Migration

SAP Mobile Platform 3.0

DOCUMENT ID: DC-01-0300-01
LAST REVISED: November 2013

Contents

Migration

You can migrate SAP® Mobile Platform 2.*x* Agentry applications, MBO applications, and OData applications to SAP Mobile Platform 3.0.

Although there are compelling reasons to upgrade to SAP Mobile Platform, version 3.0, and take advantage of the new features, migrating your applications may not always be the correct solution. If you want to migrate, you can choose between three high-level strategies: redeploy, redevelop, or redesign.

Redeploy
You can deploy a number of application types to SAP Mobile Platform Server version 3.0: Agentry, Mobiliser, and SAP Mobile Platform version 2.3. Agentry and Mobiliser applications are intrinsically supported in the main platform runtime.

No built-in upgrade support exists to migrate applications from version 2.*x* to version 3.0, so you must set up parallel infrastructure for a production environment, with a scheduled switchover. This may require you to redeploy application clients.

Redevelop
You may need to redevelop SAP Mobile Platform 2.*x* Mobile Business Object (MBO) applications, both native offline-able applications, and hybrid Web container applications using online MBOs. You may also need to redevelop applications that use HTTP and on-device portal (ODP).

The general theme of this approach is to keep certain aspects of an application, such as the existing client user interface and services, and replace the underlying communication and mobile-enabling services with the new SAP Mobile Platform 3.0 SDKs and services.

Redesign
Sometimes when taking into consideration the cost-benefit analysis, with respect to applying migration strategies, it may be more cost effective to start from the beginning. In such cases, you need only consider migrating back-end services to be OData centric services, with Delta Token capabilities to support message-based delta synchronization

Migrating Native OData Applications

You can migrate SAP Mobile Platform 2.x native OData applications to SAP Mobile Platform 3.0.

Note:

- The Messaging Channel is not supported for iOS and Android applications. The Android and iOS applications work only on the REST SDK, included as part of the standard SAP Mobile Platform SDK installer.
- The Messaging Channel is supported for BlackBerry, as part of MBO toolkit installer.
- Applications require re-registration using REST SDK when moving from SAP Mobile Platform 2.x to 3.0.

OData SDK Type in 3.0	iOS	Android	BlackBerry
Messaging Channel	Not supported	Not supported	Supported (part of MBO toolkit installer)
REST SDK	Supported (standard SDK installer)	Supported (standard SDK installer)	Not supported

Migrating iOS Native OData Applications

Migrate your iOS OData application from version 2.x to SAP Mobile Platform version 3.0.

Overview

This section covers migration of a REST based application from SAP Mobile Platform 2.3.x to version 3.x. The following aspects of a REST based application are covered:

- Registration
- Request-Response (data fetch)
- Parsing

Registration

There is no change in code and no refactoring is required to migrate an application from 2.3.x to 3.x. Existing applications based on 2.x will continue to work against the SAP Mobile Platform 3.0 OData SDK post migration, without any code changes, with just a rebuild. The deprecated class details in this section is applicable for new application development, where it is requested or mandatorily required to use the new APIs. To use OData offline and other features such as batch processing, the new classes are mandatory.

Request-Response

With 3.x, the old SDM API has been deprecated and a new Request API is introduced for uniformity in API nomenclature. With this change, all SDM* classes and methods have been replaced with class names removing the SDM tag. The following table lists the old and new class names. The method names mostly remain the same, unless specified otherwise in this section. The class names are listed first, followed by the header files to which the classes belong.

