



Developer Guide: Hybrid Apps

SAP Mobile Platform 2.3 SP02

DOCUMENT ID: DC01920-01-0232-01

LAST REVISED: May 2013

Copyright © 2013 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

Contents

Introduction to Developer Guide for Hybrid Apps	1
Documentation Roadmap for SAP Mobile Platform	1
Introduction to Developing Hybrid Apps With SAP	
Mobile Platform	3
Hybrid Web Container Architecture	3
Hybrid App Development Task Flow	6
Hybrid App Development Task Flow Using Third-Party Web Frameworks and MBOs	6
Hybrid App Development Task Flow With the Designer	7
Develop Hybrid Apps Using Third-party Web Frameworks	9
Develop MBO-based Hybrid Apps	9
Creating a Mobile Application Project	9
Developing a Mobile Business Object	10
Deploying a Mobile Application Project	11
MBO Examples	13
Generating JavaScript MBO Access API	23
Processing Responses From the Server	41
Error Handling	41
URL Parameters	42
Develop OData-based Hybrid Apps	43
Connect to an OData Source	43
Datajs OData Client Authentication in Hybrid Apps	45
Implementing Push	63
Enabling the Datajs Library on Windows Mobile	64
Hybrid Web Container and Hybrid App JavaScript APIs	64
MBO Access JavaScript API Samples	67

MediaCache Examples	71
Null Value Support	72
Calling the Hybrid Web Container	73
AttachmentViewer and Image Limitations	76
Package Hybrid Apps	76
Packaging Hybrid Apps Using the Packaging Tool	77
Packaging Hybrid Apps Manually	79
Deploying a Hybrid App Package with the Deploy Wizard	101
Develop a Hybrid App Using the Hybrid App Designer .	103
Deploy the Hybrid App Package to SAP Mobile Server	103
Generating Hybrid App Files and Deploying a Package	103
Hybrid App Patterns	104
Online Lookup	106
Server Notification	111
Cached Data	116
Hybrid App Package Customization	124
Customizing Generated Code	125
Adding Local Resources to a Hybrid App Project	125
Generated Hybrid App Files	126
Reference	133
Using Third-Party JavaScript Files	150
Repackaging Hybrid App Package Files	151
Common Customizations	151
Security	156
Credentials	156
Configuring the Hybrid App to Use Credentials .	160
Content Security on Devices	166
Localization and Internationalization	172
Localization Limitations	173
Localizing a Hybrid App Package	173

Hybrid App Package Internationalization	179
Internationalization on the Device	181
Test Hybrid App Packages	182
Testing Server-Initiated Hybrid App Packages ...	183
Launching a Server-initiated Hybrid App on the Device	184
Debugging Custom Code	184
Manage a Hybrid App Package	189
Registering or Reregistering Application Connections	189
Setting General Application Properties	191
Application ID and Template Guidelines	192
Enabling and Configuring the Notification Mailbox	193
Assigning and Unassigning a Hybrid App to an Application Connection	194
Activating the Hybrid App	194
Configuring Context Variables for Hybrid App Packages	195
Changing Hard Coded User Credentials	196
Adding a Certificate File to the Hybrid App Package	197
End to End Trace and Performance	197
Enabling the Performance Agent on the Device	198
Tracing Application Connections	198
Build a Customized Hybrid Web Container Using the Provided Source Code	201
Building the Android Hybrid Web Container Using the Provided Source Code	201
Building the Android Hybrid Web Container Outside of Eclipse	202
Building the BlackBerry Hybrid Web Container Using the Provided Source Code	202
Supplying a Signing Key	203
Building the iOS Hybrid Web Container Using the Provided Source Code	204

Building the Windows Mobile Hybrid Web Container Using the Provided Source Code	205
Install and Configure the Hybrid Web Container On the Device	207
Preparing Android Devices for the Hybrid Web Container	207
Installing the Hybrid Web Container on Android Devices	207
Configuring the Android Emulator	207
Preparing BlackBerry Devices for the Hybrid Web Container	210
Installing the Hybrid Web Container on BlackBerry Devices Over the Air	211
Enabling Hybrid Web Container Message Notification	211
Configuring the BlackBerry Simulator for Hybrid Web Containers	212
Preparing iOS Devices for the Hybrid Web Container	212
Installing the Hybrid Web Container on the iOS Device	213
Preparing Windows Mobile Devices for the Hybrid Web Container	214
Installing the Hybrid Web Container on Windows Mobile Devices	214
Installing Microsoft Synchronization Software ..	215
Installing the Hybrid Web Container on the Windows Mobile Emulator	216
Configure Connection Settings on the Device	218
Configuring Android Connection Settings	218
Configuring BlackBerry Connection Settings	219
Configuring iOS Connection Settings	220
Configuring Windows Mobile Connection Settings	221

