variable
 - [Agency Open UI Android SDK API] 1112
- ImageType_PNG variable
 - [Agency Open UI Android SDK API] 1112
- ImageType_Unknown variable
 - [Agency Open UI Android SDK API] 1112
- initialize() method
 - DataTable< DTOBJ extends DataTableObject > class [Agency Java System Connection API API] 693
- initialize() method [deprecated]
 - ComplexTable< CTOBJ > class [Agency Java System Connection API API] 681
- initialize(BooleanDisplayModel, Context) method
 - [Agency Open UI Android SDK API] 998
- initialize(BooleanEditModel, Context) method
 - [Agency Open UI Android SDK API] 1000
- initialize(ButtonDisplayModel, Context) method
 - [Agency Open UI Android SDK API] 1003
- initialize(DateAndTimeDisplayModel, Context) method
 - [Agency Open UI Android SDK API] 1005

- initialize(DateAndTimeEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1007
- initialize(DateDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1010
- initialize(DateEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1012
- initialize(DecimalDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1014
- initialize(DecimalEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1017
- initialize(DurationDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1020
- initialize(DurationEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1023
- initialize(EmbeddedImageDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1025
- initialize(ExternalDataDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1027
- initialize(ExternalDataEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1030
- initialize(IntegerDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1038
- initialize(IntegerEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1041
- initialize(LabelDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1043
- initialize(LocationDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1045
- initialize(LocationEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1048
- initialize(StringDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1050
- initialize(StringEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1052
- initialize(TimeDisplayModel, Context) method
 - [Agentry Open UI Android SDK API] 1055
- initialize(TimeEditModel, Context) method
 - [Agentry Open UI Android SDK API] 1057
- initWithBooleanDisplayModel:
 - methodSMPOpenUIBooleanDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1145
- initWithBooleanEditModel:
 - methodSMPOpenUIBooleanEditAdapter protocol [Agentry Open UI iOS SDK API] 1146
- initWithButtonDisplayModel:
 - methodSMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1148
- initWithCLLocation:
 - methodSMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] 1134
- initWithCLLocation: methodSMPOpenUILocation class [Agentry Open UI iOS SDK API] 1143
- initWithCollectionDisplayModel:
 - methodSMPOpenUICollectionDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1151
- initWithDateAndTimeDisplayModel:
 - methodSMPOpenUIDateAndTimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1154
- initWithDateAndTimeEditModel:
 - methodSMPOpenUIDateAndTimeEditAdapter protocol [Agentry Open UI iOS SDK API] 1155
- initWithDateDisplayModel:
 - methodSMPOpenUIDateDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1156
- initWithDateEditModel:
 - methodSMPOpenUIDateEditAdapter protocol [Agentry Open UI iOS SDK API] 1158
- initWithDecimalDisplayModel:
 - methodSMPOpenUIDecimalDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1159
- initWithDecimalEditModel:
 - methodSMPOpenUIDecimalEditAdapter protocol [Agentry Open UI iOS SDK API] 1160
- initWithDurationDisplayModel:
 - methodSMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1162
- initWithDurationEditModel:
 - methodSMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API] 1164
- initWithEmbeddedImageModel:
 - methodSMPOpenUIEmbeddedImageDis

- playAdapter protocol [Agentry Open UI iOS SDK API] 1167
- initWithExternalDataDisplayModel:
 - methodSMPOpenUIExternalDataDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1170
- initWithExternalDataEditModel:
 - methodSMPOpenUIExternalDataEditAdapter protocol [Agentry Open UI iOS SDK API] 1171
- initWithIntegerDisplayModel:
 - methodSMPOpenUIIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1182
- initWithIntegerEditModel:
 - methodSMPOpenUIIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] 1183
- initWithLabelDisplayModel:
 - methodSMPOpenUILabelDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1185
- initWithLatitude:andLongitude:andSatellites:andDilution:
 - methodSMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] 1134
- initWithLatitude:andLongitude:andSatellites:andDilution: methodSMPOpenUILocation class [Agentry Open UI iOS SDK API] 1143
- initWithLocationDisplayModel:
 - methodSMPOpenUILocationDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1186
- initWithLocationEditModel:
 - methodSMPOpenUILocationEditAdapter protocol [Agentry Open UI iOS SDK API] 1187
- initWithStringDisplayModel:
 - methodSMPOpenUIStringDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1188
- initWithStringEditModel:
 - methodSMPOpenUIStringEditAdapter protocol [Agentry Open UI iOS SDK API] 1190
- initWithTimeDisplayModel:
 - methodSMPOpenUITimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1192
- initWithTimeEditModel:
 - methodSMPOpenUITimeEditAdapter protocol [Agentry Open UI iOS SDK API] 1193
- initWithUnsignedIntegerDisplayModel:
 - methodSMPOpenUIUnsignedIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1195
- initWithUnsignedIntegerEditModel:
 - methodSMPOpenUIUnsignedIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] 1196
- IntegerDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1037
- IntegerDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1093
- IntegerEditAdapter class [Agentry Open UI Android SDK API]
 - description 1039
- IntegerEditModel interface [Agentry Open UI Android SDK API]
 - description 1094
- InternalName property
 - [Agentry Open UI Windows SDK API] 1248
- IsAgentryActionEnabled(string) method
 - [Agentry Open UI Windows SDK API] 1204
- isAgentryDisplayingLabel() method
 - [Agentry Open UI Android SDK API] 1034
- isAgentryDisplayingValidationFailure() method
 - [Agentry Open UI Android SDK API] 1034
- IsAutoSize property
 - [Agentry Open UI Windows SDK API] 1205
- isAutosizeSupported() method
 - [Agentry Open UI Android SDK API] 1091
- isButtonSelected() method
 - [Agentry Open UI Android SDK API] 1063
- isCarriageReturnAllowed() method
 - [Agentry Open UI Android SDK API] 1103
- isDebugMode() method
 - Logger class [deprecated] [Agentry Java System Connection API API] 668
- IsEnabled property
 - [Agentry Open UI Windows SDK API] 1206
- isEnabled() method
 - [Agentry Open UI Android SDK API] 1091

isHidden() method
 [Agentry Open UI Android SDK API] 1092
 IsHyperlinkEnabled property
 [Agentry Open UI Windows SDK API] 1206
 isHyperlinkEnabled() method
 [Agentry Open UI Android SDK API] 1092
 isImageCellSelected(long, long) method
 [Agentry Open UI Android SDK API] 1084
 isImageCellSelectedAtRow:andColumn:
 methodSMPOpenUIEmbeddedImageDis
 playModel protocol [Agentry Open UI
 iOS SDK API] 1169
 isInvalidTimeAndDate() method
 SycloCalendar class [Agentry Java System
 Connection API API] 792
 isInvalidTimeAndDate(GregorianCalendar)
 method
 SycloCalendar class [Agentry Java System
 Connection API API] 792
 isOutOfDate() method
 DataTable< DTOBJ extends DataTableObject
 > class [Agentry Java System
 Connection API API] 693
 IsPassword property
 [Agentry Open UI Windows SDK API] 1244
 isPasswordInput
 propertySMPOpenUIStringEditModel
 protocol [Agentry Open UI iOS SDK
 API] 1191
 isPasswordInput() method
 [Agentry Open UI Android SDK API] 1106
 isRebuilding() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 682
 IsSelected(int, int) method
 [Agentry Open UI Windows SDK API] 1232
 isValid() method
 [Agentry Open UI Android SDK API] 1114,
 1117, 1121, 1133
 IsVisible property
 [Agentry Open UI Windows SDK API] 1206
 isWordWrapAllowed() method
 [Agentry Open UI Android SDK API] 1103
 iterator() method
 DataTable< DTOBJ extends DataTableObject
 > class [Agentry Java System
 Connection API API] 693

J

java_logging package [Agentry Java System
 Connection API API]
 description 647

L

Label property
 [Agentry Open UI Windows SDK API] 1206
 label propertySMPOpenUIFieldModel protocol
 [Agentry Open UI iOS SDK API] 1181
 LabelDisplayAdapter class [Agentry Open UI
 Android SDK API]
 description 1041
 LabelDisplayModel interface [Agentry Open UI
 Android SDK API]
 description 1097
 lastUpdateDate() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 682
 lastUpdateHours() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 682
 lastUpdateMinutes() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 682
 lastUpdateMonth() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 683
 lastUpdateSeconds() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 683
 lastUpdateYear() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 683
 latitude propertySMPDataAPILocationProtocol
 protocol [Agentry Open UI iOS SDK
 API] 1135
 latitude propertySMPOpenUILocation class
 [Agentry Open UI iOS SDK API] 1144
 launchActivity(Intent, int) method
 [Agentry Open UI Android SDK API] 1092
 location propertySMPDataAPILocationProtocol
 protocol [Agentry Open UI iOS SDK
 API] 1135
 location propertySMPOpenUILocation class
 [Agentry Open UI iOS SDK API] 1144

