Developer Guide: Migrating to SAP Mobile SDK 2.3

SAP Mobile Platform 2.3 SP02

DOCUMENT ID: DC01912-01-0232-01
LAST REVISED: May 2013

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

Migrate Your Artifacts

(Audience: application developers) Migrate your applications to SAP® Mobile Platform 2.3 to take advantage of new features.

The upgrade to SAP Mobile Platform 2.3 is performed in place, which means you can continue to run 2.2 applications without migrating them. You might need to perform some migration tasks to take advantage of new features and system improvements. See *Best Practices for Migrating Applications* on page 1 for additional information.

After you install and upgrade your SAP Mobile Server instances, migrate your mobile business objects (MBOs), projects, and applications as needed. These instructions are for migrating client applications from SAP Mobile Platform 2.2 SP02 to 2.3.

Note: References to 2.2 and 2.3 include support packages; specific support packages are identified only if there is a change significant to a particular support package. SAP recommends you always install the latest support package available.

If you upgraded from a version earlier than 2.2 SP02, refer to *Developer Guide: Migrating to Sybase Mobile SDK 2.2 SP02* (cumulative for 2.2), and its updates, for application migration information: *http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01857.0222/doc/html/mqu1352843142074.html*

For supporting information, see:

- *New Features*
- *Supported Hardware and Software*

Best Practices for Migrating Applications

Use information to formulate best practices for migrating applications.

When you upgrade to the latest version of SAP Mobile Platform, client applications continue to run without migrating them. In some cases, adjustments are required to ensure the application runs correctly; and in cases where the client application is based on mobile business objects, the project needs to be started in the Mobile Application Diagram to automatically trigger project migration steps. But overall, the client application continues to run and can synchronize with its enterprise information system. Any exceptions are noted in the documentation.

A client application is compiled code that is based on its data model, and consists of a binary piece, and an SAP Mobile Server piece. This enables the application to execute on devices and in the server. Over time, features are added and improvements made to the SDK and SAP Mobile Server. To take advantage of these improvements, you need to upgrade your server, or implement a more recent SDK version.

If you rely only on in-place migration, after multiple server upgrades your client application may cease to work efficiently or at all. A best practice is to recompile your client application code after a major release, so that the binary and SAP Mobile Server versions are the latest. One strategy is to wait several weeks to ensure the upgraded environment is stable, and then recompile.

Migrate Agentry Applications

Procedures are required to migrate current Agentry applications to SAP Mobile Platform 2.3.

Migrating From Agentry Mobile Platform to SAP Mobile Platform

Prerequisites

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

- The SAP Mobile Platform 2.3 is assumed to be installed and properly configured.
- If implementing an SAP Mobile Platform clustered environment, this should be established, and the Agentry application defined in each node, prior to beginning the migration process. The migration should then be performed with deployment to the primary node in the cluster.
- For mobile applications which make use of a Java system connection, the Java Runtime Environment (JRE) should be installed to the host system for the SAP Mobile Platform prior to performing this procedure. Note that installation of the JRE requires the update of the system's PATH environment variable with the location of the `bin` and `lib` directories of the JRE installation.
- If the mobile application is one provided by SAP built on Agentry 6.0.x, review and have available the *Implementation and Administration* guide for the application being migrated. This manual can be found on the SAP Marketplace page for the mobile application. Items related to server configuration and environment setup, as well as system requirements, are applicable to the migration and implementation of the mobile application in SAP Mobile Platform 2.3.
- The person performing this procedure must have detailed, development-level knowledge concerning the application to be upgraded from Agentry Mobile Platform 6.0.x. This includes the following items, though this list may not be comprehensive and the requirements will vary from one application to the next:
  - Java resources, such as application specific `.jar` files
  - Application-specific configuration files

    - Application-specific resource files such as dynamic link libraries (DLL's)
    - Administration scripts typically stored within the `sql` directory of the Agentry Server
- The SAP Mobile SDK 2.3 should already have been retrieved and its contents extracted.
- The person performing this procedure must have access to and the proper permissions for the SAP Control Center to allow for the import of ZIP archives into Agentry Applications and to start and stop services within the SAP Mobile Platform.

Task

The purpose of this procedure is to upgrade or migrate a mobile application built and deployed on Agentry Mobile Platform 6.0.x, a.k.a. "Agentry Standalone" to the SAP Mobile Platform 2.3. This process can be performed to upgrade a current production implementation, or to upgrade a new implementation using an out-of-the-box mobile application provided by SAP and built on Agentry Mobile platform 6.0.x. Those familiar with the process for upgrading mobile applications from one version of Agentry to another will find this procedure to be similar, though with some key differences in the execution.

From a high level, this procedure accomplishes the following main tasks in order to migrate the mobile application:

- All application-specific resources stored on the Agentry 6.0.x Server are bundled together in a ZIP archive, with the exception of the business logic itself
- The business logic is imported from the Agentry 6.0.x Server as a new project in the Eclipse workspace for the Agentry Editor for SAP Mobile Platform 2.3. This upgrades the business logic to the latest format.
- The application is published to the Agentry Server running with the SAP Mobile Platform 2.3. This updates the configuration sections for the application related to the defined system connections.
- The ZIP archive containing the non-Agentry application-specific resources is imported using the SAP Control Center into the Agentry Server for the application within the SAP Mobile Platform 2.3.

The end result of this process is a merging of the new Agentry Server resources within the SAP Mobile Platform with the mobile application-specific resources as implemented in the Agentry 6.0.x Server. The following instructions provide the steps necessary to accomplish this migration.

This procedure is applicable to any application built and deployed on Agentry Mobile Platform 6.0.x, whether it be a product from SAP, or a custom application built by the customer.

1. If the mobile application to be migrated is not yet installed in the implementation environment, the server component for the mobile application should be installed to a separate, but accessible location in order to provide the mobile application business logic and application-specific resources. This must be a production server installation of the mobile application.

2. Install the Agentry Server **for production** as provided in the SAP Mobile SDK 2.3 according to the instructions provided in the *Install SAP Mobile SDK 2.3* guide.
3. Install and configure the Agentry Editor for SAP Mobile Platform 2.3 as provided in the SAP Mobile SDK 2.3 according to the instructions provided in the *Install SAP Mobile SDK 2.3* guide.
4. Within the Agentry Editor for SAP Mobile Platform 2.3, import the application from the Agentry 6.0.x Server, creating a new Agentry application project within the Eclipse workspace.
5. Publish the application project from the Agentry Editor to the Agentry Server installed from the SAP Mobile SDK 2.3.
6. Create a ZIP archive, preserving the directory structure, containing the following items found in the Agentry 6.0.x Server installation:
   - Application-specific configuration files, which **does not include** configuration files provided with standard Agentry Server installations
   - 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.
   - 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.
7. Add to the ZIP archive the folder `Application` found in the Agentry Server installation from the SAP Mobile SDK 2.3, as well as the `Agentry.ini` configuration file.
8. Using the SAP Control Center, define a new Agentry application according to the procedure found in the *SAP Control Center for SAP Mobile Platform* guide, in the section *Creating Agentry Application Definition*.
9. Within the SAP Control Center, stop the Agentry Server instance just created if it is currently running.
10. Import the ZIP archive containing the application-specific resources and the `Application` folder according to the procedure in the *SAP Control Center for SAP Mobile Platform*, in the section *Deploying Agentry Application Files to an Exiting Application*.
11. Start the Agentry Server instance for the application using the SAP Control Center.
12. Configure the Agentry Server within the SAP Mobile Platform using the SAP Control Center, including system connections, client-server communications, and other standard configuration tasks. For applications provided by SAP see the *Implementation and Administration* guide for the product for information on configuring the Server for the application. Note that instructions may reference the Agentry Administration Client as

provided with Agentry 6.0.x and prior releases. The SAP control Center is now used to perform the configuration, but the appropriate settings and options are the same.

With the completion of this procedure, the mobile application originally built and deployed on the Agentry Mobile Platform version 6.0.x has been upgraded and migrated to the SAP Mobile Platform 2.3. Application-specific resources have been moved to the Agentry Server instance within the SAP Mobile Platform and the business logic for the application has been upgrade and imported.

Next

The next steps should be to thoroughly test the updated application with standard testing procedures, including end-to-end synchronization tests involving the Agentry Clients.

Migrate Mobile Business Objects

You must complete the steps below to migrate 2.2 SP02 mobile business objects (MBOs) to SAP Mobile Platform version 2.3.

If you are migrating from a version earlier than 2.2 SP02, see *Developer Guide: Migrating to SAP Mobile SDK 2.2 SP02*, and its updates, on Product Documentation, the *Migrate Mobile Business Object* section: *http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01857.0222/doc/html/mqu1352843142355.html*

Migrate 2.2 SP0x Mobile Business Objects to 2.3

You must migrate 2.2 SP0*x* Mobile Business Objects (MBOs) to SAP Mobile Platform version 2.3.

1. From Eclipse, point to the existing MBO project's workspace.
2. Ensure connection profiles referenced by the MBO projects are in place or imported, and enterprise information system (EIS) data sources associated with those connection profiles can be connected.
3. Once SAP Mobile WorkSpace is started, open the Mobile Application Diagram. This automatically triggers the Mobile Application project migration.

Additional Steps to Migrate MBOs

- In versions earlier than SAP Mobile Platform version 2.2 SP02, SAP Mobile WorkSpace allowed mapping of operations with multiple MBO arguments (Filled from Attribute, client parameter, and personalization key) at the same time, even though it might not work properly on the device application during runtime.

  With version 2.2 SP02, when adding a mapping of an operation argument, SAP Mobile WorkSpace now allows only one of the three sources (MBO attribute, client parameter, personalization key) to map into the operation argument at one time; that is, the argument value sources are mutually exclusive.

However, when migrating the Mobile Application project from earlier versions, SAP Mobile WorkSpace preserves the original MBO operation argument value assignment choices the developer made, to retain backward compatibility with the project in the earlier version. SAP Mobile WorkSpace does not remove any mappings when migrating a project.

In a migrated project, if an operation argument is mapped to a client parameter as well as an attribute or personalization key, this warning appears:

```
Client parameter parameterName might not be used,  as the mapped
argument has 'Fill from Attribute' or 'Personalization Key'
specified.
```

The developer must adjust the MBO model so that an operation argument maps to only one source.

Note: The developer can provide a default value for the operation argument, regardless of how the argument is mapped.

Note: In releases prior to 2.2 SP02, SAP Mobile WorkSpace automatically created client parameters definition and mapped them to the related operation arguments. After migration, those client parameters and mapping would stay. But when the user creates a new MBO operation, the client parameter and its mapping to operation argument will not be automatically created. In case the users want to have the client parameters and the mappings to the operation arguments, they can drag and drop an operation argument to the Client Parameters folder in the Input mapping page from the MBO operation wizard or Properties view's Input tab.

Migrate Object API Applications

No steps are required to migrate 2.2 SP02 applications to SAP Mobile Platform version 2.3.

If you are migrating from a version earlier than 2.2 SP02, see *Developer Guide: Migrating to Sybase Mobile SDK 2.2 SP02*, and its updates, on Product Documentation, the *Migrate Object API Applications* section: *http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01857.0222/doc/html/mqu1352843141277.html*

Native Client Version Compatibility Matrix

Compatibility between versions of the client object API and SAP Mobile Server (Unwired Server).