Install and Test Certificates on Simulators and Devices	223
Installing X.509 Certificates on Windows Mobile Devices and Emulators	223
Installing X.509 Certificates on Android Devices and Emulators	224
Installing X.509 Certificates on BlackBerry Simulators and Devices	225
Installing X.509 Certificates on iOS Devices	226
Uninstall the Hybrid Web Container from the Device ..	231
Removing the Hybrid Web Container From the BlackBerry Device	231
Hybrid Web Container Customization	233
Adding a Custom Icon for the Hybrid App Package Using the Packaging Tool	233
Manually Adding a Custom Icon to the Manifest.xml File	234
Changing the Hybrid App Package Icon	235
Android Hybrid Web Container Customization	236
Android Customization Touch Points	236
Testing Android Hybrid Web Containers	269
BlackBerry Hybrid Web Container Customization	269
BlackBerry Customization Touch Points	270
iOS Hybrid Web Container Customization	298
iOS Customization Touch Points	299
Windows Mobile Hybrid Web Container Customization	312
Windows Mobile Customization Touch Points ...	313
Look and Feel Customization of the Windows Mobile Hybrid Web Container	314
Default Behavior Customization of the Windows Mobile Hybrid Web Container	317
Packaging a CAB File	321
Prepackaged Hybrid Apps	322

Including a Prepackaged Hybrid App in the Android Hybrid Web Container	322
Including a Prepackaged Hybrid App in the BlackBerry Hybrid Web Container	323
Including a Prepackaged Hybrid App in the iOS Hybrid Web Container	324
Including a Prepackaged Hybrid App in the Windows Mobile Hybrid Web Container	325
Adding Native Device Functionality to the Hybrid Web Container	328
Supported JavaScript PhoneGap APIs	328
Implementing PhoneGap	338
PhoneGap Custom Plug-ins	339
Removing PhoneGap From the Hybrid Web Container	350
Initializing the PhoneGap Library for the Windows Mobile Hybrid Web Container	352
PhoneGap Library Downgrade	352
Hybrid App Configuration for Data Change Notification	357
Extending Data Change Notification to Hybrid Apps	357
Non HTTP Authentication Hybrid App DCN Request	359
Sending Hybrid App DCN to Users Regardless of Individual Security Configurations	359
Hybrid App DCN Request Response	360
Hybrid App DCN Design Approach and Sample Code	361
Comparing Hybrid App DCN With and Without Payload	361
Sample Java Function for Generating Hybrid App DCN	364
Index	367

Introduction to Developer Guide for Hybrid Apps

This developer guide provides information about using SAP® Mobile Platform features to create Hybrid App packages. The audience is Hybrid App developers.

This guide describes requirements for developing a Hybrid App package, how to generate Hybrid App package files, and how to deploy the Hybrid App to the server. It also provides information about how to customize the provided source code for Hybrid Web Containers.

Companion guides include:

- *SAP Mobile WorkSpace - Mobile Business Object Development*
- *SAP Mobile WorkSpace - Hybrid App Package Development*
- *System Administration*
- *SAP Control Center for SAP Mobile Platform*
- *Tutorial: Hybrid App Package Development*
- *Troubleshooting*
- *Mobile Application Life Cycle*
- *Developer Guide: Migrating to SAP Mobile SDK*

Documentation Roadmap for SAP Mobile Platform

SAP® Mobile Platform documents are available for administrative and mobile development user roles. Some administrative documents are also used in the development and test environment; some documents are used by all users.

See *Documentation Roadmap* in *Fundamentals* for document descriptions by user role.

Check the Product Documentation Web site regularly for updates: <http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories>, then navigate to the most current version.

Introduction to Developing Hybrid Apps With SAP Mobile Platform

A Hybrid App includes both business logic (the data itself and associated metadata that defines data flow and availability), and device-resident presentation and logic.