Index

- LocationDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1044
- LocationDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1098
- LocationEditAdapter class [Agentry Open UI Android SDK API]
 - description 1046
- LocationEditModel interface [Agentry Open UI Android SDK API]
 - description 1100
- locationWithCLLocation:
 - methodSMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] 1134
- locationWithCLLocation:
 - methodSMPOpenUILocation class [Agentry Open UI iOS SDK API] 1144
- locationWithLatitude:andLongitude:andSatellites:andDilution:
 - methodSMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] 1135
- locationWithLatitude:andLongitude:andSatellites:andDilution:
 - methodSMPOpenUILocation class [Agentry Open UI iOS SDK API] 1144
- log methodSMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] 1138
- log methodSMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] 1140
- log(LogRecord) method
 - UserLogger class [Agentry Java System Connection API API] 661
- log4j package [Agentry Java System Connection API API]
 - description 661
- loggedIn() method
 - User class [Agentry Java System Connection API API] 803
- loggedOut() method
 - User class [Agentry Java System Connection API API] 803
- Logger class [Agentry Java System Connection API API]
 - description 665
- Logger class [deprecated] [Agentry Java System Connection API API]
 - appendDebug(String) method 666
 - beginDebug(String) method 667
 - debug(String, Map< String, String >, String) method 667
 - debug(String) method 667
 - endDebug(String) method 667
 - isDebugMode() method 668
 - Logger(String, boolean) constructor 666
- Logger(String, boolean) constructor
 - Logger class [deprecated] [Agentry Java System Connection API API] 666
- Login_Invalid variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 747
- Login_InvalidBlocked variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 748
- Login_Pass variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 748
- Login_Valid variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 748
- Login_ValidPasswordExpired variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 748
- Login_ValidPasswordExpiredNoChange variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 748
- Login_ValidPasswordWarning variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 749
- Login_ValidPasswordWarningNoChange variable
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 749
- login(String, String, SessionData) method [deprecated]
 - Server class [Agentry Java System Connection API API] 761

- login(User, String, SessionData) method
 - Server class [Agentry Java System Connection API API] 761
 - loginBlocked(String, StringBuffer) method
 - [deprecated]
 - Server class [Agentry Java System Connection API API] 762
 - loginBlocked(User, String, StringBuffer, SessionData) method
 - Server class [Agentry Java System Connection API API] 763
 - loginBlocked(User, StringBuffer, SessionData) method [deprecated]
 - Server class [Agentry Java System Connection API API] 763
 - LoginBlockedException class [Agentry Java System Connection API API]
 - description 704
 - LoginBlockedException() constructor 706
 - LoginBlockedException(String, Throwable) constructor 706
 - LoginBlockedException(String) constructor 706
 - LoginBlockedException() constructor
 - LoginBlockedException class [Agentry Java System Connection API API] 706
 - LoginBlockedException(String, Throwable) constructor
 - LoginBlockedException class [Agentry Java System Connection API API] 706
 - LoginBlockedException(String) constructor
 - LoginBlockedException class [Agentry Java System Connection API API] 706
 - LoginException class [Agentry Java System Connection API API]
 - description 706
 - LoginException() constructor 708
 - LoginException(String, Throwable) constructor 708
 - LoginException(String) constructor 708
 - LoginException() constructor
 - LoginException class [Agentry Java System Connection API API] 708
 - LoginException(String, Throwable) constructor
 - LoginException class [Agentry Java System Connection API API] 708
 - LoginException(String) constructor
 - LoginException class [Agentry Java System Connection API API] 708
 - loginFailed(String, StringBuffer) method
 - [deprecated]
 - Server class [Agentry Java System Connection API API] 764
 - loginFailed(User, String, LoginFailureReason, StringBuffer, SessionData) method
 - Server class [Agentry Java System Connection API API] 765
 - loginPreviousUser(String, String, SessionData) method [deprecated]
 - Server class [Agentry Java System Connection API API] 766
 - loginPreviousUser(User, String, SessionData) method
 - Server class [Agentry Java System Connection API API] 766
 - LoginSkippedException class [Agentry Java System Connection API API]
 - description 709
 - LoginSkippedException() constructor 710
 - LoginSkippedException(String, Throwable) constructor 711
 - LoginSkippedException(String) constructor 710
 - LoginSkippedException() constructor
 - LoginSkippedException class [Agentry Java System Connection API API] 710
 - LoginSkippedException(String, Throwable) constructor
 - LoginSkippedException class [Agentry Java System Connection API API] 711
 - longitude propertySMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] 1135
 - longitude propertySMPOpenUILocation class [Agentry Open UI iOS SDK API] 1145
- ## M
- mapLogLevel(Level) method
 - AgentryAppender class [Agentry Java System Connection API API] 663
 - AgentryHandler class [Agentry Java System Connection API API] 650
 - MaskColor class [Agentry Open UI Android SDK API]
 - description 1119

Index

- MaskColor(int, int, int) constructor
 - [Agentry Open UI Android SDK API] 1120
- MaskColor(short, short, short) constructor
 - [Agentry Open UI Android SDK API] 1119
- Maximum property
 - [Agentry Open UI Windows SDK API] 1237
- maximumFractionalHourValue
 - propertySMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] 1166
- MaxLength property
 - [Agentry Open UI Windows SDK API] 1245
- maxLength
 - propertySMPOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] 1191
- MaximumValue property
 - [Agentry Open UI Windows SDK API] 1223
- maximumValue
 - propertySMPOpenUIDecimalEditModel protocol [Agentry Open UI iOS SDK API] 1161
- maximumValue
 - propertySMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] 1166
- maximumValue
 - propertySMPOpenUIIntegerEditModel protocol [Agentry Open UI iOS SDK API] 1184
- maximumValue
 - propertySMPOpenUIUnsignedIntegerEditModel protocol [Agentry Open UI iOS SDK API] 1197
- Minimum property
 - [Agentry Open UI Windows SDK API] 1237
- minimumFractionalHourValue
 - propertySMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] 1167
- MinimumLength property
 - [Agentry Open UI Windows SDK API] 1245
- minimumLength
 - propertySMPOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] 1192
- MinimumValue property
 - [Agentry Open UI Windows SDK API] 1223
- minimumValue
 - propertySMPOpenUIDecimalEditModel protocol [Agentry Open UI iOS SDK API] 1162
- minimumValue
 - propertySMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] 1167
- minimumValue
 - propertySMPOpenUIIntegerEditModel protocol [Agentry Open UI iOS SDK API] 1184
- minimumValue
 - propertySMPOpenUIUnsignedIntegerEditModel protocol [Agentry Open UI iOS SDK API] 1197
- MinSec variable
 - [Agentry Open UI Android SDK API] 1127
- model:didChangeBoolean:
 - methodSMPOpenUIBooleanDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1146
- model:didChangeBoolean:
 - methodSMPOpenUIBooleanEditAdapter protocol [Agentry Open UI iOS SDK API] 1147
- model:didChangeButtonImage:
 - methodSMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1148
- model:didChangeDate:
 - methodSMPOpenUIDateDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1157
- model:didChangeDate:
 - methodSMPOpenUIDateEditAdapter protocol [Agentry Open UI iOS SDK API] 1158
- model:didChangeDateAndTime:
 - methodSMPOpenUIDateAndTimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1154
- model:didChangeDateAndTime:
 - methodSMPOpenUIDateAndTimeEditAdapter protocol [Agentry Open UI iOS SDK API] 1155
- model:didChangeDecimal:
 - methodSMPOpenUIDecimalDisplayAda

pter protocol [Agentry Open UI iOS SDK API] 1159
 model:didChangeDecimal:
 methodSMPOpenUIDecimalEditAdapter protocol [Agentry Open UI iOS SDK API] 1161
 model:didChangeDuration:
 methodSMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1162
 model:didChangeDuration:
 methodSMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API] 1164
 model:didChangeExternalData:
 methodSMPOpenUIExternalDataDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1170
 model:didChangeExternalData:
 methodSMPOpenUIExternalDataEditAdapter protocol [Agentry Open UI iOS SDK API] 1171
 model:didChangeFractionalHour:
 methodSMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1163
 model:didChangeFractionalHour:
 methodSMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API] 1165
 model:didChangeImage:
 methodSMPOpenUIEmbeddedImageDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1168
 model:didChangeInteger:
 methodSMPOpenUIIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1182
 model:didChangeInteger:
 methodSMPOpenUIIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] 1183
 model:didChangeLabel:
 methodSMPOpenUILabelDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1185
 model:didChangeLocation:
 methodSMPOpenUILocationDisplayAd

apter protocol [Agentry Open UI iOS SDK API] 1186
 model:didChangeLocation:
 methodSMPOpenUILocationEditAdapter protocol [Agentry Open UI iOS SDK API] 1187
 model:didChangeSelected:
 methodSMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1148
 model:didChangeString:
 methodSMPOpenUIStringDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1189
 model:didChangeString:
 methodSMPOpenUIStringEditAdapter protocol [Agentry Open UI iOS SDK API] 1190
 model:didChangeTime:
 methodSMPOpenUITimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1192
 model:didChangeTime:
 methodSMPOpenUITimeEditAdapter protocol [Agentry Open UI iOS SDK API] 1194
 model:didChangeUnsignedInteger:
 methodSMPOpenUIUnsignedIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1195
 model:didChangeUnsignedInteger:
 methodSMPOpenUIUnsignedIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] 1196
 model:didSelectObjectAtIndex:
 methodSMPOpenUICollectionDisplayAdapter protocol [Agentry Open UI iOS SDK API] 1151
 model:didSetEnabled:
 methodSMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API] 1175
 model:didSetHyperlinkEnabled:
 methodSMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API] 1175
 model:didSetValid:withValidationFailureText:
 methodSMPOpenUIFieldAdapter