Table 1. List of Refactored Classes

Class name in 2.3.x version (old - deprecated)	Class name in 3.0 version (new - refactored)
SDMHttpRequest (SDMHttpRequest.h)	Request (Request.h)
SDMRequestBuilder (SDMRequestBuilder.h)	RequestBuilder (RequestBuilder.h)
SDMDownloadCache (SDMDownloadCache.h)	DownloadCache (DownloadCache.h)
SDMFormDataRequest (SDMFormDataRequest.h)	FormDataRequest (FormDataRequest.h)
SDMNetworkQueue (SDMNetworkQueue.h)	NetworkQueue (NetworkQueue.h)
<SDMRequesting>** (SDMRequesting.h)	<Requesting> (Requesting.h)
<SDMCacheDelegate> (SDMCacheDelegate.h)	<CacheDelegate> (CacheDelegate.h)
<SDMHttpRequestDelegate> (SDMHttpRequestDelegate.h)	<RequestDelegate> (RequestDelegate.h)
<SDMProgressDelegate> (SDMProgressDelegate.h)	<ProgressDelegate> (ProgressDelegate.h)
SDMConnectivityException (SDMConnectivityException.h)	ConnectivityException (ConnectivityException.h)

Note: ** corresponds to the protocol refactoring

Parser

All SDM* classes have been refactored with OData* classes. The following table lists the refactored classes and protocols. All method names remain the same. In this section, all public header files are listed first with their corresponding refactored names. If the class names in the file are different and the file contains multiple class names, different legends have been provided accordingly.

Table 2. List of Refactored Classes

Class name in 2.3.x version (old - deprecated)	Class name in 3.0 version (new - refactored)
SDMFunctionImportResultParser.h **	ODataFunctionImportResultParser.h
SDMGenericParser.h **	ODataGenericParser.h
SDMOData.h *	OData.h
SDMODataCollection.h **	ODataCollection.h

Class name in 2.3.x version (old - deprecated)	Class name in 3.0 version (new - refactored)
SDMODataDataParser.h **	ODataDataParser.h
SDMODataEntitySchema.h **	ODataEntitySchema.h
SDMODataEntry.h **	ODataEntry.h
SDMODataError.h **	ODataError.h
SDMODataErrorXMLParser.h **	ODataErrorXMLParser.h
SDMODataFunctionImport.h #	ODataFunctionImport.h
SDMODataFunctionImportParameter	ODataFunctionImportParameter
SDMODataFunctionImport	ODataFunctionImport
SDMODataIconInfo.h **	ODataIconInfo.h
SDMODataLink.h #	ODataLink.h
SDMODataLink	ODataLink
SDMODataRelatedLink	*ODataRelatedLink*
SDMODataMediaResourceLink	*ODataMediaResourceLink*
SDMODataActionLink	ODataActionLink
SDMODataMetaDocumentParser.h **	ODataMetaDocumentParser.h
SDMODataProperty.h *	ODataProperty.h
SDMODataPropertyInfo.h **	ODataPropertyInfo.h
SDMODataPropertyValueFactory.h **	ODataPropertyValueFactory.h
SDMODataPropertyValues.h #	ODataPropertyValues.h
SDMODataPropertyValueObject	*ODataPropertyValueObject*
SDMODataPropertyValueInt	*ODataPropertyValueInt*
SDMODataPropertyValueString	*ODataPropertyValueString*
SDMODataPropertyValueComplex	*ODataPropertyValueComplex*
SDMODataPropertyValueDateTime	*ODataPropertyValueDateTime*
SDMODataPropertyValueBoolean	*ODataPropertyValueBoolean*
SDMODataPropertyValueGuid	*ODataPropertyValueGuid*