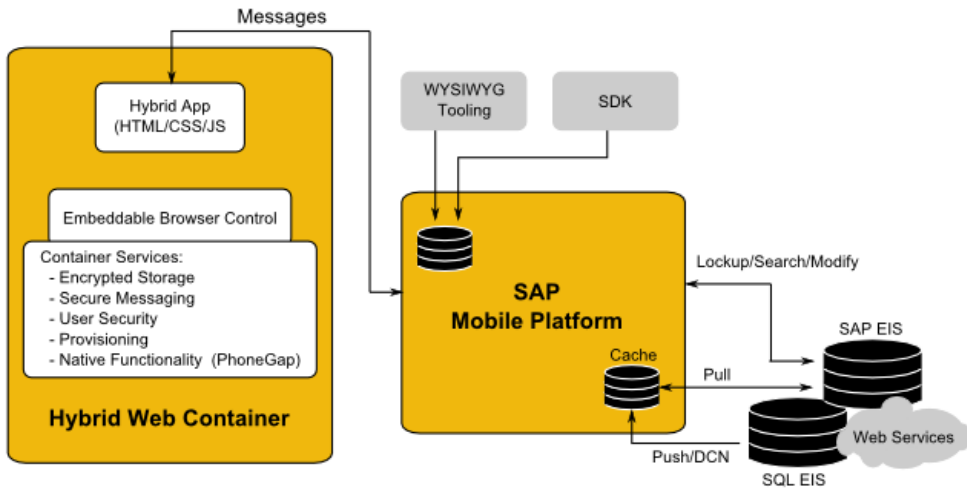
You can develop Hybrid Apps using third-party Web frameworks, enabling you to access gateway datasources through the Hybrid Web Container.

SAP Mobile Platform, development tools enable both aspects of Hybrid App development:

- The data aspects of the Hybrid App are called mobile business objects (MBO), and “MBO development” refers to defining object data models with back-end enterprise information system (EIS) connections, attributes, operations, and relationships. Hybrid Apps can reference one or more MBOs and can include load parameters, personalization, and error handling.
- Once you have developed MBOs and deployed them to SAP Mobile Server, develop device-resident presentation and logic for your Hybrid App. See *SAP Mobile WorkSpace - Mobile Business Object Development* for procedures and information about creating and deploying MBOs.
- A second data option is to access OData sources from your Hybrid Apps with the Datajs library.
- OData sources and MBOs can be used together in a Hybrid App.

Hybrid Web Container Architecture

The Hybrid Web Container is the runtime on the device within which Hybrid Apps are executed.



Hybrid Web Container Customization

A Hybrid Web Container is a native application designed to process generic function calls from a Hybrid App. The Hybrid Web Container embeds a browser control supplied by the device OS, which allows you to build applications with simplicity of Web development but utilize the power of native device services. By using the Hybrid Web Container for each device type supported in a business mobility environment, you can create a single HTML5 application that performs advanced, device specific operations on all the different devices.

Hybrid App Development

The Hybrid Web Container supports workflow type applications, which are applications that participate in a lifecycle flow involving special notifications (modified in the Transform Queue), application flow, and finally submission of form data to matching server components (through the Response Queue).

The Hybrid Web Container also supports applications that do not participate in a workflow type process. In other words, applications that are not triggered by notifications (no Transform Queue), and that do not submit asynchronous “submit” responses through queuing (no Response Queue). These applications do not communicate with the server for data access, but use the messaging channel for deployment, provisioning, and application life cycle management.

Write Hybrid Apps in standards-based HTML5, JavaScript (the standard scripting language used to create Web applications), and Cascading Style Sheets (CSS). These are technologies familiar to web developers. This enables Web developers to incorporate open source frameworks and also select their preferred development environment, for example, Sencha and JQuery Mobile.

Hybrid App Designer

The Hybrid App Designer uses the Hybrid Web Container as the runtime for Hybrid App packages. The Hybrid App Designer included with SAP Mobile Platform is a tool that helps you design the user interface and test the flow of the business process for an application. Using the Hybrid App Designer allows you to develop application screens that can call on the create, update, and delete operations, as well as object queries, of a mobile business object.