Index

protocol [Agentry Open UI iOS SDK API] 1175

model:didSetVisible:
methodSMPOpenUIFieldAdapter
protocol [Agentry Open UI iOS SDK API] 1176

model:didUpdateLabel:
methodSMPOpenUIFieldAdapter
protocol [Agentry Open UI iOS SDK API] 1176

model:objectAddedAtIndex:
methodSMPOpenUICollectionDisplayA
dapter protocol [Agentry Open UI iOS SDK API] 1151

model:objectChangedAtIndex:
methodSMPOpenUICollectionDisplayA
dapter protocol [Agentry Open UI iOS SDK API] 1152

model:objectDeletedAtIndex:
methodSMPOpenUICollectionDisplayA
dapter protocol [Agentry Open UI iOS SDK API] 1152

model:wantsExtensionString:
methodSMPOpenUIFieldAdapter
protocol [Agentry Open UI iOS SDK API] 1177

model:wantsViewHeightForWidth:
methodSMPOpenUIFieldAdapter
protocol [Agentry Open UI iOS SDK API] 1177

modelDidChangeImageCellSelection:
methodSMPOpenUIEmbeddedImageDis
playAdapter protocol [Agentry Open UI iOS SDK API] 1168

models package [Agentry Open UI Android SDK API]
description 1057

N

name methodSMPDataAPIProtocol protocol
[Agentry Open UI iOS SDK API] 1140

name propertySMPOpenUIImage class [Agentry
Open UI iOS SDK API] 1142

needsBitmapData() method
[Agentry Open UI Android SDK API] 1115

next() method
DataTableMapIterator< K, V > class [Agentry
Java System Connection API API]
665

NoBackEndsAuthenticated variable
Server.LoginFailureReason enum [Agentry
Java System Connection API API]
750

notificationText() method [deprecated]
Steplet class [Agentry Java System Connection
API API] 785

notificationTitle() method [deprecated]
Steplet class [Agentry Java System Connection
API API] 785

NumberValue property
[Agentry Open UI Windows SDK API] 1239

O

Objects() method
[Agentry Open UI Windows SDK API] 1247

okButtonLabel() method [deprecated]
Steplet class [Agentry Java System Connection
API API] 785

onActivityResult(int, int, Intent) method
[Agentry Open UI Android SDK API] 1035

OnPropertyChanged(string) method
[Agentry Open UI Windows SDK API] 1205

openui package [Agentry Open UI Android SDK
API]
description 996, 1110

OpenUIImage interface [Agentry Open UI Android
SDK API]
description 1132

P

PasswordExpiredCannotChange variable
Server.LoginFailureReason enum [Agentry
Java System Connection API API]
750

PasswordExpiredCannotChangeException class
[Agentry Java System Connection API
API]
description 711

PasswordExpiredCannotChangeException()
constructor 713

PasswordExpiredCannotChangeException(Str
ing, Throwable) constructor 713

PasswordExpiredCannotChangeException(Str
ing) constructor 713

- PasswordExpiredCannotChangeException() constructor
 - PasswordExpiredCannotChangeException class [Agentry Java System Connection API API] 713
- PasswordExpiredCannotChangeException(String, Throwable) constructor
 - PasswordExpiredCannotChangeException class [Agentry Java System Connection API API] 713
- PasswordExpiredCannotChangeException(String) constructor
 - PasswordExpiredCannotChangeException class [Agentry Java System Connection API API] 713
- PasswordExpiredException class [Agentry Java System Connection API API] description 713
 - PasswordExpiredException() constructor 715
 - PasswordExpiredException(String, Throwable) constructor 715
 - PasswordExpiredException(String) constructor 715
- PasswordExpiredException() constructor
 - PasswordExpiredException class [Agentry Java System Connection API API] 715
- PasswordExpiredException(String, Throwable) constructor
 - PasswordExpiredException class [Agentry Java System Connection API API] 715
- PasswordExpiredException(String) constructor
 - PasswordExpiredException class [Agentry Java System Connection API API] 715
- PasswordInvalid variable
 - Server.LoginFailureReason enum [Agentry Java System Connection API API] 750
- PasswordInvalidException class [Agentry Java System Connection API API] description 716
 - PasswordInvalidException() constructor 717
 - PasswordInvalidException(String, Throwable) constructor 718
 - PasswordInvalidException(String) constructor 717
- PasswordInvalidException() constructor
 - PasswordInvalidException class [Agentry Java System Connection API API] 717
- PasswordInvalidException(String, Throwable) constructor
 - PasswordInvalidException class [Agentry Java System Connection API API] 717
- PasswordWarningCannotChangeException class [Agentry Java System Connection API API] description 718
 - PasswordWarningCannotChangeException() constructor 720
 - PasswordWarningCannotChangeException(String, Throwable) constructor 720
 - PasswordWarningCannotChangeException(String) constructor 720
- PasswordWarningCannotChangeException() constructor
 - PasswordWarningCannotChangeException class [Agentry Java System Connection API API] 720
- PasswordWarningCannotChangeException(String, Throwable) constructor
 - PasswordWarningCannotChangeException class [Agentry Java System Connection API API] 720
- PasswordWarningCannotChangeException(String) constructor
 - PasswordWarningCannotChangeException class [Agentry Java System Connection API API] 720
- PasswordWarningException class [Agentry Java System Connection API API] description 720
 - PasswordWarningException() constructor 722
 - PasswordWarningException(String, Throwable) constructor 722
 - PasswordWarningException(String) constructor 722
- PasswordWarningException() constructor
 - PasswordWarningException class [Agentry Java System Connection API API] 722

Index

- PasswordWarningException(String, Throwable)
 - constructor
- PasswordWarningException class [Agentry Java System Connection API API] 722
- PasswordWarningException(String) constructor
 - PasswordWarningException class [Agentry Java System Connection API API] 722
- position propertySMPOpenUIImage class [Agentry Open UI iOS SDK API] 1142
- presentation propertySMPOpenUIImage class [Agentry Open UI iOS SDK API] 1143
- processDecimalInput(double) method [Agentry Open UI Android SDK API] 1080
- processInput
 - methodSMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] 1149
- processInput() method [Agentry Open UI Android SDK API] 1064
- processInput(AgentryLocation) method [Agentry Open UI Android SDK API] 1101
- processInput(boolean) method [Agentry Open UI Android SDK API] 1061
- ProcessInput(DateTime) method [Agentry Open UI Windows SDK API] 1211
- processInput(double) method [Agentry Open UI Android SDK API] 1074
- ProcessInput(double) method [Agentry Open UI Windows SDK API] 1217
- processInput(GregorianCalendar) method [Agentry Open UI Android SDK API] 1067, 1110
- processInput(int) method [Agentry Open UI Android SDK API] 1080
- processInput(String) method [Agentry Open UI Android SDK API] 1087, 1106
- ProcessInput(string) method [Agentry Open UI Windows SDK API] 1227, 1244
- ProcessInput(T) method [Agentry Open UI Windows SDK API] 1237
- ProcessInput(TimeSpan) method [Agentry Open UI Windows SDK API] 1223
- processInputBoolean:
 - methodSMPOpenUIBooleanEditModel
- protocol [Agentry Open UI iOS SDK API] 1147
- processInputDate:
 - methodSMPOpenUIDateEditModel
 - protocol [Agentry Open UI iOS SDK API] 1158
- processInputDateAndTime:
 - methodSMPOpenUIDateAndTimeEditModel protocol [Agentry Open UI iOS SDK API] 1156
- processInputDecimal:
 - methodSMPOpenUIDecimalEditModel
 - protocol [Agentry Open UI iOS SDK API] 1161
- processInputDuration:
 - methodSMPOpenUIDurationEditModel
 - protocol [Agentry Open UI iOS SDK API] 1165
- processInputExternalData:
 - methodSMPOpenUIExternalDataEditModel protocol [Agentry Open UI iOS SDK API] 1172
- processInputFractionalHour:
 - methodSMPOpenUIDurationEditModel
 - protocol [Agentry Open UI iOS SDK API] 1166
- processInputInteger:
 - methodSMPOpenUIIntegerEditModel
 - protocol [Agentry Open UI iOS SDK API] 1184
- processInputLocation:
 - methodSMPOpenUILocationEditModel
 - protocol [Agentry Open UI iOS SDK API] 1188
- ProcessInputReturn class [Agentry Open UI Android SDK API]
 - description 1121
- ProcessInputReturn(boolean, boolean, boolean)
 - constructor
 - [Agentry Open UI Android SDK API] 1122
- processInputSelection:
 - methodSMPOpenUICollectionDisplayModel protocol [Agentry Open UI iOS SDK API] 1153
- processInputString:
 - methodSMPOpenUIStringEditModel
 - protocol [Agentry Open UI iOS SDK API] 1191