Native Client Object API and SAP Mobile Server Version Compatibility

	Unwired Server 2.1	Unwired Server 2.1 ESD #1	Unwired Server 2.1 ESD #2	Unwired Server 2.1 ESD #3	Unwired Server 2.2 SP02	SAP Mobile Server 2.3
Native Client Object API 2.1	Yes	Yes	Yes	Yes	Yes	Yes
Native Client Object API 2.1 ESD #1	No	Yes	Yes	Yes	Yes	Yes
Native Client Object API 2.1 ESD #2	No	No	Yes	Yes	Yes	Yes
Native Client Object API 2.1 ESD #3	No	No	No	Yes	Yes	Yes
Native Client Object API 2.2 SP02	No	No	No	No	Yes	Yes
Native Client Object API 2.3	No	No	No	No	No	Yes

Note:

- Yes – the client application built in this SDK version is supported in the server version (minor adjustments may be necessary to ensure the application runs correctly; see the migration details for the appropriate application type, if any).

- No – the client application built in this SDK version is not supported in the server version.
- Server version – refers to the server version to which an original package is migrated, and not a newly deployed package. For the example of "Native Client Object API 2.1" vs. "SAP Mobile Server 2.3", the application package that runs on "SAP Mobile Server 2.3" may not always be newly created and deployed from MobileSDK2.3; it may have been originally created from MobileSDK2.1 and deployed to 2.1 server, and then migrated to 2.3 server.

Migration Paths for Android

Paths available to migrate Android object API applications from earlier versions to the current version.

Application is Built with SDK Version	Migration Tasks
2.1.1 2.1.2 2.1.3	Migrate your application to the current version. See the migration instructions: • *Migrating Android Applications to 2.2* on page 8
2.2 2.2 SP01 2.2 SP02 2.3	No migration changes are required.

Migrating Android Applications to 2.2

These changes are required to migrate Android applications to 2.2.

Afaria library changes require you to modify and recompile your applications.

1. Access the Android Afaria client library and JAR files that are available in: *SMP_HOME*\MobileSDK<*X.X*>\ObjectAPI\Android

   Note: Alternatively, navigate to the Mobile Enterprise Technical Support website at *http://frontline.sybase.com/support/downloads.aspx* (registration required).

   Download the appropriate Android Afaria client (see *Supported Hardware and Software*).

2. Import the Android Afaria client using information in *Developer Guide: Android Object API Applications*. See *Importing Libraries and Code* (in either the *Development Task Flow for Object API Applications* section, or the *Development Task Flow for DOE-based Object API Applications* section as appropriate).

Migration Paths for BlackBerry

Paths available to migrate BlackBerry object API applications from earlier versions to the current version.

Application is Built with SDK Version	Migration Tasks
2.1 2.1.1	Migrate your application to the current version. See the migration instructions: • *Migrating BlackBerry Applications to 2.1 ESD #2* on page 12 • *Migrating BlackBerry Applications to 2.2* on page 9
2.1.2	Migrate your application to the current version. See the migration instructions: • *Migrating BlackBerry Applications to 2.2* on page 9
2.1.3 2.2 2.2 SP01 2.2 SP02	Migrate your application to the current version. See the migration instructions: • *Migrating BlackBerry Applications to 2.2* on page 9
2.3	No migration changes are required.

Migrating BlackBerry Applications to 2.2

No migration changes are required for BlackBerry Object API applications; however, you may need to perform some migration steps to take advantage of new features in 2.2.

- **Client library changes –** for BlackBerry:
  - The `sup_client2.jar` client is now shipped as a library, with no separate `sup_client2.cod` and `sup_client2.alx` files. This requires a change to how you develop BlackBerry projects:
    - **Eclipse projects –** export `sup_client2.jar` into the build path configuration.
    - **BlackBerry JDE projects –** create a library project including `sup_client2.jar`.
  - Several client files have been deleted in version 2.2 SP02: `CommonClientLib`, `MessagingClientLib`, `MocaClientLib files`, and `MCL.jar` substitutes. However, `MCL.jar` packages and classes are shipped into `sup_client2.jar`, so

change your application to reference `sup_client2.jar` and `UltraliteJ12.jar`

For information and examples for migrating existing BlackBerry applications to 2.2 SP02 implementing these changes, see *Migrating BlackBerry Applications (Eclipse Project)* on page 10 and *Migrating BlackBerry Applications (JDE Project)* on page 11.

- **API changes –** a new `setApplicationIdentifier(String value, String signerId)` API is available to replace the old signing implementation. It is based on BlackBerry Password Based Code Signing Authority.

  To learn more about the BlackBerry Password Based Code Signing Authority on which the API is based, and about the parameter `signerId`: *http://supportforums.blackberry.com/t5/Java-Development/Protect-persistent-objects-from-access-by-unauthorized/ta-p/524282*.

  To download the BlackBerry signing tool used with this new API: *https://swdownloads.blackberry.com/Downloads/entry.do?code=D82118376DF344B0010F53909B961DB3*.

  For information and examples for migrating existing BlackBerry applications to 2.2 SP02 implementing this change, see *Migrating BlackBerry Applications (Eclipse Project)* on page 10 and *Migrating BlackBerry Applications (JDE Project)* on page 11.

  Note: With this change, the `setApplicationIdentifier(String value)` API is deprecated and will be removed in a future release.

Migrating BlackBerry Applications (Eclipse Project) to 2.2

Migrate BlackBerry Object API applications from 2.1 ESD #3 to version 2.2 using an Eclipse project.

These steps use an example that demonstrates the new BlackBerry signing API method.

To learn more about the BlackBerry Password Based Code Signing Authority on which the API is based, and about the parameter `Signer Id`: *http://supportforums.blackberry.com/t5/Java-Development/Protect-persistent-objects-from-access-by-unauthorized/ta-p/524282*

1. Download the BlackBerry signer tool, and install it in your development environment: *https://swdownloads.blackberry.com/Downloads/entry.do?code=D82118376DF344B0010F53909B961DB3*.
2. After installing the signer tool, generate a new key file (for example: `suptest.key`).
3. Create the BlackBerry project in Eclipse:
   a) Navigate to **Configure Build Path** > **Libraries** tab, and reference:
      - `sup_client2.jar`
      - `UltraliteJ12.jar`
   b) Navigate to the **Order and Export** tab, and check to make sure the `sup_client2.jar` file is included in your application JAR file.

4. Copy the generated key file (for example, `suptest.key`) to the project `src` folder.
5. In your application source code, set the new key file (`suptest` in this example):

   ```
   com.sybase.mobile.Application.getInstance().setApplication
   Identifier(end2end.test.Const.ApplicationIdentifier,
   "suptest");
   ```
6. Build your project, and run the application on a simulator to test it.
7. When you are ready to run the application on a real device, sign the .cod files using the signature tool (**BlackBerry > Sign**). After you sign the .cod files with the BlackBerry signature tool, use the File Signer that you installed in step 1 to sign the .cod file again.
8. Install the cod files on the device using provisioning procedures, and run the application.

Migrating BlackBerry Applications (JDE Project) to 2.2

Migrate BlackBerry Object API applications from 2.1 ESD #3 to version 2.2 using a BlackBerry JDE project.

These steps use an example that demonstrates the new BlackBerry signing API method.

To learn more about the BlackBerry Password Based Code Signing Authority on which the API is based, and about the parameter `Signer Id`: *http://supportforums.blackberry.com/t5/Java-Development/Protect-persistent-objects-from-access-by-unauthorized/ta-p/524282*

1. Download the BlackBerry signer tool, and install it in your development environment: *https://swdownloads.blackberry.com/Downloads/entry.do?code=D82118376DF344B0010F53909B961DB3*.
2. After installing the signer tool, generate a new key file (for example: `suptest.key`).
3. Create a BlackBerry library project in the IDE, add `sup_client2.jar` to the project, and then build it.
4. Create an empty BlackBerry project in the IDE:
   a) Navigate to **Configure Build Path**, and import JAR files:
      - `UltraliteJ12.jar`
      - `ULjDatabaseTransfer.jar`
   b) Navigate to the **Project Dependencies** tab, and check the library project.
5. Copy the generated key file (for example, `suptest.key` to the project root folder.
6. In your application source code, set the new key file (`suptest` in this example):

   ```
   com.sybase.mobile.Application.getInstance().setApplication
   Identifier(end2end.test.Const.ApplicationIdentifier,
   "suptest");
   ```
7. Build your project, and run the application on a simulator to test it.
8. When you are ready to run the application on a real device, sign the .cod files using the signature tool. After you sign the .cod files with the BlackBerry signature tool, use the File Signer that you installed in step 1 to sign the .cod file again.

9. Install the cod files on the device using provisioning procedures, and run the application.

Migrating BlackBerry Applications to 2.1 ESD #2

These changes are required to migrate BlackBerry applications to 2.1 ESD #2.

Update your application:

1. The Application APIs (in the `Application` class) are required for managing application registrations, connections, and context. Rewrite the initialization code in your application to use the Application APIs.
   For information on the `Application` interface, search for *Application APIs* in the Developer Guide for your platform.
2. Callbacks related to application events are contained in a separate `ApplicationCallback` interface. Rewrite your application code to use this interface.
   For information on the `ApplicationCallback` interface, search for *Callback and Listener APIs* in the Developer Guide for your platform.
3. Replication-based synchronization clients require two data channels: a data channel for data synchronization, and a messaging channel for sending registration and push notifications to the client. Update your port configuration for both channels. See *Sybase Control Center for Sybase Unwired Platform* > *Administer* > *Unwired Server* > *Server Properties*.
4. To continue using server-initiated synchronization, you must write code for handling notifications. If change notifications are enabled for synchronization groups, you can implement the `onSynchronize` callback method to monitor this condition, and either allow or disallow default background synchronization.

```
public int onSynchronize(ObjectList groups,
SynchronizationContext context)
{
  int status = context.getStatus();
  if (status == SynchronizationStatus.STARTING_ON_NOTIFICATION)
  {
    // There is changes on the synchronization group
    if (busy)
    {
      return SynchronizationAction.CANCEL;
    }
    else
    {
      return SynchronizationAction.CONTINUE;
    }
  }

  // return CONTINUE for all other status
  return SynchronizationAction.CONTINUE;
}
```

5. Rebuild your application as described in *Migrating BlackBerry Applications (Eclipse Project)* or *Migrating BlackBerry Applications (JDE Project)*.

Migration Paths for iOS

Paths available to migrate iOS object API applications from earlier versions to the current version.

Application is Built with SDK Version	Migration Tasks
2.1 2.1.1	Migrate your application to the current version. See the migration instructions: • *Transitioning Applications to Release 2.1 ESD #2* on page 16 • *Transitioning MBS Applications to the Current Release (2.1 ESD #3 or Later)* on page 19
2.1.2	Migrate your application to the current version. See the migration instructions: • *Transitioning MBS Applications to the Current Release (2.1 ESD #3 or Later)* on page 19
2.1.3 2.2 2.2 SP01 2.2 SP02 2.3	No migration changes are required.

Migrating iOS Native Custom Applications

Understand the strategies and steps to follow when you transition applications to the current release.

Migration Strategies

Your strategy for transitioning MBS-based iOS applications to the current release depends on your current installation configuration, upgrade plans, and the data model changes in the application to be transitioned. Follow the guidance in the scenario that fits your installation configuration and upgrade plan.**Scenario 1**

- Current Installation - 2.1 ESD #2 or earlier MBS client application on 2.1 ESD #2 or earlier Unwired Server
- Upgrade Plan - Upgrade only Unwired Server to the current version, and maintain the existing MBS client application

Your MBS client application should continue to work without error after server upgrade, though some RBS features will not be available for your MBS client application. See *Maintaining MBS Client Applications* on page 18

Scenario 2

- Current Installation - 2.1 ESD #2 or earlier MBS client application on 2.1 ESD #2 or earlier Unwired Server
- Upgrade Plan - Upgrade both Unwired Server and client application to the current version. Upgrade the client application to an RBS-based application.
- No Data Model Changes in the application

Recommended Steps:

1. Instruct application users to submit all pending data to the Unwired Server using the existing MBS client application before you migrate to the new RBS application, and coordinate the upgrade. This is an important step as it will ensure that application users do not lose any modified data during your migration. With MBS, once **submitPending** is invoked, the modified data is wrapped as an operation replay message to be sent as soon as connectivity with the server is available. If the application user does not invoke **submitPending** prior to migration, all of their data changes will be lost once migration begins. For this reason, you will need to instruct the application users to use the appropriate UI control exposed by the MBS application to invoke **submitPending** before you migrate the application.
2. Follow the steps included in *Transitioning MBS Client Applications* on page 19 to convert the MBS application to the new RBS application, creating a different application name for the new RBS application on the device. Include explicit screens/message popups within the application to alert the application user to follow these steps:
   a. Submit all pending data from the MBS client application to the Unwired Server.
   b. Confirm that the pending data has been submitted, delete the MBS application, and then begin using the new RBS application.