Class name in 2.3.x version (old - deprecated)	Class name in 3.0 version (new - refactored)
SDMODataPropertyValueBinary	*ODataPropertyValueBinary*
SDMODataPropertyValueSingle	*ODataPropertyValueSingle*
SDMODataPropertyValueDouble	*ODataPropertyValueDouble*
SDMODataPropertyValueDecimal	*ODataPropertyValueDecimal*
SDMDuration	*ODataDuration*
SDMODataPropertyValueTime	*ODataPropertyValueTime*
SDMODataPropertyValueTimeOffset	*ODataPropertyValueTimeOffset*
SDMODataSchema.h **	ODataSchema.h
SDMODataServiceDocument.h **	ODataServiceDocument.h
SDMODataServiceDocumentParser.h **	ODataServiceDocumentParser.h
SDMODataWorkspace.h **	ODataWorkspace.h
SDMODataXMLBuilder.h #	ODataXMLBuilder.h
SDMODataEntryXML	*ODataEntryBody*
SDMOpenSearchDescription.h #	OpenSearchDescription.h
SDMOpenSearchDescriptionURLTemplate	*OpenSearchDescriptionURLTemplate*
SDMOpenSearchDescription	*OpenSearchDescription*
SDMOpenSearchDescriptionXMLParser.h **	OpenSearchDescriptionXMLParser.h
<SDMParserDelegate.h> ##	<ODataParserDelegate.h>
SDMParserException.h **	ODataParserException.h
SDMPerformanceUtil.h **	PerformanceUtil.h
SDMSubscriptionXMLBuilder.h #	ODataSubscriptionXMLBuilder.h
SDMSubscriptionInfo	*ODataSubscriptionInfo*
SDMSubscriptionXML	*ODataSubscriptionXML*

- * Indicates a header file and not class definition. Renaming the header file in the `#import` statement is sufficient.
- ** Indicates that the class name is same as the name of the header file. For example : `SDMODataError.h` file has a class definition whose name is `SDMODataError`.

- # Indicates that these header files have multiple class definitions in the header file and are listed below the same, italicized.
- ## Indicates that this corresponds to protocol definition in iOS.

Note: SAP recommends you to update the APIs to the newly re-factored APIs listed. The deprecated SDM APIs are supported for backward compatibility.

Migrating Android Native OData Applications

Migrate your Android OData application from version 2.x to SAP Mobile Platform version 3.0.

There is no change in code and no refactoring is required to migrate an application from 2.3.x to 3.x. Existing applications based on 2.x will continue to work against the SAP Mobile Platform 3.0 OData SDK post migration, without any code changes, with just a rebuild. The deprecated class details in this section is applicable for new application development, where it is requested or mandatorily required to use the new APIs. To use OData offline and other features such as batch processing, the new classes are mandatory.

Table 3. List of Refactored APIs

API in Version 2.x	API in Version 3.x
buildSDMODataEntryXML	buildODataEntryRequestBody
parseSDMODataServiceDocumentXML	parseODataServiceDocument
parseSDMODataSchemaXML	parseODataSchema
parseSDMODataEntriesXML	parseODataEntries
parseSDMODataOpenSearchDescriptionXML	parseODataOpenSearchDescription
parseSDMODataErrorXML	parseODataError
parseFunctionImportResultXML	parseFunctionImportResult

Note: SAP recommends you to update the APIs to the newly refactored APIs listed. The deprecated SDM APIs are supported for backward compatibility.

Migrating MBO Applications

You can migrate SAP Mobile Platform 2.*x* Mobile Business Object (MBO) applications to SAP Mobile Platform 3.0 by redeveloping the MBOs.

Note: For complete details about migrating SAP Mobile Platform 2.*x* MBO applications to 3.0, see the documentation in the Migration folder on the SAP Mobile Platform 3.0 Ramp-Up Knowledge Transfer (RKT) Web site.

MBO and OData Architectural Differences

There are a considerable number of architectural differences between SAP Mobile Platform versions 2.*x* and 3.*x*.

This table summarizes the differences between SAP Mobile Platform versions 2.*x* and 3.0 architectures.