processInputTime:
 methodSMPOpenUITimeEditModel
 protocol [Agentry Open UI iOS SDK
 API] 1194

processInputUnsignedInteger:
 methodSMPOpenUIUnsignedIntegerEditModel
 protocol [Agentry Open UI iOS
 SDK API] 1197

processIntegerInput(int) method
 [Agentry Open UI Android SDK API] 1096

Properties() method
 [Agentry Open UI Windows SDK API] 1247

propertyType
 methodSMPDataAPIPropertyProtocol
 protocol [Agentry Open UI iOS SDK
 API] 1138

PropertyType property
 [Agentry Open UI Windows SDK API] 1254

publish(LogRecord) method
 AgentryHandler class [Agentry Java System
 Connection API API] 650

PushSession class [Agentry Java System
 Connection API API]
 beginPushError() method [deprecated] 726
 beginPushReadStep() method 727
 beginPushRemoval() method 727
 beginPushResponse() method [deprecated]
 727
 beginPushRetrieval() method 728
 description 723
 endPushError() method [deprecated] 728
 endPushReadStep() method 728
 endPushRemoval() method 729
 endPushResponse() method [deprecated] 729
 endPushRetrieval() method 729
 PushSession(String, Server, SessionData)
 constructor 726

PushSession(String, Server, SessionData)
 constructor
 PushSession class [Agentry Java System
 Connection API API] 726

PushUserSession class [Agentry Java System
 Connection API API]
 beginDisablePush() method 733
 beginEnablePush() method 733
 beginPushError() method 733
 beginPushResponse() method 734
 description 729
 disablePush() method 734

enablePush() method 734
 endDisablePush() method 734
 endEnablePush() method 735
 endPushError() method 735
 endPushResponse() method 735
 PushUserSession(String, Server, SessionData,
 User) constructor 732
 PushUserSession(String, Server, SessionData,
 User) constructor
 PushUserSession class [Agentry Java System
 Connection API API] 732

R

reload() method [deprecated]
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 684

reLoggedIn() method
 User class [Agentry Java System Connection
 API API] 804

remove() method
 DataTableMapIterator< K, V > class [Agentry
 Java System Connection API API]
 665

requestLayoutHeight:
 methodSMPOpenUIFieldModel protocol
 [Agentry Open UI iOS SDK API] 1180

requestLayoutHeight(int) method
 [Agentry Open UI Android SDK API] 1092

requiresLayout() method
 AgentryAppender class [Agentry Java System
 Connection API API] 663

RetryTransactionException class [Agentry Java
 System Connection API API]
 description 735
 RetryTransactionException(String, String,
 String, Throwable) constructor 737
 RetryTransactionException(String, String,
 String) constructor 737

RetryTransactionException(String, String, String,
 Throwable) constructor
 RetryTransactionException class [Agentry
 Java System Connection API API]
 737

RetryTransactionException(String, String, String)
 constructor
 RetryTransactionException class [Agentry
 Java System Connection API API]
 737

Index

RetryTransactionWithChangeException class
 [Agentry Java System Connection API
 API]
 description 737
 RetryTransactionWithChangeException(String,
 String, String, Throwable)
 constructor 739
 RetryTransactionWithChangeException(String,
 String, String) constructor 739
RetryTransactionWithChangeException(String,
 String, String, Throwable) constructor
RetryTransactionWithChangeException class
 [Agentry Java System Connection
 API API] 739
RetryTransactionWithChangeException(String,
 String, String) constructor
 RetryTransactionWithChangeException class
 [Agentry Java System Connection
 API API] 739
revalidate(String) method
 User class [Agentry Java System Connection
 API API] 804
root methodSMPDataAPIProtocol protocol
 [Agentry Open UI iOS SDK API] 1140
Root property
 [Agentry Open UI Windows SDK API] 1249
Rows property
 [Agentry Open UI Windows SDK API] 1233
rows
 propertySMPOpenUIEmbeddedImageDi
 splayModel protocol [Agentry Open UI
 iOS SDK API] 1169

S

satellites propertySMPDataAPILocationProtocol
 protocol [Agentry Open UI iOS SDK
 API] 1136
satellites propertySMPOpenUILocation class
 [Agentry Open UI iOS SDK API] 1145
SelectCell(int, int) method
 [Agentry Open UI Windows SDK API] 1232
SelectColor property
 [Agentry Open UI Windows SDK API] 1233
selected propertySMPOpenUIButtonDisplayModel
 protocol [Agentry Open UI iOS SDK
 API] 1150
SelectedItem property
 [Agentry Open UI Windows SDK API] 1209
selectedStateChanged(boolean) method
 [Agentry Open UI Android SDK API] 1003

selection
 propertySMPOpenUICollectionDisplay
 Model protocol [Agentry Open UI iOS
 SDK API] 1153
SelectItem(int) method
 [Agentry Open UI Windows SDK API] 1209
Server class [Agentry Java System Connection API
 API]
 createComplexTableSession(String,
 SessionData, User) method 750
 createDataTableSession(String, SessionData,
 User) method 751
 createFetchSession(String, Server,
 SessionData, User) method
 [deprecated] 752
 createFetchSession(String, SessionData, User)
 method 752
 createPushSession(String, Server,
 SessionData) method [deprecated]
 753
 createPushSession(String, SessionData)
 method 753
 createPushUserSession(String, Server,
 SessionData, User) method
 [deprecated] 754
 createPushUserSession(String, SessionData,
 User) method 754
 createServiceEventSession(String, Server,
 SessionData) method [deprecated]
 755
 createServiceEventSession(String,
 SessionData) method 756
 createTransactionSession(String, Server,
 SessionData, User) method
 [deprecated] 756
 createTransactionSession(String,
 SessionData, User) method 757
 createUser(String, int) method [deprecated]
 757
 createUser(String) method 758
 debug(String) method 758
 decryptPassword(String) method 759
 description 739
 findConfigurationFile(String) method 759
 getImplementationVersion() method 759
 getInstance() method 760
 getSpecificationVersion() method 760
 getTimeZone() method 760

- login(String, String, SessionData) method [deprecated] 761
- login(User, String, SessionData) method 761
- loginBlocked(String, StringBuffer) method [deprecated] 762
- loginBlocked(User, String, StringBuffer, SessionData) method 763
- loginBlocked(User, StringBuffer, SessionData) method [deprecated] 763
- loginFailed(String, StringBuffer) method [deprecated] 764
- loginFailed(User, String, LoginFailureReason, StringBuffer, SessionData) method 765
- loginPreviousUser(String, String, SessionData) method [deprecated] 766
- loginPreviousUser(User, String, SessionData) method 766
- Server() constructor 750
- setDebugEnabled(boolean) method [deprecated] 767
- shutdown() method 767
- startup() method 767
- Server.LoginEnumeration enum [Agentry Java System Connection API API] description 746
- Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API]
- Login_Invalid variable 747
- Login_InvalidBlocked variable 748
- Login_Pass variable 748
- Login_Valid variable 748
- Login_ValidPasswordExpired variable 748
- Login_ValidPasswordExpiredNoChange variable 748
- Login_ValidPasswordWarning variable 749
- Login_ValidPasswordWarningNoChange variable 749
- throwException() method 747
- Server.LoginFailureReason enum [Agentry Java System Connection API API] description 749
- NoBackEndsAuthenticated variable 750
- PasswordExpiredCannotChange variable 750
- PasswordInvalid variable 750
- Server() constructor
 - Server class [Agentry Java System Connection API API] 750
- ServiceEvent class [Agentry Java System Connection API API]
 - _server variable 770
 - _sessionData variable 770
 - dataReceived(Object) method 769
 - description 768
 - ServiceEvent(Server, SessionData, CallbackInterface) constructor 769
- ServiceEvent(Server, SessionData, CallbackInterface) constructor
 - ServiceEvent class [Agentry Java System Connection API API] 769
- ServiceEventSession class [Agentry Java System Connection API API]
 - beginDataAndUpdateSteps() method 773
 - beginReadSteps() method 773
 - beginServiceEventError() method 774
 - description 770
 - endDataAndUpdateSteps() method 774
 - endReadSteps() method 774
 - endServiceEventError() method 774
 - ServiceEventSession(String, Server, SessionData) constructor 773
- ServiceEventSession(String, Server, SessionData) constructor
 - ServiceEventSession class [Agentry Java System Connection API API] 773
- Session class [Agentry Java System Connection API API]
 - debug(String) method 777
 - description 775
 - getName() method 777
 - getServer() method 777
 - getSessionData() method 778
 - getUser() method 778
 - Session(String, Server, SessionData, User) constructor 776
 - Session(String, Server, SessionData) constructor 776
 - sessionAborted() method 778
 - Session(String, Server, SessionData, User) constructor
 - Session class [Agentry Java System Connection API API] 776