      Note: Once the application user acknowledges and confirms that pending data from the old application has been submitted, do not display the popup/screen messages again.
   c. Subscribe and synchronize the new RBS application with the upgraded Unwired Server.

Note: You need to use a different Application Name to avoid an accidental update of the MBS application before the application user has a chance to submit their changes. However, you *can* use the same Application ID for both the new RBS application and for the existing MBS application.

For more in depth steps to transition your MBS client application to RBS, see *Transitioning MBS Client Applications* on page 19

Scenario 3

- Current Installation - 2.1 ESD #2 or earlier MBS client application on 2.1 ESD #2 or earlier Unwired Server
- Upgrade Plan - Upgrade both Unwired Server and client application to the current version. Upgrade the client application to an RBS-based application.
- Data Model Changes in the application or MBO project

Recommended Steps:

1. Instruct application users to submit all pending data to the Unwired Server using the existing MBS application before you migrate to the new RBS-based application, and coordinate the upgrade. This is an important step as it will ensure that application users do not lose any modified data during your migration. With MBS, once **submitPending** is invoked, the modified data is wrapped as an operation replay message to be sent as soon as connectivity with the server is available. If the application user does not invoke **submitPending** prior to migration, all of their data changes will be lost once migration begins. For this reason, you will need to instruct the application users to use the appropriate UI control exposed by the MBS application to invoke **submitPending** before you migrate the application.
2. Deploy the new package with data model changes to the server using a new Application ID. Create a new application connection in the Sybase® Control Center.
3. Follow the steps included in *Transitioning MBS Client Applications to the Current Release* on page 19 to convert the MBS application to the new RBS application, creating a different application name and application id for the new RBS application on the device. Include explicit screens/message popups within the application to alert the user to follow these steps:
   a. Submit all pending data from the MBS client to the Unwired Server.
   b. Confirm that the pending data has been submitted, delete the MBS application, and then begin using the new RBS application.

      Note: Once the application user acknowledges and confirms that pending data from the old application has been submitted, do not display the popup/screen messages again.
   c. Subscribe and synchronize the new RBS application with the upgraded Unwired Server.

For more in depth steps to transition your MBS client application to RBS, see *Transitioning MBS Applications to the Current Release (2.1.3 ESD #3 or Later)* on page 19

Note: For Scenario 2 and 3, there is no data transitioning solution when migrating MBS applications to RBS applications. After the application is converted to RBS, the application user must synchronize the application with the Unwired Server. The new application will not use the data residing in the device database for the old application so the application user will need to delete the old application from the device. If the old application is not removed from the device, the database for the old application will continue to reside on the device; this may double the space consumed on the device when the new application downloads records to the new database.

Transitioning Applications to Release 2.1 ESD #2

Transition applications to release 2.1 ESD #2 by making changes to application registration.

Making Changes to Application Registration

This task is not required if your application is built with SDK version 2.1 ESD #2. For applications built with SDKs prior to 2.1 ESD #2, make changes to the application to allow it to register.

1. The Application APIs (`SUPApplication` class) are required for managing application registrations, connections, and context. Rewrite the initialization code in your application to use the Application APIs. For information on the `Application` interface, search for *Application APIs* in the Developer Guide for your platform.
   For iOS applications, the Messaging Client API has been removed. Replace references in your application to the Messaging Client API (`SUPMessage` class) with the appropriate use of the Application APIs (`SUPApplication`).
2. Callbacks related to application events are now contained in a separate `ApplicationCallback` interface. Rewrite your application code to use this interface. For information on the `ApplicationCallback` interface, search for *Callback and Listener APIs* in the Developer Guide for your platform.
3. Complete application registration through an automatic or manual process. See the *Application and User Management Overview* topic group in *SAP Control Center for SAP Mobile Platform*.
   Use the `SUPApplicationCallback` APIs to check that the application successfully registered and the messaging client connection is established.

The following is sample code from the SUP101 project for `ApplicationCallbackHandler`.

```
#import "SUPApplicationDefaultCallback.h"

// These strings will be used to send out NSNotifications.
#define ON_CONNECTING  @"SUPConnecting"
#define ON_CONNECT_FAILURE @"SUPConnectFailure"
#define ON_CONNECT_DISCONNECT @"SUPConnectDisconnect"
#define ON_CONNECT_SUCCESS @"SUPConnectSuccess"
#define ON_REGISTER_SUCCESS @"SUPRegisterSuccess"
#define ON_REGISTER_FAILURE @"SUPRegisterFailure"

@interface ApplicationCallbackHandler :
SUPApplicationDefaultCallback
{

}

+ (ApplicationCallbackHandler*)getInstance;
@end
#import "ApplicationCallbackHandler.h"
```

```
@implementation ApplicationCallbackHandler

+ (ApplicationCallbackHandler*)getInstance
{
   ApplicationCallbackHandler* _me_1 = [[ApplicationCallbackHandler
alloc] init];
    [_me_1 autorelease];
    return _me_1;
}


- (void)notify:(NSNotification *)notification
{
     [[NSNotificationCenter defaultCenter]
postNotification:notification];
}

- (void)onConnectionStatusChanged:
(SUPConnectionStatusType)connectionStatus :(int32_t)errorCode :
(NSString*)errorMessage
{
    NSLog(@"================================================");
    NSLog(@"onConnectionStatusChanged: status = %d, code = %d,
message = %@",connectionStatus,errorCode,errorMessage);
    NSLog(@"================================================");
    NSString *notification = nil;
    switch(connectionStatus)
    {
        case SUPConnectionStatus_CONNECTING:
            notification = ON_CONNECTING;
            break;
        case SUPConnectionStatus_CONNECTION_ERROR:
            notification = ON_CONNECT_FAILURE;
            break;
        case SUPConnectionStatus_CONNECTED:
            notification = ON_CONNECT_SUCCESS;
            break;
        default:
            // Ignore all other status changes for this example.
            break;
    }

    if (notification != nil)
    {
         NSNotification *n = [NSNotification
notificationWithName:notification object:nil];
         [self performSelectorOnMainThread:@selector(notify:)
withObject:n waitUntilDone:NO];
    }

}


- (void)onRegistrationStatusChanged:
```

```
(SUPRegistrationStatusType)registrationStatus :(int32_t)errorCode :
(NSString*)errorMessage;
{
    NSLog(@"================================================");
    NSLog(@"onRegistrationStatusChanged: status = %d, code = %d,
message = %@",registrationStatus,errorCode,errorMessage);
    NSLog(@"================================================");

    if (registrationStatus ==
SUPRegistrationStatus_REGISTRATION_ERROR)
    {

        NSNotification *n = [NSNotification
notificationWithName:ON_REGISTER_FAILURE object:nil];
        [self performSelectorOnMainThread:@selector(notify:)
withObject:n waitUntilDone:NO];
    }

    if (registrationStatus == SUPRegistrationStatus_REGISTERED)
    {

        NSNotification *n = [NSNotification
notificationWithName:ON_REGISTER_SUCCESS object:nil];
        [self performSelectorOnMainThread:@selector(notify:)
withObject:n waitUntilDone:NO];
    }

}

@end
```

Maintaining MBS Client Applications

To continue to use your existing MBS client applications, continue to use an earlier version of the SDK.

When you upgrade your Sybase Mobile SDK, the installation does not overwrite earlier versions of the SDK. Instead, the installation coexists with the earlier version of the SDK, and retains full backward compatibility with applications developed in the earlier version. However, features available in 2.1 ESD #3 or later versions of the SDK may not be available for applications developed in earlier versions of the SDK.

The following replication-based synchronization features are unavailable for messaging-based synchronization applications:

- Asynchronous upload of operation replay results
- Push synchronization APIs for sending change notifications to devices
- Change log APIs to allow a client to retrieve entity changes from the back end

For information on support of earlier SDKs with a 2.1 ESD #3 or later server, see the *Installation Guide for Sybase Mobile SDK > Getting Started > Backward Compatibility*.

For information on messaging-based synchronization applications, see the *Developer Guide: iOS Object API Applications* from 2.1 ESD #2.

Transitioning MBS Applications to the Current Release (2.1 ESD #3 or Later)

(Not applicable to DOE based applications) iOS applications built with earlier versions of the SDK use messaging-based synchronization (MBS) for data delivery. Applications built using SDK version 2.1 ESD #3 or later use replication-based synchronization (RBS) for data delivery to reduce synchronization time.

This task flow shows you how to transition your messaging-based application to the current release as a replication-based application. The tasks include setting up the project, updating the application, and testing the application.

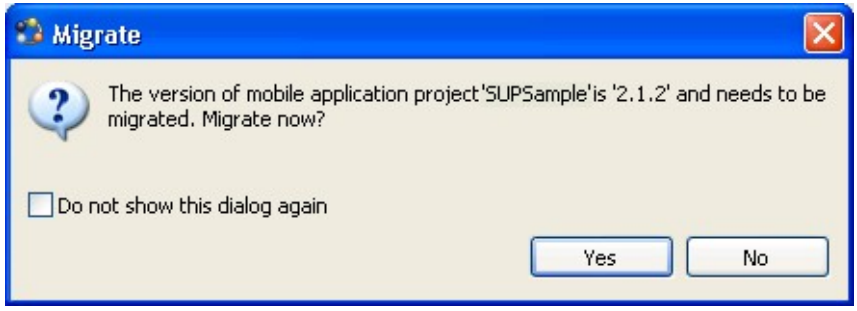
Note: The code samples in this task flow are from the SUP101 project from the *Tutorial: iOS Object API Application Development*.

Migrating the Project and Generating Code

Migrate the existing project to the current version of sdk-name, and generate new RBS object API code.

Important: Upgrade to the current version of sdk-name prior to migrating your project.

1. Export the existing mobile application project from the earlier version of tooling-name.
2. In the current version of tooling-name, import your existing application project.
3. Right-click the project and select **Open in Diagram Editor**.
4. Select **Yes** to migrate the project to the current version of the SDK.



5. Right-click the project and select **Generate Code** to generate code that supports replication-based synchronization.

For more information on code generation options, see *Developer Guide: iOS Object API Applications > Developer Task Flow for Object API Applications > Generating Objective-C Object API Code*.

Setting Up the Xcode Project

Set up the Xcode project with the generated code and libraries required in the current version of the SDK.