Version 3.0	Version 2.*x*	Notes
No caching database	Cache database	In a typical SAP Mobile Platform 2.*x* production environment, separate hardware runs the cache database, which is used for differencing and replication-based MBO synchronizations.
Settings are stored in database	Settings are stored in database and files	In version 2.*x*, some settings are stored in the cluster database, but most settings are stored in files that must be synchronized across the cluster. **Note:** SAP Mobile Platform 3.0 does not provide cluster support.
Runs in SAP Light Java Server	Mix of x86 and Java runtime	The version 2.*x* servers runs only on Windows-based machines. Because SAP Mobile Platform 3.0 runs in SAP Light Java Server, you can install it on a range of Linux and Unix servers as well.
Service packages are managed by OSGi	Custom service and package management	This is also a great differentiator making it much easier to manage the middleware services on an SAP Mobile Platform 3.0 server. You can also install custom service packages, allowing you to deploy services to mobile platforms. These packages are referred to as features within the platform, and administrators can manage them.
HTTP/HTTPS	Custom protocols	One goal of SAP Mobile Platform3 is to standardize on network protocols
Support for standard reverse proxies	Some support for reverse proxies, but SAP recommends Relay Server	
Integration services are deferred to NetWeaver Gateway or Gateway For Java	Integration is part of MBO design	

MBO and OData SDK Differences

The functionality of SAP Mobile Platform 2.*x* MBO applications differs greatly from SAP Mobile Platform 3.0 OData SDKs.

Note: SAP Mobile Platform 3.0 implements OData version 2, and includes delta token support from version 4.

The OData SDK is primarily responsible for user on-boarding and processing OData requests. Many features that applications require, for example, reading and updating data from back-end systems, are features of the OData standard; SAP Mobile Platform 3.0 implements a version of this standard. OData does not define how integration is done, but defines contracts between clients and servers using Entity Data Models.

MBO primary functions are to define:

- Integration into back-end systems
- How data is cached and synchronized to devices
- Data models for applications

To compare features, we must compare the functional capability of MBOs with the functional capability of SAP Mobile Platform 3.0 and the OData standard itself.

These are the functional mappings between SAP Mobile Platform 2.*x* MBOs and the SAP Mobile Platform 3.0 OData SDK:

MBO	OData	Description
Defines integration	N/A	**Note:** OData defines the contract. Integration is performed with other tools, such as Netweaver Gateway or Gateway for Java.
Defines middleware caching	N/A	For performance reasons, SAP Mobile Platform 3.0 does not implement middleware caching.
Defines synchronization with delta calculation using a cache	Uses the delta token approach for delta synchronizations.	Since SAP Mobile Platform 3.0 does not use a middleware cache, enterprise services must implement delta tracking.
Defines data models	Defines entity data models (EDM)	
Defines relationships	Defines associations	

MBO	OData	Description
Defines client-side object relational models via code generation	N/A	Although there are many tools available to generate client-side object relational models for OData EDMs, no such tool is included with the SAP Mobile Platform 3.0 SDK.
Defines a client-side relational database for offline lookup	OData cache requires in-memory lookup.	The SAP Mobile Platform 3.0 SDK does not include any tools that provide a client-side relational database for offline lookup. You can store results from OData queries in a cache, which must be loaded into memory to search.
Offline `Find By` queries	N/A	No direct mapping exists. The document cache provided by the OData SDK does not allow you to query the data using SQL. OData usually represents aggregated data and cannot be treated as normalized data.
Offline custom queries	N/A	
Online `Find By` queries	HTTP GET, `$filter`	
Multiple MBO sync with sync groups	HTTP GET, `$filter`, `$expand`	Not a direct mapping; `$expand` can only retrieve associated entities.
Create	HTTP POST	
Multilevel inserts	HTTP POST with `$batch`	
Update	HTTP PUT, PATCH	
Delete	HTTP DELETE	
Static libraries for user onboarding	HTTP API for creating application connections	
Push notifications via dynamic circuit networks and target-change notifications	N/A	Push notifications must be handled manually

MBO	OData	Description
On device relational database	N/A	No direct mapping exists. The OData SDK provides a document cache, but you cannot use this to replace a normalized relational database.

Comparing MBO Models to OData Entity Data Models

Converting an MBO-based application to an OData application requires that you first develop the OData entity data model (EDM). MBO models are similar to OData EDMs, both are entity-relationship models.

In SAP Mobile Platform 2.*x*, you can use the Mobile Application Diagram to create MBOs from different data sources, such as databases, Web services, and Business Application Programming Interfaces.