Index

- Session(String, Server, SessionData) constructor
 - Session class [Agentry Java System Connection API API] 776
- sessionAborted() method
 - Session class [Agentry Java System Connection API API] 778
- SessionData interface [Agentry Java System Connection API API]
 - description 805
 - eval(String) method 808
 - getBoolean(String) method 808
 - getBytes(String) method 808
 - getDouble(String) method 809
 - getFloat(String) method 809
 - getInteger(String) method 809
 - getLong(String) method 809
 - getString(String) method 810
 - getTimeAndDate(String, String) method 810
 - getTimeAndDate(String) method 810
 - sessionData(String) constructor 807
- sessionData(String) constructor
 - SessionData interface [Agentry Java System Connection API API] 807
- setBitmapData(byte[]) method
 - [Agentry Open UI Android SDK API] 1115
- setDebugEnabled(boolean) method [deprecated]
 - Server class [Agentry Java System Connection API API] 767
- setDilution(double) method
 - [Agentry Open UI Android SDK API] 1118
- setEnabled(boolean) method
 - [Agentry Open UI Android SDK API] 1035
- setHyperlinkEnabled(boolean) method
 - [Agentry Open UI Android SDK API] 1035
- setImageCellSelected(long, long) method
 - [Agentry Open UI Android SDK API] 1084
- setLatitude(double) method
 - [Agentry Open UI Android SDK API] 1118
- setLevel(Level) method
 - UserLogRecord class [Agentry Java System Connection API API] 657
- setLoggerName(String) method
 - UserLogRecord class [Agentry Java System Connection API API] 657
- setLongitude(double) method
 - [Agentry Open UI Android SDK API] 1118
- setMessage(String) method
 - UserLogRecord class [Agentry Java System Connection API API] 657
- setMillis(long) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setNewDataUpdateTime(GregorianCalendar) method
 - ComplexTable< CTOBJ > class [Agentry Java System Connection API API] 684
- setParameters(Object[]) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setResourceBundle(ResourceBundle) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setResourceBundleName(String) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setSatellites(int) method
 - [Agentry Open UI Android SDK API] 1118
- setSequenceNumber(long) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setSourceClassName(String) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setSourceMethodName(String) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setThreadID(int) method
 - UserLogRecord class [Agentry Java System Connection API API] 658
- setThrown(Throwable) method
 - UserLogRecord class [Agentry Java System Connection API API] 659
- setValid(boolean, String) method
 - [Agentry Open UI Android SDK API] 1036
- setValid(boolean) method
 - [Agentry Open UI Android SDK API] 1118
- setVisible(boolean) method
 - [Agentry Open UI Android SDK API] 1036
- shutdown() method
 - Server class [Agentry Java System Connection API API] 767
- SMPActionResult enumeration
 - [Agentry Open UI Windows SDK API] 1256
- SMPActionState enumeration
 - [Agentry Open UI Windows SDK API] 1257
- SMPDataAPIDataType enumeration
 - SMPDataAPIProtocols.h file [Agentry Open UI iOS SDK API] 1141

- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] description 1134
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] dilution property 1135
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] initWithCLLocation: method 1134
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] initWithLatitude:andLongitude:andSatellites:andDilution: method 1134
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] latitude property 1135
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] location property 1135
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] locationWithCLLocation: method 1134
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] locationWithLatitude:andLongitude:andSatellites:andDilution: method 1135
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] longitude property 1135
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] satellites property 1136
- SMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] valid property 1136
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asBool method 1136
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asDate method 1136
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asDateAndTime method 1136
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asDecimal method 1137
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asLocation method 1137
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asLong method 1137
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asString method 1137
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] asTime method 1138
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] description 1136
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] log method 1138
- SMPDataAPIPropertyProtocol protocol [Agentry Open UI iOS SDK API] propertyType method 1138
- SMPDataAPIPropertyType enumeration
- SMPDataAPIProtocols.h file [Agentry Open UI iOS SDK API] 1141
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] ancestor method 1139
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] dataIdentifier method 1139
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] dataType method 1139
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] descendant: method 1139
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] descendantCount method 1140
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] description 1138
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] displayName method 1140
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] log method 1140
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] name method 1140
- SMPDataAPIProtocol protocol [Agentry Open UI iOS SDK API] root method 1140

- SMPDataAPIProtocols.h file [Agentry Open UI iOS SDK API] SMPDataAPIDataType enumeration 1141
- SMPDataAPIProtocols.h file [Agentry Open UI iOS SDK API] SMPDataAPIPropertyType enumeration 1141
- SMPDurationFormat enumeration [Agentry Open UI Windows SDK API] 1257
- SMPOpenUIActionEnableType enumeration SMPOpenUIFieldModel.h file [Agentry Open UI iOS SDK API] 1197
- SMPOpenUIActionResult enumeration SMPOpenUIFieldModel.h file [Agentry Open UI iOS SDK API] 1198
- SMPOpenUIAutosizeBehavior enumeration SMPOpenUIFieldAdapter.h file [Agentry Open UI iOS SDK API] 1198
- SMPOpenUIBooleanDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1145
- SMPOpenUIBooleanDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithBooleanDisplayModel: method 1145
- SMPOpenUIBooleanDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeBoolean: method 1146
- SMPOpenUIBooleanDisplayModel protocol [Agentry Open UI iOS SDK API] description 1146
- SMPOpenUIBooleanDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1146
- SMPOpenUIBooleanEditAdapter protocol [Agentry Open UI iOS SDK API] description 1146
- SMPOpenUIBooleanEditAdapter protocol [Agentry Open UI iOS SDK API] initWithBooleanEditModel: method 1146
- SMPOpenUIBooleanEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeBoolean: method 1147
- SMPOpenUIBooleanEditModel protocol [Agentry Open UI iOS SDK API] description 1147
- SMPOpenUIBooleanEditModel protocol [Agentry Open UI iOS SDK API] processInputBoolean: method 1147
- SMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1148
- SMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithButtonDisplayModel: method 1148
- SMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeButtonImage: method 1148
- SMPOpenUIButtonDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeSelected: method 1148
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] buttonImage property 1149
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] buttonText property 1149
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] buttonType property 1149
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] description 1149
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] processInput method 1149
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] selected property 1150
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] supportsAction property 1150
- SMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1150
- SMPOpenUIButtonDisplayModel.h file [Agentry Open UI iOS SDK API] SMPOpenUIButtonType enumeration 1198
- SMPOpenUIButtonType enumeration SMPOpenUIButtonDisplayModel.h file [Agentry Open UI iOS SDK API] 1198
- SMPOpenUICollectionDisplayAdapter protocol [Agentry Open UI iOS SDK API] allObjectsChanged: method 1150

- SMPOpenUICollectionDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
description 1150
- SMPOpenUICollectionDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
initWithCollectionDisplayModel:
method 1151
- SMPOpenUICollectionDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
model:didSelectObjectAtIndex: method
1151
- SMPOpenUICollectionDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
model:objectAddedAtIndex: method
1151
- SMPOpenUICollectionDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
model:objectChangedAtIndex: method
1152
- SMPOpenUICollectionDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
model:objectDeletedAtIndex: method
1152
- SMPOpenUICollectionDisplayModel protocol
[Agentry Open UI iOS SDK API]
collection method 1152
- SMPOpenUICollectionDisplayModel protocol
[Agentry Open UI iOS SDK API]
description 1152
- SMPOpenUICollectionDisplayModel protocol
[Agentry Open UI iOS SDK API]
displayedObjectAtIndex: method 1153
- SMPOpenUICollectionDisplayModel protocol
[Agentry Open UI iOS SDK API]
displayedObjectCount property 1153
- SMPOpenUICollectionDisplayModel protocol
[Agentry Open UI iOS SDK API]
processInputSelection: method 1153
- SMPOpenUICollectionDisplayModel protocol
[Agentry Open UI iOS SDK API]
selection property 1153
- SMPOpenUIDateAndTimeDisplayAdapter
protocol [Agentry Open UI iOS SDK
API] description 1154
- SMPOpenUIDateAndTimeDisplayAdapter
protocol [Agentry Open UI iOS SDK
API] model:didChangeDateAndTime:
method 1154
- SMPOpenUIDateAndTimeDisplayModel protocol
[Agentry Open UI iOS SDK API]
description 1154
- SMPOpenUIDateAndTimeDisplayModel protocol
[Agentry Open UI iOS SDK API] value
property 1155
- SMPOpenUIDateAndTimeEditAdapter protocol
[Agentry Open UI iOS SDK API]
description 1155
- SMPOpenUIDateAndTimeEditAdapter protocol
[Agentry Open UI iOS SDK API]
initWithDateAndTimeEditModel:
method 1155
- SMPOpenUIDateAndTimeEditAdapter protocol
[Agentry Open UI iOS SDK API]
model:didChangeDateAndTime: method
1155
- SMPOpenUIDateAndTimeEditModel protocol
[Agentry Open UI iOS SDK API]
description 1156
- SMPOpenUIDateAndTimeEditModel protocol
[Agentry Open UI iOS SDK API]
processInputDateAndTime: method
1156
- SMPOpenUIDateDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
description 1156
- SMPOpenUIDateDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
initWithDateDisplayModel: method
1156
- SMPOpenUIDateDisplayAdapter protocol
[Agentry Open UI iOS SDK API]
model:didChangeDate: method 1157
- SMPOpenUIDateDisplayModel protocol [Agentry
Open UI iOS SDK API] description
1157
- SMPOpenUIDateDisplayModel protocol [Agentry
Open UI iOS SDK API] value property
1157
- SMPOpenUIDateEditAdapter protocol [Agentry
Open UI iOS SDK API] description
1157

- SMPOpenUIDateEditAdapter protocol [Agentry Open UI iOS SDK API]
 - initWithDateEditModel: method 1158
- SMPOpenUIDateEditAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didChangeDate: method 1158
- SMPOpenUIDateEditModel protocol [Agentry Open UI iOS SDK API] description 1158
- SMPOpenUIDateEditModel protocol [Agentry Open UI iOS SDK API]
 - processInputDate: method 1158
- SMPOpenUIDecimalDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1159
- SMPOpenUIDecimalDisplayAdapter protocol [Agentry Open UI iOS SDK API]
 - initWithDecimalDisplayModel: method 1159
- SMPOpenUIDecimalDisplayAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didChangeDecimal: method 1159
- SMPOpenUIDecimalDisplayModel protocol [Agentry Open UI iOS SDK API] description 1160
- SMPOpenUIDecimalDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1160
- SMPOpenUIDecimalEditAdapter protocol [Agentry Open UI iOS SDK API] description 1160
- SMPOpenUIDecimalEditAdapter protocol [Agentry Open UI iOS SDK API]
 - initWithDecimalEditModel: method 1160
- SMPOpenUIDecimalEditAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didChangeDecimal: method 1161
- SMPOpenUIDecimalEditModel protocol [Agentry Open UI iOS SDK API] description 1161
- SMPOpenUIDecimalEditModel protocol [Agentry Open UI iOS SDK API] maximumValue property 1161
- SMPOpenUIDecimalEditModel protocol [Agentry Open UI iOS SDK API] minimumValue property 1162
- SMPOpenUIDecimalEditModel protocol [Agentry Open UI iOS SDK API]
 - processInputDecimal: method 1161
- SMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1162
- SMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API]
 - initWithDurationDisplayModel: method 1162
- SMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didChangeDuration: method 1162
- SMPOpenUIDurationDisplayAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didChangeFractionalHour: method 1163
- SMPOpenUIDurationDisplayFormat enumeration SMPOpenUIDurationDisplayModel.h file [Agentry Open UI iOS SDK API] 1199
- SMPOpenUIDurationDisplayModel protocol [Agentry Open UI iOS SDK API] description 1163
- SMPOpenUIDurationDisplayModel protocol [Agentry Open UI iOS SDK API] displayFormat property 1163
- SMPOpenUIDurationDisplayModel protocol [Agentry Open UI iOS SDK API] fractionalHourValue property 1163
- SMPOpenUIDurationDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1164
- SMPOpenUIDurationDisplayModel.h file [Agentry Open UI iOS SDK API] SMPOpenUIDurationDisplayFormat enumeration 1199
- SMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API] description 1164
- SMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API]
 - initWithDurationEditModel: method 1164
- SMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didChangeDuration: method 1164
- SMPOpenUIDurationEditAdapter protocol [Agentry Open UI iOS SDK API]

- model:didChangeFractionalHour: method 1165
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] description 1165
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] maximumFractionalHourValue property 1166
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] maximumValue property 1166
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] minimumFractionalHourValue property 1167
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] minimumValue property 1167
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] processInputDuration: method 1165
- SMPOpenUIDurationEditModel protocol [Agentry Open UI iOS SDK API] processInputFractionalHour: method 1166
- SMPOpenUIEmbeddedImageDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1167
- SMPOpenUIEmbeddedImageDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithEmbeddedImageModel: method 1167
- SMPOpenUIEmbeddedImageDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeImage: method 1168
- SMPOpenUIEmbeddedImageDisplayAdapter protocol [Agentry Open UI iOS SDK API] modelDidChangeImageCellSelection: method 1168
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] columns property 1169
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] description 1168
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] highlightSelectedColor property 1169
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] image property 1169
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] imageCellClickedAtRow:andColumn: method 1168
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] isImageCellSelectedAtRow:andColumn: method 1169
- SMPOpenUIEmbeddedImageDisplayModel protocol [Agentry Open UI iOS SDK API] rows property 1169
- SMPOpenUIExternalDataDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1170
- SMPOpenUIExternalDataDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithExternalDataDisplayModel: method 1170
- SMPOpenUIExternalDataDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeExternalData: method 1170
- SMPOpenUIExternalDataDisplayModel protocol [Agentry Open UI iOS SDK API] description 1170
- SMPOpenUIExternalDataDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1171
- SMPOpenUIExternalDataEditAdapter protocol [Agentry Open UI iOS SDK API] description 1171
- SMPOpenUIExternalDataEditAdapter protocol [Agentry Open UI iOS SDK API] initWithExternalDataEditModel: method 1171
- SMPOpenUIExternalDataEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeExternalData: method 1171

- SMPOpenUIExternalDataEditModel protocol
 - [Agentry Open UI iOS SDK API]
 - description 1172
- SMPOpenUIExternalDataEditModel protocol
 - [Agentry Open UI iOS SDK API]
 - processInputExternalData: method 1172
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - agentryShouldDisplayLabel method 1173
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - agentryShouldDisplayValidationFailure method 1174
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - autosizeBehavior method 1174
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - description 1172
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didSetEnabled: method 1175
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didSetHyperlinkEnabled: method 1175
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didSetValid:withValidationFailureText: method 1175
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didSetVisible: method 1176
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:didUpdateLabel: method 1176
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:wantsExtensionString: method 1177
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - model:wantsViewHeightForWidth: method 1177
- SMPOpenUIFieldAdapter protocol [Agentry Open UI iOS SDK API]
 - viewForFrame: method 1178
- SMPOpenUIFieldAdapter.h file [Agentry Open UI iOS SDK API]
- SMPOpenUIAutosizeBehavior
 - enumeration 1198
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - agentryActionEnableState: method 1179
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - agentryString: method 1179
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - autosizing property 1180
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - description 1178
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - enabled property 1181
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - executeAgentryAction: method 1179
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - executeHyperlinkAction method 1180
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - hidden property 1181
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - hyperlinkEnabled property 1181
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - label property 1181
- SMPOpenUIFieldModel protocol [Agentry Open UI iOS SDK API]
 - requestLayoutHeight: method 1180
- SMPOpenUIFieldModel.h file [Agentry Open UI iOS SDK API]
 - SMPOpenUIActionEnableType enumeration 1197
- SMPOpenUIFieldModel.h file [Agentry Open UI iOS SDK API]
 - SMPOpenUIActionResult enumeration 1198
- SMPOpenUIFieldModel.h file [Agentry Open UI iOS SDK API]
 - SMPOpenUIProcessInputReturn enumeration 1200
- SMPOpenUIImage class [Agentry Open UI iOS SDK API]
 - description 1142
- SMPOpenUIImage class [Agentry Open UI iOS SDK API]
 - image property 1142
- SMPOpenUIImage class [Agentry Open UI iOS SDK API]
 - name property 1142