Important: Install the Xcode version required for the current version of sdk-name prior to setting up the Xcode project. See *Supported Hardware and Software*

1. In the Xcode project, open your existing application.
2. Remove the existing generated code and add the new generated code.
   To remove the existing generated code:
   a. In the Xcode tree view, right-click the `Generated Code` folder and select **Delete**.
   b. In the confirmation dialog, select **Delete**.
   c. In Finder, go to the Xcode project folder. Delete the empty `Generate Code` physical folder to ensure that the new generated code gets imported correctly by Xcode.
3. Remove all of the libraries that you added from the *SMP_HOME*`\MobileSDK\ObjectAPI\iOS\Libraries\` folder when you created the application in an earlier version of the SDK.
4. Add all of the libraries from the *SMP_HOME*`\MobileSDK<`*version*`>\ObjectAPI\iOS\RBS\Libraries\` folder in the current version of the SDK.
5. Remove the existing `\includes` header files and add the new ones from *SMP_HOME*`\MobileSDK<`*version*`>\ObjectAPI\iOS\RBS\includes\`
   To remove the existing files:
   a. In the Xcode tree view, right-click the `includes` folder and select **Delete**.
   b. In the confirmation dialog, select **Delete**.
   c. In Finder, go to the Xcode project folder. Delete the empty `includes` physical folder to ensure that the new generated code gets imported correctly by Xcode.

Making Changes to Application Initialization

Make changes to the application to allow it to initialize as required in 2.1 ESD #3.

1. Set the login credentials for login and database synchronization.

```
SUPConnectionProfile *sp = [SUPSampleSUPSampleDB
getSynchronizationProfile];
[sp setUser:@"supAdmin"];
[sp setPassword:@"supPwd"];
```

2. After you complete the registration, the server exchanges settings from the application connection template with the device. In most circumstances, you do not need to set additional properties for the application in the synchronization profile. If you need to override some of the properties from the template you can do so through the synchronization profile.

```
[sp setServerName:@"relayservername.com"];
[sp setNetworkProtocol:@"networkProtocol"];
[sp setPortNumber:portNumber];
```

3. Login and subscribe to the server using the credentials set up in step 1. In an MBS application, `subscribe` causes data to be pushed to the client from the server. For RBS, it allows the server to clean up client-specific information proactively (for example, synchronization parameters when they are no longer required). The server is typically configured to remove inactive artifacts after a certain period of time. With an RBS `subscribe`, no data is pushed to the device, it is only used for administrative purposes.

```
[SUP101SUP101DB onlineLogin];
[SUP101SUP101DB subscribe];
```

4. Determine the mode of synchronization to exchange data with the server. In RBS, you can perform synchronization synchronously or asynchronously. Synchronous means that the calling thread is blocked until the synchronization is complete whereas asynchronous synchronization leverages a background thread. Synchronization consists of upload (sending up the operation replay) and download (pulling down the new/changed data) phases. If you invoke the asynchronous API to perform synchronization, the appropriate callbacks are invoked to inform you of its completion.

   ```
   [SUP101SUP101DB synchronize];  // synchronous API
   [SUPSampleSUPSampleDB beginSynchronize];  // asynchronous API
   ```

5. Determine the mode of operation replay. As with synchronization, operation replay can be processed by the server in a synchronous or asynchronous manner.Synchronous means that the synchronization session that uploads the operation replay waits for its completion before initiating the download phase to pull down data, including the result and status of the operation replay. Asynchronous replay means the synchronization session immediately goes to the download phase after the operation replay is successfully queued.

   For an MBS application migrating to RBS, asynchronous replay is closer in behavior. You can implement this behavior in the synchronization API that has an `uploadOnly` parameter. By setting this parameter to true, the synchronization session skips the download phase, and only the operation replay is sent to the server. However, that is not to say that you should always use asynchronous replay. You should make the decision based on the business use case instead of the behavior of the previous implementation.

   You only need to set the asynchronous replay flag once.

   ```
   [sp setAsyncReplay:NO];  // Synchronous replay
   [SUP101SUP101DB synchronize];  // Synchronous synchronization
   ```

   When the `synchronize` method returns, the operation replay has completed and the data/result is on the client side database. You can also use the asynchronous synchronization API:

   ```
   [sp setAsyncReplay:NO];  // Synchronous replay
   [SUP101SUP101DB beginSynchronize];  // Asynchronous
   synchronizaiton
   ```

   The `onSynchronize` callback with a `SUPSynchronizationStatus_FINISHING` status is fired when the synchronization has completed. At this point, the operation replay has completed and the data/result is on the client side database. To leverage asynchronous replay, use the API that supports the `uploadOnly` parameter.

   ```
   [sp setAysncReplay:YES];  // Asynchronous replay
   [SUP101SUP101DB beginSynchronize:syncGroups
   withContext:userContext withUploadOnly:YES];
   ```

   With the `AsyncReplay` flag turned on, the client object API calls the `onSynchronize` callback method with an `SUPSynchronizationStatus_ASYNC_REPLAY_UPLOADED` status after the

upload phase, followed by an `SUPSynchronizationStatus_FINISHING` status. No data is pulled down to the device database as there is no download phase.

Note: Control returns immediately, without the replay results synchronized to the client. The `beginSynchronize` method is a nonblocking call. The following callback from the `SUPDefaultCallbackHandler` is invoked as the synchronization session progresses.

```
(SUPSynchronizationActionType)onSynchronize:
(SUPObjectList*)syncGroupList withContext:
(SUPSynchronizationContext *)context
```

Note: In later versions of the Mobile SDK, the `uploadOnly` parameter is available with the synchronous API.

It is not recommended to initiate synchronization without the `uploadOnly` parameter set to YES due to a race condition. You cannot predict if the download phase pulls down data/results pertaining to the operation replay. If there are multiple operation replays being uploaded, some may complete and get downloaded. When the batch of uploaded operation replays is completed, the server sends a notification triggering the `onSynchronize` callback with `SUPSynchronizationStatus_ASYNC_REPLAY_COMPLETED`. A synchronization is automatically initiated to pull down the data/result. You can allow this synchronization to continue or abort it by returning CANCEL to the `onSynchronize` callback associated with this synchronization. This `onSynchronize` callback has `SUPSynchronizationStatus_STARTING`.

In some use cases, you may perform a full (upload and download) synchronization.

6. Handle the callbacks associated with the completed operation replay if appropriate. Typically, you use these callbacks in your application to signal to the UI layer that the data/result are now available or pending status is to be turned off.
   - `- (void)onReplayFailure:(id)entityObject`
   - `- (void)onReplaySuccess:(id)entityObject`

Connecting Through a Relay Server

An iOS RBS client that connects through a Relay Server needs two different farm IDs: one for a messaging client connection to register the application, and the RBS connection for database synchronization.

In your iOS application, set up the messaging client and database connection through Relay Server. Note that, in most cases, the application template already contains settings for the RBS connection so you do not need to set any properties. The settings from the template are downloaded to the client after registration is completed. However, it may be necessary in a development environment to directly manipulate the settings.

1. To set up a messaging client connection, use:

```
SUPApplication * app = [SUPApplication getInstance];

// should be same as application id from SCC
```

```
[app setApplicationIdentifier:@”appId”];
SUPConnectionProperties* props = app.connectionProperties;
[props setServerName:serverName];
[props setPortNumber:80]; // or 443 for HTTPS
[props setNetworkprotocol:@"http"]; // or https for secure
connection
[props setUrlSuffix:@""];
[props setFarmId:@"farmIDMBS"];
SUPLoginCredentials* login = [SUPLoginCredentials getInstance];
login.username = @"userName"; // same as in Application Connection
login.password = nil;
props.loginCredentials = login;
props.activationCode = @"123"; // same as in Application
Connection
props.securityConfiguration = @"admin";
```

2. To set up a database connection:
   - If the application connection template on SCC is configured with all the required Relay Server information, application code only needs to do:

     ```
     SUPConnectionProfile *sp = [SUP101SUP101DB
     getSynchronizationProfile];
     [sp setUser:@"supAdmin"];
     [sp setPassword:@"password"];
     [sp setAsyncReplay:NO];
     ```

   - Otherwise, application code needs to fill all the Relay Server information before doing data synchronization:

     ```
     SUPConnectionProfile *sp = [SUP101SUP101DB
     getSynchronizationProfile];
     [sp setUser:@"supAdmin"];
     [sp setPassword:@"password"];
     [sp setAsyncReplay:NO];

     [sp setServerName:@"relayServerHostName"];
     [sp setPortNumber:443]; // or 80 for http
     [sp setNetworkProtocol:@"https"];
     // certificateName: this should come from the relay server and
     should be
     // included in the Resource folder of the XCode project
     [sp
     setNetworkStreamParams:@"trusted_certificates=certificateName;
     compression=zlib;url_suffix=urlSuffixRBS"];
     ```

     Note: `urlSuffixRBS` needs to match the exact string of Relay Server RBS url_suffix configuration.

The above code should be done before doing any data synchronization (including subscribe/ onlineLogin).

Setting Up Callbacks

Update your application to use callbacks available in SDK version 2.1 ESD #3 or later.

All callback methods are included in the `SUPCallbackHandler` protocol, and you must implement them in any class that directly implements the protocol without subclassing the default implementation in `SUPDefaultCallbackHandler`.

1. If you have directly implemented the `SUPCallbackHandler` protocol, you must implement all methods. In replication-based synchronization, there are several methods in the protocol that are specific to messaging-based synchronization, and will never be called.

   If you have created your callback handler as a subclass of `SUPDefaultCallbackHandler`, you can safely remove the following messaging-based synchronization callbacks, as the `SUPDefaultCallbackHandler` has empty implementations of all the required methods.

   - `beforeImport`, `onImport`, and `onImportSuccess`
   - `onLoginSuccess`
   - `onSubscribeFailure`, and `onSubscribeSuccess`
   - `onSuspendSubscriptionFailure`, and `onSuspendSubscriptionSuccess`
   - `onResumeSubscriptionFailure`, and `onResumeSubscriptionSuccess`
   - `onUnsubscribeFailure`, and `onUnsubscribeSuccess`
   - `onMessageException`
   - `onTransactionCommit`, and `onTransactionRollback`
   - `onRecoverFailure`, and `onRecoverSuccess`

   For a complete list of callbacks you can implement in your application, see *Developer Guide: iOS Object API Applications* > *Client Object API Usage* > *Callback and Listener APIs*. If the application uses `onImport` to generate a notification on instance creation and modification, you must change to use the `ChangeLog` facility in RBS. By default, `ChangeLog` is disabled and you can enable it using the generated database class. Once enabled, the server creates a change log record to identify each updated and deleted instance. Due to a limitation of RBS, the change log record only contains two operation types: update and delete. An update is actually an upsert (update/insert). Generating change logs can be expensive if you are downloading a large amount of data. For that reason, it is recommended that you disable the change log facility for initial or large delta synchronization. See *Generating Change Logs*.
2. If your application uses `SUPApplicationCallback`, update it to use these methods:

   Old method:

   ```
   - (void)onConnectionStatusChanged:(SUPInt)connectionStatus :
   (SUPInt)errorCode :(SUPNullableString)errorMessage;
   ```

   New method:

```
- (void)onConnectionStatusChanged:
(SUPConnectionStatusType)connectionStatus :(int32_t)errorCode :
(NSString*)errorMessage;
```

Old method:

```
- (void)onRegistrationStatusChanged:(SUPInt)registrationStatus :
(SUPInt)errorCode :(SUPNullableString)errorMessage;
```

New method:

```
- (void)onRegistrationStatusChanged:
(SUPRegistrationStatusType)registrationStatus :
(int32_t)errorCode :(NSString*)errorMessage;
```

Old method:

```
- (void)onDeviceConditionChanged :(SUPInt)condition;
```

New method:

```
- (void)onDeviceConditionChanged :
(SUPDeviceConditionType)condition;
```

Generating Change Logs

Use the Change Log API to generate change logs that are sent to the client after the synchronization.

In MBS, the application can use the information in the change logs to update its UI tables with new records and deletions. To do in the same in RBS, enable change logs in your application before synchronizing.

```
[SUP101SUP101DB enableChangeLog];
```

This method notifies you of all changes including the initial synchronization records. You may want to set a flag to indicate when the initial synchronization is done so you do not update the UI for all these initial records.

