Migration

SAP Mobile Platform 3.0 SP02

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.
- 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
Messaging Channel	Not supported	Not supported
REST SDK	Supported (standard SDK installer)	Supported (standard SDK installer)

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.

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.

Table 3. List of Refactored Classes

Class name in 2.3.x version (old - deprecated)	Class name in 3.0 version (new - refactored)
SDMRequestManager	RequestManager
SDMConnectivityParameters	ConnectivityParameters
SDMPreferences	Preferences
SDMLogger	Logger

Class name in 2.3.x version (old - deprecated)	Class name in 3.0 version (new - refactored)
SDMBaseRequest	BaseRequest
SDMBundleRequest	BundleRequest
SDMResponseImpl	ResponseImpl
SDMHttpChannelListeners	HttpChannelListeners
SDMConnectivityException	ConnectivityException
SDMRequestStateElement	RequestStateElement
SDMPreferencesException	PreferencesException
ISDMNetListener	INetListener

Parser
The following table lists the refactored classes and protocols.

Table 4. 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.

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

You can migrate 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. Be sure to complete the prerequisites before attempting the migration process.

Preparing Agentry Applications With Java System Connections

When migrating an Agentry application to the SAP Mobile Platform Server 3.0, be sure to account for changes within the architecture that directly affect how the Java synchronization logic of the mobile application processes file references. Skip this information if your application does not include a Java system connection.

In releases of SAP Mobile Platform prior to 3.0 and Agentry Mobile Platform versions 6.0.x and earlier, the Java synchronization logic could reference file resources relative to the location of the Agentry Server publish directory (where Agentry.ini is located). In product-name 3.0, the current directory when processing the Java logic is no longer the publish directory.

To prepare Agentry applications for migration to SAP Mobile Platform 3.0, you modify the Java synchronization logic to change any references to configuration files or other resource files within the logic. In each case where such a file is referenced, replace it with a call to the method `com.syclo.agentry.Server.findConfigurationFile` within the Agentry Java API class `com.syclo.agentry.Server`. 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 to process a configuration file.

When making this change, 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 Platform SDK. Update the AJ-API to the version provided with SAP Mobile Platform SDK 3.0. This API is contained in the `Agentry-v5.jar` file provided with SAP Mobile Platform SDK.

Continue with the migration steps to copy all the necessary resource files to the proper location for SAP Mobile Platform.

Migrating Agentry Applications to SAP Mobile Platform

You can migrate SAP Mobile Platform 2.3 Agentry applications to SAP Mobile Platform 3.0.

Prerequisites

- Install SAP Mobile Platform SDK and SAP Mobile Platform Server.

- Modify the file references within the Java Synchronization logic for any Agentry applications that contain a Java system connection.
- Ensure you have access to the previous Agentry Editor version and Eclipse workspace in which the Agentry application project exists.
- Install the Agentry Editor plug-in as provided in the SAP Mobile Platform SDK to Eclipse.

Task

1. In an Agentry perspective in the SAP Mobile Platform 2.3 Agentry Editor, export the entire Agentry application project.
   The resulting file name contains the extension `.agx` or `.agxz`.
2. In an Agentry perspective in the SAP Mobile Platform 3.0 Agentry Editor, right-click in Project Explorer view and select **Import**.
3. Follow the wizard to complete the import process.

   During the import process, you can associate the project with the development server.

   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 Project Explorer view.
4. Make sure the Network Connect Type is **WebSockets over HTTPS** for all transmit configurations.
5. (Optional) Migrate the associated Java back-end project to the Eclipse workspace.
6. Publish the application as a Production version.
   a) Click the **Publish** button, then browse to the location where you want to publish (save) the ZIP file.
   b) (Optional) Right-click within Publish Folder Structure to add auxiliary project files, which may include Java resource files, application-specific DLL files, JAR files for the back end, and other back-end configuration files.
   c) Click **Next**, then **Finish**.
7. In Management Cockpit, create an Agentry application definition in Management Cockpit.
   a) Under Quick Links, select **Configure Application**.
   b) In Applications, click **New**, enter the information for the application definition.

      Select **Agentry** for the type, then click **Save**.
   c) In App Specific Settings, under Publish, browse to the Agentry application zip file, then click **Open**.
   d) In Back End, enter the back-end connection information for the back-end connection type.

      It may be helpful to have the previous `Agentry.ini` file as a reference when entering the back-end information.
   e) In Authentication, indicate the security profile or create a new one.
   f) Set any other applicable configuration options.

g) Click **Save**.

   Once you save the application definition, if you change any application-specific configuration options preceded with an "i," stop and restart the SAP Mobile Platform Server.

8. For Java System connections, add the `com.sap.mobile.platform.server.agentry.application` directory to the system path.

- On Windows, edit `Server/bin/setenv.bat`. At the end of the file, add `set PATH=%SMP_HOME%\configuration\com.sap.mobile.platform.server.agentry.application;%PATH%`
- On UNIX, edit `Server/bin/setenv.sh`. At the end of the file, add `LD_LIBRARY_PATH=$SMP_HOME/configuration/com.sap.mobile.platform.server.agentry.application:$LD_LIBRARY_PATH export LD_LIBRARY_PATH`

9. Perform full end-to-end testing.

When this is complete, upgrade the Agentry Client installations within the implementation environment to migrate all mobile users to the new environment.

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 for the client device type.

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

Next

At some point after the completion of fully migrating all mobile users to the new environment, you can 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