- SMPOpenUIImage class [Agentry Open UI iOS SDK API] position property 1142
- SMPOpenUIImage class [Agentry Open UI iOS SDK API] presentation property 1143
- SMPOpenUIImage.h file [Agentry Open UI iOS SDK API] SMPOpenUIImagePosition enumeration 1199
- SMPOpenUIImage.h file [Agentry Open UI iOS SDK API] SMPOpenUIImagePresentation enumeration 1199
- SMPOpenUIImagePosition enumeration SMPOpenUIImage.h file [Agentry Open UI iOS SDK API] 1199
- SMPOpenUIImagePresentation enumeration SMPOpenUIImage.h file [Agentry Open UI iOS SDK API] 1199
- SMPOpenUIIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1181
- SMPOpenUIIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithIntegerDisplayModel: method 1182
- SMPOpenUIIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeInteger: method 1182
- SMPOpenUIIntegerDisplayModel protocol [Agentry Open UI iOS SDK API] description 1182
- SMPOpenUIIntegerDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1182
- SMPOpenUIIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] description 1183
- SMPOpenUIIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] initWithIntegerEditModel: method 1183
- SMPOpenUIIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeInteger: method 1183
- SMPOpenUIIntegerEditModel protocol [Agentry Open UI iOS SDK API] description 1183
- SMPOpenUIIntegerEditModel protocol [Agentry Open UI iOS SDK API] maximumValue property 1184
- SMPOpenUIIntegerEditModel protocol [Agentry Open UI iOS SDK API] minimumValue property 1184
- SMPOpenUIIntegerEditModel protocol [Agentry Open UI iOS SDK API] processInputInteger: method 1184
- SMPOpenUILabelDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1184
- SMPOpenUILabelDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithLabelDisplayModel: method 1185
- SMPOpenUILabelDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeLabel: method 1185
- SMPOpenUILabelDisplayModel protocol [Agentry Open UI iOS SDK API] description 1185
- SMPOpenUILabelDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1185
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] description 1143
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] dilution property 1144
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] initWithCLLocation: method 1143
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] initWithLatitude:andLongitude:andSatellites:andDilution: method 1143
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] latitude property 1144
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] location property 1144
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] locationWithCLLocation: method 1144
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] locationWithLatitude:andLongitude:andSatellites:andDilution: method 1144
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] longitude property 1145
- SMPOpenUILocation class [Agentry Open UI iOS SDK API] satellites property 1145

- SMOpenUINavigationController class [Agentry Open UI iOS SDK API] valid property 1145
- SMOpenUINavigationControllerDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1186
- SMOpenUINavigationControllerDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithLocationDisplayModel: method 1186
- SMOpenUINavigationControllerDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeLocation: method 1186
- SMOpenUINavigationControllerDisplayModel protocol [Agentry Open UI iOS SDK API] description 1186
- SMOpenUINavigationControllerDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1187
- SMOpenUINavigationControllerEditAdapter protocol [Agentry Open UI iOS SDK API] description 1187
- SMOpenUINavigationControllerEditAdapter protocol [Agentry Open UI iOS SDK API] initWithLocationEditModel: method 1187
- SMOpenUINavigationControllerEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeLocation: method 1187
- SMOpenUINavigationControllerEditModel protocol [Agentry Open UI iOS SDK API] description 1188
- SMOpenUINavigationControllerEditModel protocol [Agentry Open UI iOS SDK API] processInputLocation: method 1188
- SMOpenUIProcessInputReturn enumeration SMOpenUIFieldModel.h file [Agentry Open UI iOS SDK API] 1200
- SMOpenUIStringDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1188
- SMOpenUIStringDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithStringDisplayModel: method 1188
- SMOpenUIStringDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeString: method 1189
- SMOpenUIStringDisplayModel protocol [Agentry Open UI iOS SDK API] allowsCarriageReturn property 1189
- SMOpenUIStringDisplayModel protocol [Agentry Open UI iOS SDK API] description 1189
- SMOpenUIStringDisplayModel protocol [Agentry Open UI iOS SDK API] usesWordWrap property 1189
- SMOpenUIStringDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1190
- SMOpenUIStringEditAdapter protocol [Agentry Open UI iOS SDK API] description 1190
- SMOpenUIStringEditAdapter protocol [Agentry Open UI iOS SDK API] initWithStringEditModel: method 1190
- SMOpenUIStringEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeString: method 1190
- SMOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] description 1191
- SMOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] isPasswordInput property 1191
- SMOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] maxLength property 1191
- SMOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] minLength property 1192
- SMOpenUIStringEditModel protocol [Agentry Open UI iOS SDK API] processInputString: method 1191
- SMOpenUITimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1192
- SMOpenUITimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithTimeDisplayModel: method 1192
- SMOpenUITimeDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeTime: method 1192
- SMOpenUITimeDisplayModel protocol [Agentry Open UI iOS SDK API] description 1193

- SMPOpenUITimeDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1193
- SMPOpenUITimeEditAdapter protocol [Agentry Open UI iOS SDK API] description 1193
- SMPOpenUITimeEditAdapter protocol [Agentry Open UI iOS SDK API] initWithTimeEditModel: method 1193
- SMPOpenUITimeEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeTime: method 1194
- SMPOpenUITimeEditModel protocol [Agentry Open UI iOS SDK API] description 1194
- SMPOpenUITimeEditModel protocol [Agentry Open UI iOS SDK API] processInputTime: method 1194
- SMPOpenUIUnsignedIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] description 1194
- SMPOpenUIUnsignedIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] initWithUnsignedIntegerDisplayModel: method 1195
- SMPOpenUIUnsignedIntegerDisplayAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeUnsignedInteger: method 1195
- SMPOpenUIUnsignedIntegerDisplayModel protocol [Agentry Open UI iOS SDK API] description 1195
- SMPOpenUIUnsignedIntegerDisplayModel protocol [Agentry Open UI iOS SDK API] value property 1195
- SMPOpenUIUnsignedIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] description 1196
- SMPOpenUIUnsignedIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] initWithUnsignedIntegerEditModel: method 1196
- SMPOpenUIUnsignedIntegerEditAdapter protocol [Agentry Open UI iOS SDK API] model:didChangeUnsignedInteger: method 1196
- SMPOpenUIUnsignedIntegerEditModel protocol [Agentry Open UI iOS SDK API] description 1196
- SMPOpenUIUnsignedIntegerEditModel protocol [Agentry Open UI iOS SDK API] maximumValue property 1197
- SMPOpenUIUnsignedIntegerEditModel protocol [Agentry Open UI iOS SDK API] minimumValue property 1197
- SMPOpenUIUnsignedIntegerEditModel protocol [Agentry Open UI iOS SDK API] processInputUnsignedInteger: method 1197
- SMPPProcessInputReturn enumeration [Agentry Open UI Windows SDK API] 1257
- startup() method
 - Server class [Agentry Java System Connection API API] 767
- Steplet class [Agentry Java System Connection API API]
 - _session variable 785
 - description 778
 - doSteplet() method 782
 - getNotificationText() method 783
 - getNotificationTitle() method 783
 - getOkButtonLabel() method 783
 - getReturnData() method 784
 - getSession() method 784
 - notificationText() method [deprecated] 785
 - notificationTitle() method [deprecated] 785
 - okButtonLabel() method [deprecated] 785
 - Steplet(FetchSession) constructor 781
 - Steplet(PushSession) constructor 781
 - Steplet(PushUserSession) constructor 781
 - Steplet(ServiceEventSession) constructor 782
 - Steplet(TransactionSession) constructor 781
- Steplet(FetchSession) constructor
 - Steplet class [Agentry Java System Connection API API] 781
- Steplet(PushSession) constructor
 - Steplet class [Agentry Java System Connection API API] 781
- Steplet(PushUserSession) constructor
 - Steplet class [Agentry Java System Connection API API] 781
- Steplet(ServiceEventSession) constructor
 - Steplet class [Agentry Java System Connection API API] 782