To set a flag, use code similar to this in your callback `onSynchronize` (`isCompleteSynchronize` is an application variable, set to true after the first synchronization is complete):

```
- (SUPSynchronizationActionType)onSynchronize:
(SUPObjectList*)syncGroupList withContext:
(SUPSynchronizationContext*)context
{
  if (context.status == SUPSynchronizationStatus_ERROR)
  {
      MBOLogError(@"onSynchronize failed for context %@ with
exception %@", context.userContext, [context.exception reason]);
  } else if (context.status == SUPSynchronizationStatus_FINISHING)
  {

      if (self.isCompleteSynchronize)
      {
          // Handle change log
        SUPObjectList *changeLogs = (SUPObjectList *)[SUP101SUP101DB
```

```
getChangeLogs:[SUPQuery getInstance]];
          if([changeLogs size] > 0)
          {
              [changeLogs retain];

              // delete these so we don't do updates later on these.
              [SUP101SUP101DB deleteChangeLogs];
              for (id<SUPChangeLog> cl in changeLogs)
              {
                  MBOLogDebug(@"Changelog: %@['%c', %ld]\n",
                              [SUP101SUP101DB getEntityName:[cl
entityType]],
                             [cl operationType], [cl surrogateKey]);

                // If your UI needs to find the actual object you can
                  // convert the entity name to a class.
                  Class entityClass =
NSClassFromString([SUP101SUP101DB  getEntityName:[cl entityType]]);
                  if (entityClass)
                  {
                   // You can either use the surrogate key or change
to the "keyToString" equivalent.
                      NSString *primaryKey = [SUPStringUtil
                                              toString_long:[cl
surrogateKey]];
                      NSString *type = ([cl operationType] == 'D'
                                        ) ? @"delete" : @"update";

                      // Notify your UI with NSNotification...
                  } //entityClass
                }
              [changeLogs release];
          }
      }
  }

  return SUPSynchronizationAction_CONTINUE;
}
```

Creating, Updating, or Deleting Records

In SDK version 2.1 ESD #2 applications, after creating, updating or deleting records, you called the `save` method to save the change to the local database, and called `submitPending` to send the change to the server. In SDK version 2.1 ESD #3 applications, after updating or creating records, you call the `save` and `submitPending` methods, and call `synchronize` to send the changes to the server.

1. In the 2.1 ESD #2 *Tutorial: iOS Object API Application Development*, locate this code:

   ```
   [newCustomer save];
   ```

   ```
   [newCustomer submitPending];
   ```

Note: In MBS, the generated operation from `submitPending` is automatically sent to the Unwired Server. In your RBS applications, you must instead invoke the `synchronize` method to send the record to the Unwired Server.

2. Add the following new code. You call `synchronize` to send the update or new record to the server. The call can be either synchronous or asynchronous.

```
@try {
  [SUP101SUP101DB synchronize];
}
@catch (NSException *exception) {
  MBOLogError(@"%@: %@", [exception name], [exception reason]);
}
```

   The above code examples synchronize the default group. Alternatively, you can synchronize based on the synchronization group the MBO belongs to.

```
NSString *customer_sg = [customer
metaData].synchronizationGroup;
[db synchronize:customer_sg];
```

Note: Unlike MBS, the `submitPending` method in RBS is a client-side only operation, but is still required before calling the database class's `synchronize` method, which sends the changes to the server.

Testing the Application

After you have transitioned your application to SDK version 2.1 ESD #3, test the application to ensure that it can establish messaging and database connections to the Unwired Server, perform an initial synchronization, and update the database.

Note: There is no data-transitioning solution. The data residing in the old device database is not used after the application is converted to RBS. The application users should submit all pending data to the Unwired Server using the existing MBS client application before the migration to the new 2.1 ESD #3 RBS application. See *Migration Strategies for 2.1 ESD #3* in *Migrating iOS Native Custom Applications* on page 13 After all the pending changes are synchronized to the Unwired Server, the application user needs to remove the old application and/or the older existing database on the device. If the old application is not removed from the device, the database for the old application will continue to reside on the device; this may double the space consumed on the device when the new application downloads records to the new database.

Start and test the client application:

1. Verify that no exceptions have been received from the code that subscribes to the database. If an exception has been received, check the connection profile.

   If no exception has been received, you have successfully established the connection to the database.
2. Verify that no exceptions have been received from the code that performs initial synchronization. If an exception has been received, check for any server-side issues in the

server log. Also ensure that there is no incompatibility in versions between the deployed package on the server and the generated code.

If no exception has been received, you have successfully performed an initial synchronization.

3. Verify that no exceptions have been received from the code that creates or updates a record. Also verify that you can view the update on the server.

Migration Paths for Windows and Windows Mobile Applications

Paths available to migrate Windows and Windows Mobile object API applications from earlier versions to the current version.

Application is Built with SDK Version	Migration Tasks
2.1 2.1.1	Migrate your application to the current version. See the migration instructions: • *Migrating Windows and Windows Mobile Applications to 2.1 ESD #2* on page 29 • *Migrating Windows and Windows Mobile Applications to 2.1 ESD #3* on page 29 • *Migrating Windows and Windows Mobile Applications to 2.2* on page 29
2.1.2	Migrate your application to the current version. See the migration instructions: • *Migrating Windows and Windows Mobile Applications to 2.1 ESD #3* on page 29 • *Migrating Windows and Windows Mobile Applications to 2.2* on page 29
2.1.3 2.2 2.2 SP01 2.2 SP02	Migrate your application to the current version. See the migration instructions: • *Migrating Windows and Windows Mobile Applications to 2.2* on page 29
2.3	No migration changes are required.

Migrating Windows and Windows Mobile Applications to 2.2

No migration changes are required for BlackBerry Object API applications; however, you may need to perform some migration steps to take advantage of new features.

A client library name change requires you to modify and recompile your Windows Mobile and Win32 applications. The version number is appended to the file name: `CMessagingClient.dll` has been renamed to `CMessagingClient2.2.2.dll`.

Migrating Windows and Windows Mobile Applications to 2.1 ESD #3

These changes are required for Windows and Windows Mobile applications being migrated from a version earlier than 2.1 ESD #3.

In 2.1 ESD#3, there are two new required libraries for Windows clients.

Rebuild your project to include additional references to the new libraries:

1. Add the following new libraries as items in the Visual Studio project. Set the "Build Action" to **Content** and "Copy to Output Directory" to **Copy always**.
   - For Windows:
     - libeay32.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\`.
     - ssleay32.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\`.
2. Verify that you have added all required references to your client projects as described in *Developer Guide: Windows and Windows Mobile Object API Applications* > *Development Task Flow for Object API Applications* > *Creating a Project* > *Adding References to a Mobile Application Project*.

See *Developer Guide: Windows and Windows Mobile Object API Applications* for information on developing your application.

Migrating Windows and Windows Mobile Applications to 2.1 ESD #2

These changes are required for Windows and Windows Mobile applications being migrated from a version earlier than 2.1 ESD #2.

Update and rebuild your application:

1. The Application APIs (in the `Application` class) are required for managing application registrations, connections, and context. Rewrite the initialization code in your application to use the Application APIs.
   For information on the `Application` interface, search for *Application APIs* in the Developer Guide for your platform.
2. Callbacks related to application events are now contained in a separate `ApplicationCallback` interface. Rewrite your application code to use this interface.

For information on the `ApplicationCallback` interface, search for *Callback and Listener APIs* in the Developer Guide for your platform.

3. Replication-based synchronization clients require two data channels: a data channel for data synchronization, and a messaging channel for sending registration and push notifications to the client. Update your port configuration for both channels. See *Sybase Control Center for Sybase Unwired Platform* > *Administer* > *Unwired Server* > *Server Properties*.
4. To continue using server-initiated synchronization, you must write code for handling notifications. If change notifications are enabled for synchronization groups, you can implement the `onSynchronize` callback method to monitor this condition, and either allow or disallow default background synchronization.

```
public int OnSynchronize(GenericList<ISynchronizationGroup>
groups, SynchronizationContext context)
{
  int status = context.Status;
  if (status == SynchronizationStatus.STARTING_ON_NOTIFICATION)
  {
    // There is changes on the synchronization group
    if (busy)
    {
      return SynchronizationAction.CANCEL;
    }
    else
    {
      return SynchronizationAction.CONTINUE;
    }
  }

  // return CONTINUE for all other status
  return SynchronizationAction.CONTINUE;
}
```

5. In 2.1 ESD #2, the new location of the required libraries is `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK\ObjectAPI`.

   Rebuild your project as follows:

   a. Reset the references of the following libraries for the appropriate device platform in the Visual Studio project according to the new location:
      - For Windows Mobile:
        - sup-client.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM`.
        - iAnywhere.Data.UltraLite.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM\Ultralite`.
        - iAnywhere.Data.UltraLite.resources.dll (several languages are supported) – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM\Ultralite\<language>`.

- For Windows:
  - sup-client.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32`.
  - iAnywhere.Data.UltraLite.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\Ultralite`.
  - iAnywhere.Data.UltraLite.resources.dll (several languages are supported) – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\Ultralite\<language>`.

b. Remove the following libraries for the appropriate device platform as items in the Visual Studio project. The libraries are no longer required.

- For Windows Mobile:
  - ulnet11.dll
  - mlcrsa11.dll (if HTTPS protocol is used)
  - PUtilTRU.dll
- For Windows:
  - ulnet11.dll
  - mlcrsa11.dll (if HTTPS protocol is used)
  - mlczlib11.dll (if using compression)

c. Add the following libraries for the appropriate device platform as items in the Visual Studio project. Set the "Build Action" to **Content** and "Copy to Output Directory" to **Copy always**.

- For Windows Mobile:
  - ulnet12.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM\Ultralite`.
  - mlcrsa12.dll (if HTTPS protocol is used) – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM\Ultralite`.
  - mlczlib12.dll (if HTTPS protocol is used) – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM\Ultralite`.
  - CMessagingClient.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\WM\<DeviceType>`. `<DeviceType>` can be Pocket PC or Smartphone as applicable.
- For Windows:
  - ulnet12.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\Ultralite`.
  - mlcrsa12.dll (if HTTPS protocol is used) – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\Ultralite`.

- mlczlib12.dll (if using compression) - from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32\Ultralite`.
- CMessagingClient.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32`.
- ECTrace.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32`.
- TravelerLib.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32`.
- zlib1.dll – from `<UnwiredPlatform_InstallDir>\MobileSDK\ObjectAPI\Win32`.

Migrate Hybrid Web Container Projects

No steps are required to migrate 2.2 SP02 Hybrid Web Container projects to SAP Mobile Platform version 2.3.

If you are migrating from a version earlier than 2.2 SP02, see *Developer Guide: Migrating to Sybase Mobile SDK 2.2 SP02*, and its updates, on Product Documentation, the *Migrate Hybrid Web Container Projects* section: *http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01857.0222/doc/html/mqu1352929931447.html*

Hybrid Web Container Compatibility Matrix

Compatibility between versions of the Hybrid Web Container and server, and Hybrid Web Container and Hybrid App applications.

Hybrid Web Container and Unwired Server/SAP Mobile ServerCompatibility

Client/ Hybrid Web Container	Unwired Server 2.1	Unwired Server 2.1 ESD #2	Unwired Server 2.1 ESD #3	Unwired Server 2.2 SP02	SAP Mobile Server 2.3
Hybrid Web Container 2.1	Yes	Yes	Yes	Yes	Yes
Hybrid Web Container 2.1 ESD #2	No	Yes	Yes	Yes	Yes

Client/ Hybrid Web Container	Unwired Server 2.1	Unwired Server 2.1 ESD #2	Unwired Server 2.1 ESD #3	Unwired Server 2.2 SP02	SAP Mobile Server 2.3
Hybrid Web Container 2.1 ESD #3	No	Yes	Yes	Yes	Yes
Hybrid Web Container 2.2 SP02	No	Yes	Yes	Yes	Yes
Hybrid Web Container 2.3	No	Yes	Yes	Yes	Yes

There was no 2.1 ESD #1 Hybrid Web Container; 2.1 ESD #1 shipped with 2.1 Mobile Workflow clients.