Hybrid App package files are generated using the Hybrid App Package generation wizard in the Hybrid App designer. The generated Hybrid App package contains files that reference a mobile business object (MBO) package, an MBO in that package, and the operation or object query to call along with a mapping of which key values map to parameter values. The generated Hybrid App package's output is translated to HTML\CSS\JavaScript. The logic for accessing the data and navigating between screens is exposed as a JavaScript API.

The Hybrid App packages generated by the Hybrid App designer are not proprietary, they are identical to what would need to be produced when using other tools and Web application frameworks. Hybrid App Designer-generated packages use jQuery Mobile as their primary Web application framework on most platforms.

Deploy Hybrid App packages to SAP Mobile Server and assign to users using the Hybrid App Designer in Eclipse.

Generated Customization Files

The Hybrid Web Container uses HTML, JavaScript, and CSS Web technologies, which allow you to customize the generated files with JavaScript code.

- **HTML** – the generated files depend on the device platform. You can open these files with a third-party Web-development tool and modify them, but they are overwritten if generated from the Hybrid App deployment tool. The Hybrid App Designer also includes a **HTMLView** user interface element that can be placed on a screen, and in which custom HTML code can be inserted, which will be published in-line when the file is re-generated.
- **JavaScript** – the JavaScript API exposes customization points for navigation events, and allows access to data-access functions for requests and cached values. Customization of the HTML page should be executed using the embedded jQuery in these customization points. For example, execute jQuery logic to modify the toolbar in `customBeforeHybridAppLoad()`. You can add additional custom JavaScript files to the Hybrid App package in the Eclipse Workspace.

Note: In prior releases, JavaScript files for customization were automatically included in the generated Hybrid App HTML files. The JavaScript files are still added to the generated package, but no longer referenced in the HTML.

- **CSS** – the Hybrid Web Container uses a third-party CSS library, which enables you to modify the look-and-feel of the HTML page. The jQueryMobile CSS file is embedded as the default look-and-feel, which allows you to select from the variety of themes within the jQueryMobile framework, or use your own CSS rules for skinning pages and screen

elements. These can be device operating system-specific. You can also leverage existing CSS style rules from your own organization's Web standards.

The generated files are documented in the *Reference* section of this guide.

Management

You can deploy Hybrid App packages in Eclipse and manage them through the SAP Control Center console. No device interaction is required from the administrator. Once a Hybrid App package is deployed into an existing installation, the administrator can configure the Hybrid App package and assign it to any active user in the system.

Offline Capabilities

Server-initiated notifications extract data from the backend and SAP Mobile Platform sends them to the client device. The client device does not need to be online at the time the notification is sent—the message is received as soon as the client device comes online. Submit actions on the client can also be sent while the device is offline. They will be sent to the server as soon as the device comes online. These notifications are made available offline for processing once they are delivered to the device.

Online Request actions only work when the device is online. The results of object queries run by these types of actions can be cached on the client so that the next time the same query is invoked with the same parameters it is able to get those results from the client-side cache without needing to go to the server. This is achieved by specifying a non-zero cache timeout for the action.

You can also store data locally (when the device is offline) using the `SUPStorage` JavaScript API.

Hybrid App Development Task Flow

This task flow describes task flows for the different Hybrid App development options.

Hybrid App Development Task Flow Using Third-Party Web Frameworks and MBOs

This describes the basic steps for developing Hybrid Apps that access MBO operations and object queries using a third-party tool, or by hand.

1. Define the data you want to use from your backend system and to expose through your Hybrid App, and the methods and operations to perform.
2. *Create a Mobile Application project* on page 9.
3. *Develop the Mobile Business Objects* on page 10.
4. *Deploy the Mobile Application Project* on page 9.
5. *Generate the JavaScript API* on page 23.

6. *Package the Hybrid App files.* on page 76
7. *In SAP Control Center, deploy the Hybrid App package to SAP Mobile Server.* on page 101

Hybrid App Development Task Flow With the Designer

Developing a Hybrid App includes these basic tasks.

1. Open or import a mobile application project with predefined mobile business objects (MBOs).
2. Deploy the Mobile Application Project:
 - b. On the Target Server page, select the server and connect to it.
 - c. On the Server Connection Mapping page, map the database connection profile to the server.
3. Create the application connection in SAP Control Center.

Note: This step is normally performed by the system administrator.