In SAP Mobile Platform 3.0, the entity data model defines the service contract, which is independent of how the service is implemented. A number of different techniques exist for using SAP tools to implement services for contracts.

The following table illustrates the general mappings between structures in an MBO diagram and structures in an OData EDM diagram. There is not always a direct mapping from MBO model elements to OData model elements. In MBOs, create, read, update, and delete (CRUD) operations are synchronized database transactions. Since an OData EDM is used as a contract in an HTTP REST pattern, CRUD operations are performed using HTTP verbs, such as GET, POST, PUT, DELETE and MERGE.

MBO Model	OData Model	Notes
MBO Collection	Entity Collection	
MBO	Entity	
Attributes	Properties	
CRUD operations	N/A	CRUD operations are not part of the OData model; OData follows the HTTP REST pattern for CRUD.
Other operations	Function imports	
Relationship	Association, navigation	Associations are equivalent to MBO relationships, and incorporate cardinality. To use OData associations, you must define additional navigation entities.

Migrating of Agentry Applications

Information on migrating your Agentry-based mobile application from either Agentry Mobile Platform 6.0.x or from SAP Mobile Platform 2.3 to SAP Mobile Platform 3.0 is provided here, including step-by-step instructions and overall environment needs.

In general this process involves the installation of the SAP Mobile Platform Server 3.0, the installation of the Agentry Editor plug-in provided in the SAP Mobile Platform SDK 3.0, and access to the currently installed mobile application from a previous platform version. This access can be to the published version of the application, or to the exported Agentry application project form the Agentry Editor.

In addition, for applications with a Java system connection, there is additional information provided on preparing the mobile application for migration and deployment to the SAP Mobile Platform Server 3.0.

Preparing Agentry Applications With Java System Connections

When migrating an Agentry application to the SAP® Mobile Platform Server 3.0, it is necessary to account for changes within the architecture that directly affect how the Java synchronization logic of the mobile application processes file references. Note if your application does not include a Java system connection, the information in this section can be ignored.

In releases of the SAP® Mobile Platform prior to 3.0, as well as in releases of the Agentry Mobile Platform versions 6.0.x and earlier, the Java synchronization logic could reference file resources relative to the installation location of the Agentry Server; or within the SAP® Mobile Platform Runtime, relative to the directory created when an Agentry application was defined within the runtime environment. The current directory when the Java logic was processed was always the location of the Agentry Server and file paths to resources to be loaded or processed by the Java logic were relative. For example, the path `.\JavaBe.ini` would be valid for a file named `JavaBe.ini`, located within folder where thel Agentry Server was located. In SAP® Mobile Platform version 3.0, the current directory when processing Java logic is no longer the location of the Agentry Server.

This change is the result of the architectural changes made to the Agentry Server within the SAP® Mobile Platform Server. When migrating an Agentry mobile application to the SAP® Mobile Platform version 3.0, it is therefore necessary to make changes to the Java logic and/or those file resources referenced by the Java logic in order to properly migrate the application.

The following information provides the two methods in which this can be handled. The first is to place the referenced files in the main SAP® Mobile Platform Server 3.0 installation directory. The second is to modify the Java synchronization logic for the application to use the method `findConifgurationFile` within the Agentry Java API class

`com.syclo.agentry.Server`. Of these methods it is the second, modification of the Java synchronization logic, which is recommended.

Configuration File Placement for Java Logic Access

One option while migrating the Agentry application to the SAP® Mobile Platform version 3.0 is to move all configuration files, along with any other resource files referenced by the Java logic into the main installation location of the SAP® Mobile Platform Server 3.0. When files are referenced by a relative path, this location is returned. This requires the directory structure to match that of the Agentry Server as implemented within the Agentry Mobile Platform 6.0.x environment, or within the SAP® Mobile Platform Runtime Environment 2.3.

As an example, if a resource file is located in the directory `C:\Agentry\ServerProd\CustomerDLLs\myResource.dll`, the same `CustomerDLLs` directory must be created in the main SAP® Mobile Platform Server 3.0 directory.