Note:

- Yes – the client application built in this SDK version is supported in the server version (minor adjustments may be necessary to ensure the application runs correctly; see the migration details for the appropriate application type, if any).
- No – the client application built in this SDK version is not supported in the server version.
- Server version – refers to the server version to which the original package is migrated, not the newly deployed package.

Hybrid Web Container and Hybrid App Compatibility

Client/ Hybrid Web Container	Hybrid App 2.1	Hybrid App 2.1 ESD #2	Hybrid App 2.1 ESD #3	Hybrid App 2.2 SP02	Hybrid App 2.3
Hybrid Web Container 2.1	Yes	No	No	No	No
Hybrid Web Container 2.1 ESD #2	Yes	Yes	No	No	No
Hybrid Web Container 2.1 ESD #3	Yes	Yes	Yes	No	No
Hybrid Web Container 2.2 SP02	Yes	Yes	Yes	Yes	No

Client/ Hybrid Web Container	Hybrid App 2.1	Hybrid App 2.1 ESD #2	Hybrid App 2.1 ESD #3	Hybrid App 2.2 SP02	Hybrid App 2.3
Hybrid Web Container 2.3	Yes	Yes	Yes	Yes	Yes

There was no 2.1 ESD #1 Hybrid Web Container; 2.1 ESD #1 shipped with 2.1 Mobile Workflow clients.

Note:

- Yes – the client application built in this SDK version is supported in the server version (minor adjustments may be necessary to ensure the application runs correctly; see the migration details for the appropriate application type, if any).
- No – the client application built in this SDK version is not supported in the server version.
- Server version – refers to the server version to which the original package is migrated, not the newly deployed package.

Migrate Hybrid Apps to JavaScript API

There have been some ongoing changes to the JavaScript API for Hybrid Apps to accommodate new Hybrid Web Container functionality, and make it more flexible and easy to use. These changes were first made available in version 2.2, and became the default generation method in version 2.2 SP02.

Existing applications can still use the older generated JavaScript API; if you want to use the new Hybrid Web Container functionality or the new API, there are certain steps you must perform to successfully migrate existing applications.

The new JavaScript API includes:

- New Hybrid Web Container functionality
- Access to third-party frameworks and designers by generating fewer Hybrid App-specific files
- Use of a global namespace-style variable to help resolve naming conflicts
- A more logical file layout
- Naming change from Workflow to Hybrid App

Although the new files were available for 2.2 SP01, they were not used as the default generation option. Beginning with 2.2 SP02, the Hybrid App Designer began to directly support the new Hybrid Web Container functionality, and therefore the new JavaScript API became the default in the generation wizard.

In this way, the Hybrid App Designer could move forward with the updated JavaScript API, while still allowing previously written apps to successfully migrate without changes, giving developers the choice to generate the backward-compatible API from earlier versions.

Note: When you generate an application that was developed using the older API, be sure to enable the **Use backwards-compatible API for generation (deprecated)** option, under the Advanced Options section of the generation wizard.

Manual Migration Tasks

Migrating a Hybrid App or Workflow to the new JavaScript API requires the developer to perform a few manual steps.

API.js & Utils.js

Reapply any custom changes made to `API.js` and `Utils.js` in the previous Hybrid App to the new Hybrid App.

Method Calls in Custom Code

Some methods used in previous versions may have been deprecated, renamed, removed, or relocated within the “hwc” global namespace. In your custom code, resolve any calls to these methods.

A new `Custom.js` file is automatically generated, but it will be empty. Move any functionality from the previous version of `Custom.js` to the new file, paying attention to any method name changes.

Customized Files

Move any custom CSS, JavaScript, image, or other files in the generated folder of the previous Hybrid App or Workflow to the generated folder for the new Hybrid App.

Table 1. Generated Folder and File Name Changes

Generated Folder Name	2.1 ESD #3 (old)	Starting with 2.2 SP02, and 2.3 (new)
`html/css`	No changes	No changes
`html/css/bb`	No changes	No changes
`html/css/bb/img`	No changes	No changes
`html/css/iphone`	No changes	No changes
`html/css/iphone/images`	No changes	No changes

Generated Folder Name	2.1 ESD #3 (old)	Starting with 2.2 SP02, and 2.3 (new)
`html/css/jquery`	`jquery.mobile-1.0.css` (removed)	`jquery.mobile-1.1.0.css` (added)
`html/css/jquery/images`		`ajax-loader.gif` (added)
`html/js`	`json2.js` (removed)	`datajs-1.0.3.js` (new) `hwc-api.js` (new) `hwc-comms.js` (new) `hwc.utils.js` (new) `HybridApp.js` (replaces `Workflow.js`) `PlatformIdentification.js` (new)
`html/js/android`	`Phonegap-1.4.1.javascript` (removed)	`cordova-2.0.0.javascript` (added)
`html/js/ios`	`Phonegap-1.4.1.javascript` (removed)	`cordova-2.0.0.javascript` (added)
`html/js/jquery`	`jquery-1.6.4.js` (removed) `jquery.mobile-1.0.js` (removed)	`jquery-1.7.1.js` (added) `jquery.mobile-1.1.0.js` (added)

Generated Folder Name	2.1 ESD #3 (old)	Starting with 2.2 SP02, and 2.3 (new)
`html/js/widgets`	`sy.ui.iphone.isc-roll.js` (removed)	`sy.ui.iphone.isc-roll4Lite.js` (added) `sy.ui.iphone.pick-er.js` (added) `sy.ui.iphone.sig-nature.js` (added) `sy.ui.iphone.util.js` (added) `sy.ui.js` (added)
`html/css/makit`	N/A	New directory added for 2.2 SP02
`html/images/makit`	N/A	New directory added for 2.2 SP02
`html/js/makit`	N/A	New directory added for 2.2 SP02
`html/js/wm`	N/A	New directory added for 2.2 SP02
`html/js/blackberry`	N/A	New directory added for 2.2 SP02

Check for Output Message

During project and application generation, check for messages in the console; these messages provide valuable tips related to migrating your application.

Generated Application Differences

There are some important differences and concepts that have changed from the earlier (often called "legacy") API and the new API. Use this background information to successfully migrate your own applications.

See *Migrating Hybrid Apps to JavaScript API* on page 43 for an example procedure that migrates a customized application into the new API.

Output Directories

The generated output directory has changed.

Earlier versions generated the files into:

```
\Generated Workflow\project_name\html\...
```

The current version generates the files into:

```
\Generated Hybrid App\project_name\html\...
```

The "hwc" Namespace

The SAP Mobile Platform JavaScript API implementation emulates the namespace concept.

JavaScript does not use the concept of a true namespace, but you can achieve the same result by placing functions and objects with a new function. (This is often called a "JavaScript Namespace" for simplicity). The SAP Mobile Platform implementation uses the JavaScript "Immediately-Invoked Function Expression" (IIFE—pronounced "iffy" like in "jiffy") pattern to contain the SAP Mobile Platform API in the global object named "hwc". The "hwc" object provides a lexical scope, which is the JavaScript shorthand for what other languages term a "namespace".

This lexical scope isolates the SAP Mobile Platform API (in the same way as a namespace) and provides tools for minification and other processes. This change is reflected throughout the code. A few JavaScript functions have proxy versions in the global namespace; however SAP recommends that you use the versions in the "hwc" namespace.

We have attempted to not move code around unnecessarily, to facilitate developers using "difference scans" to identify changes, and to understand where to place their existing customizations.

In many cases, you need only wrap the entire contents of a file with the "hwc" object definition. The files that have more extensive changes are described in later topics.

Name Changes

Several coding-level name changes have been implemented.

Workflow Changed to HybridApp

The phrase `Workflow` has been replaced with `HybridApp`, both in the directory name, as well as in the function code.

- `onWorkflowLoad()` is now `hwc.onHybridAppLoad()`
- `customBeforeWorkflowLoad()` is now `hwc.customBeforeHybridAppLoad()`
- `customAfterWorkflowLoad()` is now `hwc.customAfterHybridAppLoad()`

Note: Although the change from `Workflow` to `HybridApp` occurs in many places, it is not pervasive. The following in particular:

- The file and type `WorkflowMessage`.
- The global variable `workflowMessage` in the file `Utils.js`.

Workflow Expanded to Data Message
The meaning "workflow" message has been expanded to a generic "data"' message.

- `getWorkflowMessage()` is now `hwc.getDataMessage()`.
- `processWorkflowMessage()` now includes the namespace `hwc.processDataMessage()`.
  Custom implementations that override `processWorkflowMessage()` must now override `hwc.processDataMessage()`.

  Note: For a better understanding of this change, examine the implementation in `hwc-comms.js`.
- `customBeforeProcessWorkflowMessage()` is now `hwc.customBeforeProcessDataMessage()`.
- `customAfterProcessWorkflowMessage()` is now `hwc.customAfterProcessDataMessage()`.

Debug Logging
`logToWorkflow()` is now `hwc.log()`.

Note: The legacy global name still exists.

Closing Applications
`closeWorkflow()` is now `hwc.close()`.

Note: The legacy global name still exists.

New Files

In the new API, many functions have been split into new files. This change helps to both isolate functionality, and localize areas of engagement.

The new files are:

- `html/js/PlatformIdentification.js` – the device and platform identification logic like `hwc.isIOS()`.
- `html/js/hwc-comms.js` – the functions that communicate with the container, and often ultimately to SAP Mobile Server.
- `html/js/hwc-api.js` – the functions that work with application query, and control of the container itself. Use these functions to restyle container behavior.
- `html/js/hwc-utils.js` – various utility functions used by both the container and application.

Compare old and new HTML files, and update file source references to point to these new files.

Files and Functions: Platform Identification

The routines that handle platform identification, like `isIOS()`, have been moved from `API.js` into `PlatformIdentification.js`.

These routines are now evaluated only once at application start-up, which gives the application an overall performance boost.

The routines retain the legacy API version, in the global namespace, along with the new API version in the "hwc" namespace. Even though SAP recommends a scoped routine like `hwc.isIOS()`, the global routine `isIOS()` is still available.

Files and Functions: Container API

To separate functionality between layers, the container-oriented parts of the API have been moved into separate files. This has also split up some functions that provided communication between the Hybrid Web Container and the JavaScript application.

The container-oriented files use an "hwc" prefix, and include:

- **hwc-api.js** – public API for any Hybrid Web Container application.
- **hwc-comms.js** – communications support for Hybrid Web Container applications.
- **hwc-utils.js** – miscellaneous support routines, including both internal worker functions, and other routines that are available for Hybrid Web Container applications.