4. Use the Hybrid App Designer to create a new Hybrid App.

Note: Optionally, you can create a Hybrid App manually, however, using the Hybrid App Designer, automates many tasks and provides integration across different device platforms.

5. Use the Hybrid App Designer to generate screens by dragging and dropping MBOs and MBO operations from WorkSpace Navigator to the Flow Design page.
6. Create, delete, and edit screens, controls, menus, screen navigations, and so on.
7. Generate the Hybrid App files.
8. (Optional) Customize the generated `Custom.js` file.
9. (Optional) If you customize the Hybrid App files, re-generate and repackage the files.
10. Deploy the Hybrid App package to SAP Mobile Server.
11. Install and configure the Hybrid Web Container on the device or simulator.
12. In SAP Control Center, assign the Hybrid App to the device user.
13. On the device or simulator, run, test and debug the Hybrid App.

Note: See *SAP Mobile WorkSpace - Mobile Business Object Development* for procedures and information about creating and deploying MBOs.

Identify a Business Process for a Hybrid App That Uses a Lifecycle Flow

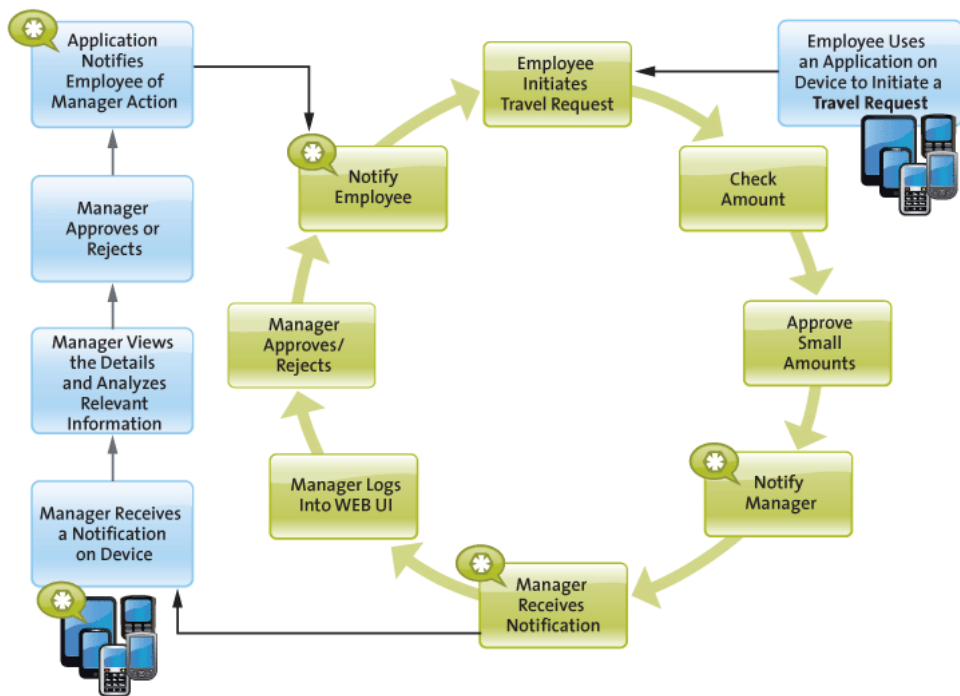
The first step in developing a Hybrid App that participates in a lifecycle flow involving special e-mail messages (modified in the Transform Queue), application flow, and finally submission of form data to matching server components (through the Response Queue) is identifying whether a Hybrid App can implement a decision point in a particular business process.

Hybrid Apps enable a decision step or triggering of a business process, essentially mobilizing a small decision window in a business process. While some business processes require a thick application with business logic and access to reference data, some others do not. Sometimes a

business process can be made mobile simply by providing the ability to capture a single "Yes" or "No" from a user, or by providing the ability to send data in structured form into the existing backend systems.

A typical Hybrid App allows creating, updating or deleting of data in a backend data source (EIS), either directly or through the SAP Mobile Server, and retrieving that data, then displaying that information in a decision step. A more complex Hybrid App could involve an application that uses online request menu items to invoke various create, update, or delete operations and/or object queries all in the same flow.

An example of a business process that would be a suitable Hybrid App would be the ability of an employee to use a mobile device to submit an expense report while out of office, or to report on their project activities, or to make a request for travel.