Index

- Steplet(TransactionSession) constructor
 - Steplet class [Agentry Java System Connection API API] 781
 - StepletAbortException class [Agentry Java System Connection API API]
 - description 786
 - StepletAbortException(String) constructor 787
 - StepletAbortException(String) constructor
 - StepletAbortException class [Agentry Java System Connection API API] 787
 - StepletStopException class [Agentry Java System Connection API API]
 - description 787
 - StepletStopException(String) constructor 788
 - StepletStopException(String) constructor
 - StepletStopException class [Agentry Java System Connection API API] 788
 - StringDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1048
 - StringDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1101
 - StringEditAdapter class [Agentry Open UI Android SDK API]
 - description 1051
 - StringEditModel interface [Agentry Open UI Android SDK API]
 - description 1104
 - StringValue property
 - [Agentry Open UI Windows SDK API] 1241
 - supportsAction
 - propertySMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] 1150
 - SycloCalendar class [Agentry Java System Connection API API]
 - description 788
 - getInvalidTimeAndDate() method 792
 - isInvalidTimeAndDate() method 792
 - isInvalidTimeAndDate(GregorianCalendar) method 792
 - SycloCalendar() constructor 790
 - SycloCalendar(GregorianCalendar, Locale) constructor 790
 - SycloCalendar(GregorianCalendar) constructor 789
 - SycloCalendar(int, int, int, int, int, int) constructor 790
 - SycloCalendar(int, int, int, int, int) constructor 791
 - SycloCalendar(int, int, int) constructor 791
 - SycloCalendar(Locale) constructor 791
 - SycloCalendar(TimeZone, Locale) constructor 791
 - SycloCalendar(TimeZone) constructor 792
 - SycloCalendar() constructor
 - SycloCalendar class [Agentry Java System Connection API API] 790
 - SycloCalendar(GregorianCalendar) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 789
 - SycloCalendar(int, int, int, int, int, int) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 790
 - SycloCalendar(int, int, int, int, int) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 791
 - SycloCalendar(int, int, int) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 791
 - SycloCalendar(Locale) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 791
 - SycloCalendar(TimeZone, Locale) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 791
 - SycloCalendar(TimeZone) constructor
 - SycloCalendar class [Agentry Java System Connection API API] 792
- ## T
- throwException() method
 - Server.LoginEnumeration enum [deprecated] [Agentry Java System Connection API API] 747
 - TimeDisplayAdapter class [Agentry Open UI Android SDK API]
 - description 1053
 - TimeDisplayModel interface [Agentry Open UI Android SDK API]
 - description 1107

timedOut() method
 User class [Agentry Java System Connection API API] 805
 TimeEditAdapter class [Agentry Open UI Android SDK API]
 description 1055
 TimeEditModel interface [Agentry Open UI Android SDK API]
 description 1108
 TimeValue property
 [Agentry Open UI Windows SDK API] 1214
 ToBoolean() method
 [Agentry Open UI Windows SDK API] 1251
 ToDate() method
 [Agentry Open UI Windows SDK API] 1252
 ToDateTime() method
 [Agentry Open UI Windows SDK API] 1252
 ToDouble() method
 [Agentry Open UI Windows SDK API] 1252
 ToInt() method
 [Agentry Open UI Windows SDK API] 1253
 toString() method
 UserLogRecord class [Agentry Java System Connection API API] 659
 ToString() method
 [Agentry Open UI Windows SDK API] 1253
 ToTime() method
 [Agentry Open UI Windows SDK API] 1253
 ToUInt() method
 [Agentry Open UI Windows SDK API] 1253
 TransactionSession class [Agentry Java System Connection API API]
 beginTransaction() method 795
 description 793
 endTransaction() method 795
 TransactionSession(String, Server, SessionData, User) constructor 794
 TransactionSession(String, Server, SessionData, User) constructor
 TransactionSession class [Agentry Java System Connection API API] 794

U

update(GregorianCalendar) method
 User class [Agentry Java System Connection API API] 805
 updateLabel(String) method
 [Agentry Open UI Android SDK API] 1036

User class [Agentry Java System Connection API API]
 _name variable 805
 backendTimeAndDate() method [deprecated] 799
 beginChangePassword() method 800
 changePassword(String, String) method 800
 changePasswordFailed(StringBuffer) method 801
 changePasswordSessionAborted() method 801
 debug(String) method 801
 description 795
 endChangePassword() method 802
 getName() method 802
 getSystemConnectionTime() method 802
 getTimeZone(StringBuffer) method [deprecated] 803
 loggedIn() method 803
 loggedOut() method 803
 reLoggedIn() method 804
 revalidate(String) method 804
 timedOut() method 805
 update(GregorianCalendar) method 805
 User(String) constructor 799
 User.ChangePasswordResult enum [Agentry Java System Connection API API]
 ChangePassword_Blocked variable 798
 ChangePassword_Failure variable 798
 ChangePassword_NotHandled variable 799
 ChangePassword_Success variable 799
 description 797
 getValue() method 798
 User(String) constructor
 User class [Agentry Java System Connection API API] 799
 UserLogger class [Agentry Java System Connection API API]
 description 659
 getUser() method 660
 getUserLogger(String, String, User) method 660
 getUserLogger(String, User) method 660
 log(LogRecord) method 661
 UserLogRecord class [Agentry Java System Connection API API]
 description 652
 equals(Object) method 655
 getLevel() method 655

- getLoggerName() method 655
- getMessage() method 656
- getMillis() method 656
- getParameters() method 656
- getResourceBundle() method 656
- getResourceBundleName() method 656
- getSequenceNumber() method 656
- getSourceClassName() method 656
- getSourceMethodName() method 656
- getThreadID() method 657
- getThrown() method 657
- getUser() method 657
- hashCode() method 657
- setLevel(Level) method 657
- setLoggerName(String) method 657
- setMessage(String) method 657
- setMillis(long) method 658
- setParameters(Object[]) method 658
- setResourceBundle(ResourceBundle) method 658
- setResourceBundleName(String) method 658
- setSequenceNumber(long) method 658
- setSourceClassName(String) method 658
- setSourceMethodName(String) method 658
- setThreadID(int) method 658
- setThrown(Throwable) method 659
- toString() method 659
- UserLogRecord(User, Level, String) constructor 655
- UserLogRecord(User, LogRecord) constructor 655
- UserLogRecord(User, Level, String) constructor
- UserLogRecord class [Agentry Java System Connection API API] 655
- UserLogRecord(User, LogRecord) constructor
- UserLogRecord class [Agentry Java System Connection API API] 655
- usesWordWrap
 - propertySMPOpenUIStringDisplayModel protocol [Agentry Open UI iOS SDK API] 1189
- utility package [Agentry Java System Connection API API]
 - description 647
- V**
- valid propertySMPDataAPILocationProtocol protocol [Agentry Open UI iOS SDK API] 1136
- valid propertySMPOpenUILocation class [Agentry Open UI iOS SDK API] 1145
- Value property
 - [Agentry Open UI Windows SDK API] 1214
- value propertySMPOpenUIBooleanDisplayModel protocol [Agentry Open UI iOS SDK API] 1146
- value propertySMPOpenUIButtonDisplayModel protocol [Agentry Open UI iOS SDK API] 1150
- value
 - propertySMPOpenUIDateAndTimeDisplayModel protocol [Agentry Open UI iOS SDK API] 1155
- value propertySMPOpenUIDateDisplayModel protocol [Agentry Open UI iOS SDK API] 1157
- value propertySMPOpenUIDecimalDisplayModel protocol [Agentry Open UI iOS SDK API] 1160
- value propertySMPOpenUIDurationDisplayModel protocol [Agentry Open UI iOS SDK API] 1164
- value
 - propertySMPOpenUIExternalDataDisplayModel protocol [Agentry Open UI iOS SDK API] 1171
- value propertySMPOpenUIIntegerDisplayModel protocol [Agentry Open UI iOS SDK API] 1182
- value propertySMPOpenUILabelDisplayModel protocol [Agentry Open UI iOS SDK API] 1185
- value propertySMPOpenUILocationDisplayModel protocol [Agentry Open UI iOS SDK API] 1187
- value propertySMPOpenUIStringDisplayModel protocol [Agentry Open UI iOS SDK API] 1190
- value propertySMPOpenUITimeDisplayModel protocol [Agentry Open UI iOS SDK API] 1193
- value
 - propertySMPOpenUIUnsignedIntegerDisplayModel protocol [Agentry Open UI iOS SDK API] 1195
- value variable
 - DataTableObject class [Agentry Java System Connection API API] 688

value() method [deprecated]
 DataTableObject class [Agentry Java System
 Connection API API] 688
 valueChanged(AgentryLocation) method
 [Agentry Open UI Android SDK API] 1046,
 1048
 valueChanged(boolean) method
 [Agentry Open UI Android SDK API] 998,
 1001
 valueChanged(double) method
 [Agentry Open UI Android SDK API] 1015,
 1017
 valueChanged(GregorianCalendar) method
 [Agentry Open UI Android SDK API] 1005,
 1008, 1010, 1012, 1055, 1057
 valueChanged(int) method
 [Agentry Open UI Android SDK API] 1020,
 1023, 1039, 1041

valueChanged(String) method
 [Agentry Open UI Android SDK API] 1028,
 1030, 1043, 1050, 1053
 viewForFrame: methodSMPOpenUIFieldAdapter
 protocol [Agentry Open UI iOS SDK
 API] 1178

W

willRebuildTable() method
 ComplexTable< CTOBJ > class [Agentry Java
 System Connection API API] 685
 WordWrap property
 [Agentry Open UI Windows SDK API] 1241