In cases where a function has been split between a HybridApp aspect and a container-specific aspect, the suffix "_CONT" has been added to the container-specific function. The commonly used function **hwc.doOnlineRequest()** first performs HybridApp-specific processing, such as custom callbacks, then delegates **hwc.doOnlineRequest_CONT()** for the actual container-side HTTP call. This same technique applies to these routines:

- **onWorkflowLoad()** has been replaced by:
  - **hwc.onHybridAppLoad ()** in **Utils.js**, which delegates to:
  - **hwc.onHybridAppLoad_CONT()** in **hwc-utils.js**.
- **addNativeMenuItemsForScreen ()** has been replaced by:
  - **hwc.addNativeMenuItemsForScreen()** in **Utils.js**, which delegates to:
  - **hwc.addNativeMenuItem_CONT()** in **hwc-utils.js**.
- **handleCredentialChange()** has been replaced by:
  - **hwc.handleCredentialChange()** in **Utils.js**, which delegates to:
  - **hwc.handleCredentialChange_CONT()** in **hwc-utils.js**.
- **doOnlineRequest()** has been replaced by:
  - **hwc.doOnlineRequest()** in **API.js**, which delegates to:
  - **hwc.doOnlineRequest_CONT()** in **hwc-comms.js**.
- **doOnlineRequest()** has been replaced by:
  - **hwc.doOnlineRequest()** in **API.js**, which delegates to:
  - **hwc.doOnlineRequest_CONT()** in **hwc-comms.js**.

- **doSubmitWorkflow()** has been replaced by:
  - **hwc.doSubmitWorkflow()** in **API.js**, which delegates to:
  - **hwc.doSubmitWorkflow_CONT()** in **hwc-comms.js**.
- **setScreenTitle()** has been replaced by:
  - **hwc.setScreenTitle()** in **API.js**, which delegates to:
  - **hwc.setScreenTitle_CONT()** in **hwc-comms.js**.
- **addMenuItem()** has been replaced by:
  - **hwc.addMenuItem()** in **API.js**, which delegates to:
  - **hwc.addMenuItem_CONT()** in **hwc-comms.js**.
- **showAttachmentContents()** has been replaced by:
  - **hwc.showAttachmentContents()** in **API.js**, which delegates to:
  - **hwc.showAttachmentContents_CONT()** in **hwc-comms.js**.
- **showAttachmentFromCache()** has been replaced by:
  - **hwc.showAttachmentFromCache()** in **API.js**, which delegates to:
  - **hwc.showAttachmentFromCache_CONT()** in **hwc-comms.js**.
- **doAttachmentDownload()** has been replaced by:
  - **hwc.doAttachmentDownload()** in **API.js**, which delegates to:
  - **hwc.doAttachmentDownload_CONT()** in **hwc-comms.js**.
- **doActivateWorkflow()** has been replaced by:
  - **hwc.doActivateWorkflow()** in **API.js**, which delegates to:
  - **hwc.doActivateWorkflow_CONT()** in **hwc-comms.js**.
- **doCredentialsSubmit()** has been replaced by:
  - **hwc.doCredentialsSubmit()** in **API.js**, which delegates to:
  - **hwc.doCredentialsSubmit_CONT()** in **hwc-comms.js**.

processWorkflowMessage()

There are some changes to consider if you override the global **processWorkflowMessage()** routine, which is required in many use cases.

In the older version, the global **processWorkflowMessage()** routine (in **Utils.js**), included either inline code changes, or customizations in the **customBefore…()** and **customAfter…()** callback routines. In the new API, inline code changes reside in the **hwc.processDataMessage()** routine (also in **Utils.js**). The callback routines still have similar names.

Old API:

- **processWorkflowMessage()** – in old file **Utils.js**
- **customBeforeProcessWorkflowMessage()** – in old file **custom.js**
- **customAfterProcessWorkflowMessage()** – in old file **custom.js**

New API:

- **hwc.processDataMessage()** – in file **Utils.js**
- **hwc.customBeforeProcessDataMessage ()** – in file **custom.js**
- **hwc.customAfterProcessDataMessage ()** – in file **custom.js**

Custom Callback Handlers

The file and techniques outlined in **js/Custom.js** are all still applicable. However, all the routines have been placed into the "hwc" namespace.

At a high level, all the routines have been wrapped in the "hwc" object (using JavaScript Immediately-Invoked Function Expression, or IIFE). Then the changes from `Workflow` to `HybridApp` have been applied. Function names have been retained except:

Old Global API	New API
customBeforeWorkflowLoad()	**hwc.customBeforeHybridAppLoad()**
customAfterWorkflowLoad()	**hwc.customAfterHybridAppLoad()**
customBeforeProcessWorkflowMessage()	**hwc.customBeforeProcessDataMessage()**
customAfterProcessWorkflowMessage()	**hwc.customAfterProcessDataMessage()**

Miscellaneous Changes

Several procedural changes have been implemented in the new API.

Applications must call:

- **hwc.setLoggingCurrentLevel()**
- **hwc.setLoggingAlertDialog()**
- **hwc.setReportErrorFromNativeCallback()**

These calls are already handled in the **hwc.onHybridAppLoad()** function, in the new **Utils.js** file. This fragment code example from the **hwc.onHybridAppLoad()** function shows the change; you may need to make changes if you customized the original function:

```
logLevel = hwc.getURLParam("loglevel");
hwc.setLoggingCurrentLevel( logLevel );        // store the log level

// set the preferred user alert dialog
hwc.setLoggingAlertDialog( hwc.showAlertDialog );

// the preferred native error callback function
hwc.setReportErrorFromNativeCallback( reportErrorFromNative );
if (logLevel >= 4) { hwc.log("entering onHybridAppLoad()", "DEBUG",
false); }
```

Variable Name Change

Some variable names have been changed to make their use more clear.

Table 2. Renamed Variable

Old Name	New Name
`previousScreenName[]`	`hwc.previousScreenKeyStack[]`

Migrating Hybrid Apps to JavaScript API

Migrate a customized legacy workflow or Hybrid App from 2.1.*x* to the new JavaScript API, available starting in 2.2 SP02.

Keep in mind that customizations vary, so these steps cannot be specific for every situation. Use the migration concepts and reference material in previous topics to make migration and customization decisions.

1. *Preparing to Migrate*

   Prepare to migrate your legacy workflow or Hybrid App.
2. *Regenerating the Application (Old API)*

   Regenerate the application using the old API. This creates a clean version of the project that you can use for comparison when you integrate customizations.
3. *Regenerating the Application (New API)*

   Regenerate the application using the new API. This creates a clean version of the project that you can use when you integrate customizations.
4. *Integrating Customizations*

   Integrate customizations into the clean version of the project generated with the new API. This example suggests a migration approach; adapt this method to your unique customizations.

Preparing to Migrate

Prepare to migrate your legacy workflow or Hybrid App.

Prerequisites

SAP recommends these tools:

- Directory-wide comparison program, such as Beyond Compare (*http://www.scootersoftware.com/*); ideally, a version that supports both comparison and merging.
- JavaScript syntax checking environment, such as Eclipse-JEE; or the Chrome or FireFox debug console into which you can load the HTML (which brings in all the JavaScript files).

Such a tool can help you identify typing errors, and errors arising from the "hwc" namespace identifier changes.

Task

1. Back up any customized JavaScript and CSS files. For example, save the files to `\myCustomizedApplication`.

   This directory should have the subdirectories `\html` and `\html\js`, just like what is generated by Hybrid App Designer.
2. Make sure the generated output directory is empty: `\Generated Hybrid App \project_name`.

Regenerating the Application (Old API)

Regenerate the application using the old API. This creates a clean version of the project that you can use for comparison when you integrate customizations.

1. Use your SAP Mobile Platform project (including the MBOs and Hybrid App Designer XBW files) to generate a new Hybrid App package. You need not deploy the package to SAP Mobile Server.

   In the generation wizard, use these defaults:
   - Output directory – use the default location to help prevent confusion.
   - Under Advanced Options, Use backwards-compatible API for generation (deprecated) – by default, this option is enabled, which generates a clean version of the project using the old API. The wizard does not create a new directory for this project.
2. Rename the newly populated, generated output directory for later comparison, for example, `\Generated Hybrid App\project_name_OLD`.

Regenerating the Application (New API)

Regenerate the application using the new API. This creates a clean version of the project that you can use when you integrate customizations.

Use your SAP Mobile Platform project (including the MBOs and Hybrid App Designer XBW files) to generate a new Hybrid App package. You need not deploy the package to SAP Mobile Server.

In the generation wizard, use these defaults:

- Output directory – the default location is `\Generated Hybrid App \project_name`. Using the default location helps prevent further confusion.
- Under Advanced Options, Use backwards-compatible API for generation (deprecated) – by default, this option is not enabled. Because you renamed the project when you generated it using the old API, you can use the default option now to create a clean project version using the new API.

Integrating Customizations

Integrate customizations into the clean version of the project generated with the new API. This example suggests a migration approach; adapt this method to your unique customizations.

Prerequisites

Verify there are three copies of the application (the directory names for your application may vary):

- Original customized version in `\myCustomizedApplication`
- Legacy API version in `\Generated Hybrid App\`*`project_name`*`_OLD`
- New API version in `\Generated Hybrid App\`*`project_name`*

Task

1. Compare the original, customized application with the legacy API version of the application just generated, and compile a list of areas on which to focus the migration effort.

   A typical scenario includes many changes in the standard customization file, **Custom.js**, new CSS files, some new JS files, and some customizations in **API.JS** and **Timezone.js**.
2. Copy the CSS files and new JS files from the original directory into the new API directory, and edit the HTML file or JS module loaders for the file references. Use the same procedure you used for your original additions.
3. Use a directory-wide comparison program to compare all the files in the original and legacy directories. This means comparing the two directories:

   - Original, customized version in:

     ```
     \myCustomizedApplication
     ```
   - Clean legacy API version in:

     ```
     \Generated Hybrid App\project_name_OLD
     ```

   Differences should include only:
   - The SAP Mobile Platform bug fixes between your last version and the current version of SAP Mobile Platform.
   - The code changes originating from your own modifications to the legacy API.

     As described above, differences are most likely to be in **Custom.js**, with some changes in **API.js** and **Timezone.js**. There will be other differences throughout the files, some from your customizations, and some SAP Mobile Platform evolutionary changes and Service Pack bug fixes.
4. Open another session of the directory-wide comparison program, ideally a version that allows both comparison and merging. Open your original version, and the new but still clean API version to compare the two directories:

- Original, customized version in:
  `\myCustomizedApplication`
- New API version in:
  `\Generated Hybrid App\project_name`

You might see many differences, however the important ones are the same as those identified in the legacy API comparison (step 3).

5. Make the appropriate changes in **Custom.js**, and merge the changes function by function.

   The order and most of the function signatures have been retained in the new API, which should simplify this process. Keep in mind that:
   - The new API functions are in the "hwc" namespace. This affects any code that uses the API functions; any such code must use the "hwc." namespace identifier.
   - All other changes, such as the `Workflow` to `HybridApp` name change, must also be addressed.
6. In the **Timezone.js** file, make the appropriate changes, including any evolutionary changes, and your customizations. Keep in mind that a Service Pack fix might have corrected a bug that you fixed using a different approach.
7. Make any appropriate changes in the **API.js** file.

   Any customizations you have made to this file will be challenging, due to both the size of the file, and to separating the "container communication" aspect into **hwc-comms.js**. The new API keeps functions in the same general order, which should make migrating from an older code line more efficient.
8. Address any other differences in your legacy API comparison, and migrate them into the new API version.
9. Launch the application in an environment you can monitor, such as FireFox or Chrome (or even the Android Emulator); test the application and fix any errors.

   Set the logging level all the way up, and continue the process of finding any missed "'hwc"' references and any other missed items. During this process, examine the server logs for enter/exit function notifications, to help find errors.

   Once the application is migrated to the new API, it should:
   - Run faster, due to caching device data like in **PlatformIdentification.js**.
   - Be smaller and more efficient, due to better minification and runtime engine optimizations from the localization and compartmentalization.
   - Use fewer global variables and functions, and use a cleaner global namespace.
   - More easily integrate with third-party packages, because of the cleaner global namespace.
   - Be easier to maintain and extend, due to separation of the container API and the application level API functions.

Android

No migration changes are required for Android Hybrid Apps.

Hybrid Web Container Migration Paths for Android

Supported Hybrid Web Container (HWC) migration paths on Android.