This method is not recommended as it introduces potential maintenance challenges that can be avoided by modifying the Java synchronization logic.

Modification of File References Within Java Synchronization Logic

The other, and recommended option for migrating an Agentry mobile application to SAP® Mobile Platform 3.0 is to modify the Java synchronization logic. This modification consists of changing any references to configuration files or other resource files within the logic. In each case where such a file is referenced, it should be replaced with a call to the method `com.syclo.agentry.Server.findConfigurationFile`. This method takes either the file name or relative path and file name as its parameter and returns the full path to the file. This return value can then be used in the same manner in which the previous relative file path was used in order to process a configuration file.

When making this change it is necessary to first import all development resources, including the Java logic, as well as the Agentry application project into the Agentry Editor plug-in provided in the SAP Mobile SDK 3.0. It is also necessary to update the AJ-API to the version provided with the SAP Mobile SDK 3.0. This API is contained in the `Agentry-v5.jar` file provided with the SAP Mobile SDK installation.

Migrating Agentry Applications to SAP® Mobile Platform 3.0

Prerequisites

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

- If the mobile application being migrated contains a Java system connection the procedure on “Preparing Agentry Applications With Java System Connections” must be performed prior to this procedure.
- You must have access to the currently installed mobile application to be migrated. The source from which the application will be migrated dictates the components to which you need access. You will always need access to the Agentry Server or SAP® Mobile Platform environment upon which the mobile application is deployed. Additionally, if importing

the application definitions from the Agentry application project you will also need to the Agentry Editor and Eclipse workspace in which the Agentry application project exists.

- The SAP® Mobile Platform 3.0 must be installed and an Agentry application defined within the Management Cockpit.
- The Agentry Editor plug-in as provided in the SAP Mobile SDK 3.0 must be installed to Eclipse.

Task

This procedure describes the steps necessary to migrate a mobile application built and/or currently deployed on the Agentry Mobile Platform version 6.0.x or earlier; or currently deployed in SAP® Mobile Platform 2.3 and is built on the Agentry archetype within that system. The procedure to migrate to SAP® Mobile Platform 3.0 is the same.

In this procedure the following terms are used and should be noted to fully understand the procedure:

- **Import Source:** The source mobile application or application project within the Agentry Mobile Platform 6.0.x or SAP® Mobile Platform 2.3 that is to be migrated to SAP® Mobile Platform 3.0. This can be an Agentry application export file generated by exporting the project from the Agentry Editor, or the mobile application as it exists on the Agentry Server.

1. Verify the Agentry Server within the SAP® Mobile Platform Server 3.0 is not running. If it is running, stop the service using the Management Cockpit before proceeding.
2. If the import source of the mobile application is an Agentry application project within the Eclipse workspace, use the Agentry Editor plug-in within Eclipse to export the entire project. This results in a file with extension `.agx` or `.agxz`.
3. Open the Eclipse installation containing the Agentry Editor plug-in as provided in the SAP Mobile SDK 3.0. Once started, open the Agentry perspective.
4. Begin the import process by either right clicking within the Package Explorer view and selecting **Import...** or by selecting the menu item **File | Import...**
5. Expand the item Agentry Project in the dialog now displayed. Then select the proper import source from the options listed. Continue the import by clicking through import wizard and selecting the proper options.

   Once the import process is complete, the new Agentry application project for the mobile application exists in the Eclipse workspace and is listed in the Package Explorer view.
6. If your mobile application includes a Java system connection, the associated Java project should now be migrated to the Eclipse workspace. If making changes to the Java logic using the `findConfigurationFile` method, described in the section “Preparing Agentry Applications With Java System Connections,” these changes should be made now and the Java project rebuilt.