Develop Hybrid Apps Using Third-party Web Frameworks

Developing Hybrid Apps this way allows you to use a greater variety of application designs, from using different HTML formatting to using different Web application frameworks, and beyond.

Note: When writing your own HTML and JavaScript to create a Hybrid App package manually, there is one absolute requirement—you must implement the following JavaScript function:

```
function processDataMessage (incomingWorkflowMessage)
```

The Hybrid Web Container needs to call this function when online request processing is complete. The incoming message is an XML-formatted string.

Develop MBO-based Hybrid Apps

Develop Hybrid Apps using mobile business objects (MBO) to define object data models with back-end enterprise information system (EIS) connections, attributes, operations, and relationships.

A project in SAP Mobile WorkSpace must contain the MBOs to use in your application. See *SAP Mobile WorkSpace - Mobile Business Object Development*.

The JavaScript APIs in the Mobile SDK are located in `<SMP_HOME>|UnwiredPlatform|MobileSDK<version>|HybridApp|API`. It is split into two categories:

- **Container** – these APIs are fundamentally independent of the UI framework you choose to use (if any). There is no reference to screens. These APIs are considered mandatory when building your Hybrid App.
- **AppFramework** – these APIs are an optional add-on to the Container APIs that give you functionality to navigate between screens, represent the messages sent to and from the server in developer-friendly form, and bind the UI to and from those messages automatically. These APIs do make some assumptions about how your UI is constructed/manipulated, and those assumptions are not necessarily true for all UI frameworks, Sencha among them.

Creating a Mobile Application Project

A mobile application project is the container for the mobile business objects that forms the business logic of mobile applications.

You must create a mobile application project before you can create mobile business objects. See *Eclipse Basics* for information about projects.

1. Select **File > New > Mobile Application Project** from the main menu bar.
2. Enter a:
 - Name
 - Location (if other than the default).

3. Click **Finish**.

The Mobile Application Project is created and an empty Mobile Application Diagram opens.

4. (Optional) Modify the Mobile Application Project configuration properties by right-clicking the project in WorkSpace Navigator, selecting **Properties**, and selecting **Mobile Application Project Configuration**. When modifying the configuration properties, keep in mind that:

- The default application ID and Display name are the same as the project. The description is "Default application ID".
- Follow these guidelines when changing application ID, application name, display name, and description:
 - **ID** – less than 64 bytes, begin with an alphabetic character, followed by alphanumeric characters, a dot, or underscores, and not contain consecutive dots or underscores.
 - **Display name** – string, length less than 80 bytes.
 - **Description** – string, length less than 255 bytes.

All added applications must have a name (display name and description can be empty), but are assigned a name at runtime when the application is created.

Developing a Mobile Business Object

You can define attributes and operations of a mobile business object (MBO) without immediately binding them to a data source, define them from and bind them to a data source, or create an MBO that does not bind to a data source (local business object, or uses only DCN as the refresh mechanism).

Prerequisites

Before developing MBOs, understand the key concepts and principals described in *Understanding Fundamental Mobile Development Concepts*. Also, see the companion guide, *Mobile Data Models: Using Mobile Business Objects*, for a deeper understanding of how to build an efficient MBO model.

Task

The attributes and operations that define an MBO must be bound to a data source at some point in the development process, unless it is a local business object, or the MBOs data is to be loaded only through Data Change Notification (DCN). If you already have a connection to the data source through a connection profile, you can quickly generate attribute and operation bindings based on the data source. However, if you do not have access to the required data

source, you define the MBO, but bind your operations and attributes to the data source at a later point. The difference between the two development approaches is when you create and bind the attributes and operations:

- Create an MBO and bind to a data source immediately – includes two methods:
 1. Drag and drop the data source onto the Mobile Application Diagram, which launches the appropriate wizards and automatically creates bindings based on the selected data source.
 2. Create an MBO and its operations and attributes using the Mobile Application Diagram and palette that launches a set of wizards and allows you to bind them directly to a data source.
- Create an MBO and defer data source binding – create an MBO and its operations and attributes using the Mobile Application Diagram and palette that launches a set of wizards and allows you to bind the MBO to a data source at a later time. After you define the data source, you bind the MBO to it from the Properties view.
- Create an MBO using a DCN cache group policy without data source binding – when an MBO's CDB data is to be filled only through DCN, a data source binding is not necessary. In these cases, the MBO must reside in a cache group that uses the DCN policy.
- Create a local business object – create a local business object by clicking the local business object icon in the palette then click the object diagram. Local business objects can only run on the client and cannot be synchronized. It can contain attributes and operations that run on the device. For example, the local business object could be combined with other MBOs, where the local business object runs an object query against results returned by other MBOs.