Table 3. Android Migration Paths

	2.1 HWC	2.1 ESD #2 HWC	2.1 ESD #3 HWC	2.2 SP02 HWC	2.3 HWC
2.1 HWC	N/A	In-place upgrade	Coexist	Coexist	Coexist
2.1 ESD #2 HWC	N/A	N/A	Coexist	Coexist	Coexist
2.1 ESD #3 HWC	N/A	N/A	N/A	In-place upgrade	In-place upgrade
2.2 SP02 HWC	N/A	N/A	N/A	N/A	In-place upgrade
2.3 HWC	N/A	N/A	N/A	N/A	N/A

Note: There was no 2.0 or 2.1 ESD #1 Android Hybrid Web Container.

- N/A – not applicable.
- Coexist – the application is not upgraded; multiple versions of the application can coexist.
- In-place upgrade – the application is upgraded to the new version (you must modify the application to add new features).

BlackBerry

No migration changes are required for BlackBerry Hybrid Apps.

Hybrid Web Container Migration Paths for BlackBerry

Supported Hybrid Web Container (HWC) migration paths on BlackBerry.

Table 4. BlackBerry Migration Paths

	2.1 HWC	2.1 ESD #2 HWC	2.1 ESD #3 HWC	2.2 SP02 HWC	2.3 HWC
2.1 HWC	N/A	In-place upgrade	In-place upgrade	Coexist	Coexist
2.1 ESD #2 HWC	N/A	N/A	In-place upgrade	Coexist	Coexist
2.1 ESD #3 HWC	N/A	N/A	N/A	Coexist	Coexist
2.2 SP02 HWC	N/A	N/A	N/A	N/A	In-place upgrade
2.3 HWC	N/A	N/A	N/A	N/A	N/A

Note: There was no 2.0 ESD #1 or 2.1 ESD #1 for BlackBerry Hybrid Web Container.

- N/A – not applicable.
- Coexist – the application is not upgraded; multiple versions of the application can coexist.
- In-place upgrade – the application is upgraded to the new version (you must modify the application to add new features).

iOS

No migration changes are required for iOS Hybrid Apps.

Hybrid Web Container Migration Paths for iOS

Supported Hybrid Web Container migration paths on iOS, including paths for applications downloaded from the Apple App Store and those built from source code.

iOS Migration Paths (Applications Downloaded from the Apple App Store)
This matrix identifies the supported Hybrid Web Container migration or the iOS container downloaded from the Apple App store.

	2.1 HWC	2.1 ESD #2 HWC	2.1 ESD #3 HWC	2.2 SP02 HWC	2.3 HWC
2.1 HWC	N/A	Coexist	Coexist	Coexist	Coexist
2.1 ESD #2 HWC	N/A	N/A	In-place upgrade	In-place upgrade	In-place upgrade
2.1 ESD #3 HWC	N/A	N/A	N/A	In-place upgrade	In-place upgrade
2.2 SP02 HWC	N/A	N/A	N/A	N/A	In-place upgrade
2.3 HWC	N/A	N/A	N/A	N/A	N/A

Note: There was no 2.1 ESD #1 Hybrid Web Container.

- N/A – not applicable.
- Coexist – the application is not upgraded; multiple versions of the application can coexist.
- In-place upgrade – the application is upgraded to the new version (you must modify the application to add new features).

iOS Migration Paths (Applications Built from Source Code)

This matrix identifies the supported Hybrid Web Container migration for the iOS container that one builds from the supplied source code while keeping the same "bundle ID" between versions.

	2.1 HWC	2.1 ESD #2 HWC	2.1 ESD #3 HWC	2.2 SP02 HWC	2.3 HWC
2.1 HWC	N/A	In-place upgrade	In-place upgrade	In-place upgrade	In-place upgrade
2.1 ESD2 HWC	N/A	N/A	In-place upgrade	In-place upgrade	In-place upgrade
2.1 ESD3 HWC	N/A	N/A	N/A	In-place upgrade	In-place upgrade
2.2 SP02 HWC	N/A	N/A	N/A	N/A	In-place upgrade
2.3 HWC	N/A	N/A	N/A	N/A	N/A

Note: There was no 2.1 ESD #1 Hybrid Web Container.

Windows Mobile

No migration changes are required for Windows Mobile Hybrid Apps.

Hybrid Web Container Migration Paths for Windows Mobile

Supported Hybrid Web Container (HWC) migration paths on Windows Mobile.

Table 5. Windows Mobile Migration Paths

	2.1 HWC	2.1 ESD #2 HWC	2.2 SP02 HWC	2.3 HWC
2.1 HWC	N/A	In-place upgrade	Coexist	Coexist
2.1 ESD #2 HWC	N/A	N/A	Coexist	Coexist
2.2 SP02 HWC	N/A	N/A	N/A	In-place upgrade
2.3 HWC	N/A	N/A	N/A	N/A

Note: There was no new 2.1 ESD #1 or 2.1 ESD #3 for Windows Mobile Hybrid Web Container; 2.1 ESD #3 shipped with 2.1 ESD #2 Windows Mobile clients.

- N/A – not applicable.
- Coexist – the application is not upgraded; multiple versions of the application can coexist.
- In-place upgrade – the application is upgraded to the new version (you must modify the application to add new features).

Migrate OData Applications

No migration changes are required for OData applications; however you may need to perform migration steps to take advantage of new features.

If you are migrating from a version earlier than 2.2 SP02, see *Developer Guide: Migrating to Sybase Mobile SDK 2.2 SP02*, and its updates, on Product Documentation, the *Migrate OData Applications* section: *http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01857.0222/doc/html/mqu1352854260620.html*

OData Client Compatibility Matrix

Compatibility between versions of OData clients and SAP Mobile Server. Also compatibility between versions of REST SDK clients and SAP Mobile Server.

OData SDK Client and Unwired Server/SAP Mobile Server Version Compatibility

OData SDK Client	Unwired Server 2.1	Unwired Server 2.1 ESD #1	Unwired Server 2.1 ESD #2	Unwired Server 2.1 ESD #3	Unwired Server 2.2 SP02	SAP Mobile Server 2.3
OData SDK Client 2.1	Yes	Yes	Yes	Yes	Yes	Yes
OData SDK Client 2.1 ESD #1	No	Yes	Yes	Yes	Yes	Yes
OData SDK Client 2.1 ESD #2	No	Yes	Yes	Yes	Yes	Yes
OData SDK Client 2.1 ESD #3	No	Yes	Yes	Yes	Yes	Yes
OData SDK Client 2.2 SP02	No	Yes	Yes	Yes	Yes	Yes
OData SDK Client 2.3	No	Yes	Yes	Yes	Yes	Yes

Note:

- Yes – the client application built in this SDK version is supported in the server version (minor adjustments may be necessary to ensure the application runs correctly; see the migration details for the appropriate application type, if any).
- No – the client application built in this SDK version is not supported in the server version.
- Server version – refers to the server version to which the original package is migrated, not the newly deployed package.

REST SDK Client and Unwired Server/SAP Mobile Server Version Compatibility

REST SDK client	Unwired Server 2.1.3	Unwired Server 2.2 SP01	Unwired Server 2.2 SP02	Unwired Server 2.2 SP03	SAP Mobile Server 2.3	SAP Mobile Server 2.3 SP02
REST Client 2.2 SP03	No	Yes	Yes	Yes	Yes	Yes

Note:

- Yes – the client application built in this SDK version is supported in the server version (minor adjustments may be necessary to ensure the application runs correctly; see the migration details for the appropriate application type, if any).
- No – the client application built in this SDK version is not supported in the server version.
- Server version – refers to the server version to which the original package is migrated, not the newly deployed package.

Android

No migration changes are required for OData Android applications.

BlackBerry

No migration changes are required for OData BlackBerry applications.

iOS

No migration changes are required for OData iOS applications.

OData SDK API Changes in Version 2.3

The HTTP REST client libraries are available for OData applications in 2.3 SP02.

HTTP REST Client Libraries in 2.3 SP02

The HTTP REST client libraries are available with 2.3 SP02, which enable you to implement REST services in OData applications (Android and iOS). The REST SDK libraries enable consumption of SAP Mobile Platform REST services with pure HTTP (by default in on-premise) connectivity. The REST SDK provides simplified APIs for registration, exchange

settings between client and server, and end-to-end tracing. The SDK also supports native push notifications.

Table 6. New HTTP REST Classes for OData

Classes	Platform
• `ClientConnection` • `UserManager` • `AppSettings`	Android
• `SMPClientConnection` • `SMPUserManager` • `SMPAppSettings`	iOS

Documented in: *Developer Guide: OData SDK*, see *REST SDK API Reference* (Android and iOS)

Migrate OData Applications to REST API

Migrate messaging-based (sometimes called iMO-based) OData applications to REST API-based, to take advantage of REST services capabilities. This enables you to run mobile applications on-premise and in the cloud.

Prerequisites

- Import the new REST client libraries from the OData SDK into your Android or iOS development environment.
- Arrange access to a test environment for both on-premise and cloud testing.

Task

1. In your development environment, modify the messaging-based (sometimes called iMO) application logic to use REST-based services.

   Some areas you may need to address:
   - Registration
   - Settings exchange
   - Request response
   - End-to-end tracing
   - Native push notifications

- For the cloud, the application may support CAPTCHA if required.

For supporting information:

- For information related to migrating OData applications to REST API, see *Guidelines for On Premise and Cloud Applications*.
- For API information for all of the above, which are different for the REST SDK, see *Developer Guide: OData SDK*:
  - *Development Task Flow Using REST SDK (HTTP Channel) – iOS section*
  - *Development Task Flow Using REST SDK (HTTP Channel) – Android section*
- For new API information, see *OData SDK API Changes in Version 2.3*.

2. Recompile the application.
3. Test the application in a device simulator or emulator, and in the test environment (both on-premise and cloud configurations). Make modifications as needed.

   For useful information for testing:

   - **iOS applications –**
     - *Developer Guide: OData SDK* (iOS section):
       - *Testing Applications*
       - *Deploying Applications to Devices*
     - *Tutorial: iOS OData Application Development with REST Services*, *Deploying the Device Application on iPhone Simulator*
   - **Android applications –**
     - *Developer Guide: OData SDK* (Android section), *Deploying Applications to Devices*
     - *Tutorial: Android OData Application Development with REST Services*, *Running your Android OData Application*
4. Deploy the application to the production environment.

Guidelines for On Premise and Cloud Applications

Consider these on-premise and cloud guidelines when migrating OData applications to REST API. The guidelines may require coding changes to your application.

- A cloud application may support CAPTCHA if required. If CAPTCHA is enabled, the application must be able to process the CAPTCHA challenge.
- Applications that support both on-premise and cloud must incorporate logic to determine the system to which the application should connect. This may extend to user interface elements that prompt the user to identify the correct system. You can set up an application to determine the system via the provided Server URL, but you must implement the logic for this.

- New versions of resource bundles on the server are not automatically pushed to the applications in the cloud scenario. You must add application logic to request new resource bundles from the server if needed.
- Security Configuration HTTP Headers are not supported, and are ignored by the cloud.
- The cloud always uses HTTPS, whereas it is optional in on-premise scenarios.
- Application connection registration is required in the cloud, whereas it is optional for on-premise scenarios.
- Domains are not supported in the cloud.
- Application connection templates are not supported in the cloud.
- A subset of the SAP Mobile Platform PUSH registration settings is available for the cloud scenario, from the full set available for the on premise scenario.
- Since the cloud enables for cross-site request forgery (XSRF) attacks, applications used in the cloud must include XSRF token handling logic, if the back-end service demands it.

Migrate REST API Applications

No migration changes are required for REST API applications.

Index