7. Next publish the application to the SAP® Mobile Platform Server 3.0, into the configuration directory of the Agentry application defined there; typically the directory within the SAP® Mobile Platform Server installation location is as follows:

   ```
   configuration
   \com.sap.mobile.platform.server.agentry.application
   ```

8. Next all other resources residing in either the Agentry Server for version 6.0.x, or in the Agentry application directory within the SAP® Mobile Platform Server 2.3 directory structure, which are specific to the mobile application must be copied to the SAP® Mobile Platform Server 3.0. The specific files are different from mobile application to the next. In general the following are the types of resources to be moved, and which should be collected in this step in preparation for move. TH next step specifies where the files should be placed within the SAP® Mobile Platform Server 3.0 installation:

   - Application-specific configuration files, plus the Agentry.ini file
   - Application-specific Java resources, including `.jar` files, but **do not include** the `Agentry-v5.jar` or `Agentry-v4.jar` (if present) files found in the Java folder of the server's installation. Any other resources found here should be included as they are likely to be application-specific. NOTE: If changes were made to the Java logic, the resulting resources (Jar files and similar) should be those moved to the SAP® Mobile Platform Server 3.0 installation, not those in the previous implementation of the application.
   - Application-specific DLL files, but **do not include** DLL's provided with a standard Agentry Server installation.
   - The contents of the `sql` directory under the Agentry Server's installation location. All files in this directory can be safely added to the ZIP archive.
   - Any other files know to be a part of the mobile application but not provided with a standard Agentry Server installation.

9. Once all file resources have been collected or their locations noted, they must be placed within the proper location within the SAP® Mobile Platform Server 3.0. The following locations each pertain to different file resource types:

   a) The `Agentry.ini` file is copied to two locations within the SAP® Mobile Platform Server 3.0 location. First, it should be copied and **renamed** to `AgentryServer.ini` and placed within the directory `Server\configuration\com.sap.mobile.platform.server.agentry`.

   b) A second copy of the `Agentry.ini` file, with the name remaining `Agentry.ini`, is copied to the location `Server\configuration\com.sap.mobile.platform.server.agentry.application`.

   c) All Java `.jar` files are copied to the location `Server\configuration\com.sap.mobile.platform.server.agentry.application\Java`

   d) All SQL files or command scripts are copied to the location `Server\configuration\com.sap.mobile.platform.server.agentry.application\sql`

e) All application-specific configuration files **not referenced** by Java system connection and synchronization logic are copied to the location `Server\configuration \com.sap.mobile.platform.server.agentry.application`

f) All application specific configuration files and all other file resources which **are referenced** by Java system connection synchronization logic are copied to one of two locations, depending on the approach taken after following the procedure provided in the topic "Preparing Agentry Applications With Java System Connections." If the recommended option to change the logic to use the API file method was implemented, copy the Java-referenced files to the location `Server\configuration \com.sap.mobile.platform.server.agentry.application`. Otherwise, these Java-referenced files must be copied to the root directory of the SAP® Mobile Platform Server 3.0 installation.

10. Start the Agentry Server within the SAP® Mobile Platform Server 3.0 using the Management Cockpit. When the startup is complete, verify the Agentry Server has successfully connected with the back end system.

11. At this point the system should be thoroughly tested, ideally with full end-to-end testing and operations. When this is complete, it is next necessary to upgrade the Agentry Client installations within the implementation environment in order to migrate all mobile users to the new environment. This is a standard client device operation consisting of the following general steps:.

a) Have all users perform a final transmit to verify all information stored on the device has been updated to the back end system.

b) All users should shut down their clients prior to upgrading them.

c) Install the Agentry Client provide with the SAP® Mobile Platform 3.0 for the client device type.

d) Each user should start the new Agentry Client, login, and perform an initial transmit.

With the complete of this procedure the mobile application has been successfully migrated to the SAP® Mobile Platform 3.0 implementation.

Next

At some point after the completion of fully migrating all mobile users to the new environment, it is possible to uninstall the previous version of the Agentry Client from the mobile devices, provided there are two separate versions of the client. For platforms, the Agentry Client may be upgraded in place, resulting in only one Agentry Client executable existing on the device at any given time.

Index