Deploying a Mobile Application Project

Deploy a Mobile Application project directly to an SAP Mobile Server, and optionally create a reusable deployment profile.

To avoid errors or inconsistent behavior, client applications must be regenerated whenever a package has been redeployed. Restarting the client application is not sufficient to reset the client for a package that has been redeployed.

1. Right-click the Mobile Application project and select **Deploy Project**.

Alternatively, you can launch the deployment wizard, which automatically sets the SAP Mobile Server portion of the wizard, by dragging a Mobile Application project folder from Workspace Navigator and dropping it on the SAP Mobile Server in Enterprise Explorer to which you are deploying.

Note: As an option, you can press F9 when your cursor is in the Mobile Application Diagram to launch the Deployment wizard for the corresponding project. If a deployment profile exists for the project, F9 performs quick deployment of the project according to the profile.

2. Select a deployment mode (**Update**, **Replace**, or **Verify**), target version, Package name, and click **Next**.
3. Select the MBOs from each Synchronization Group to be deployed and click **Next**.

Note: If any selected MBOs contain errors, the **Next** and **Finish** buttons are disabled.

4. Create or add required JAR files for MBOs that use Resultset Filters or Custom Result Checkers and click **Next**.
5. Select a target server, click **Connect**, and select a Domain and Security Configuration for the deployment package and click **Next**. (Optional) If no SAP Mobile Server connection exists, click **Create** and define a connection profile for one to which you can connect and deploy the deployment package.
6. Deploy applications to SAP Mobile Server – select the applications to deploy to SAP Mobile Server. A unique Application ID identifies the application and uses the project name by default.

SAP Mobile WorkSpace lists not only local applications defined through the mobile application project's context menu **Properties > Mobile Application Project Configuration**, but all applications already assigned to the selected domain of the target server (available applications), whether those existing applications contain this current mobile application project or not. SAP Mobile WorkSpace validates the projects for:

- If the local and server applications are the same, but the display name or description differ, they display in the target applications list, but a validation error appears because the assigned application ID must be unique.
 - When deploying the project/package with "Replace" mode, if the project/package already exists in an available application that exists on the server, but that application is not selected as the target application, a warning indicates that the server will remove the project/package from the existing application.
 - If a local application is added to the target applications list, and a server application with the same ID but different display name/description is not assigned, a warning indicates that you can modify the display name/description of the existing sever application with that of the local application.
7. Map connection profile to server connections – you must map design-time connection profiles to server-side (runtime) enterprise information system (EIS) data sources referenced by the MBOs in the project. Deployment fails if the EIS data sources are not running and available to connect to. To map the connection profile to a server connection, select the connection profile from the list of available connection profiles then select the corresponding server connection to which it maps, or select **<New Server Connection...>** to create a new server connection.

Contact the system administrator in cases where your development environment permits access to systems that the SAP Mobile Server prohibits.

Note: You can also modify server connection properties (Web service connections only).

8. If a logical role is defined in your MBO, map logical roles to physical roles. If there are no logical roles defined, this page is skipped. Click **Next**.
9. (Optional) Specify the name and location for the new deployment profile. This is useful for troubleshooting MBO and deployment errors.
 - Save the deployment settings as a deployment profile – if you do not save your settings to a deployment profile, they are lost when you exit the Deploy wizard.
 - Enter or select the parent folder – by default, Deployment is the folder in which the deployment profile is saved.
 - File name – the name of this deployment profile. The deployment profile is assigned a `.deploy` extension.
10. Click **Finish** to deploy the project to the SAP Mobile Server's Packages folder.

MBO Examples

This section shows examples of how to implement different patterns and functionality. These are examples only. You must modify the procedures based on the actual MBOs, object queries, and parameters you are using.

Implementing Online Lookup for Hybrid Apps

In this example, online lookup provides direct interaction between the data requester (client) and the enterprise information system (EIS), supplying real-time EIS data rather than cached data.

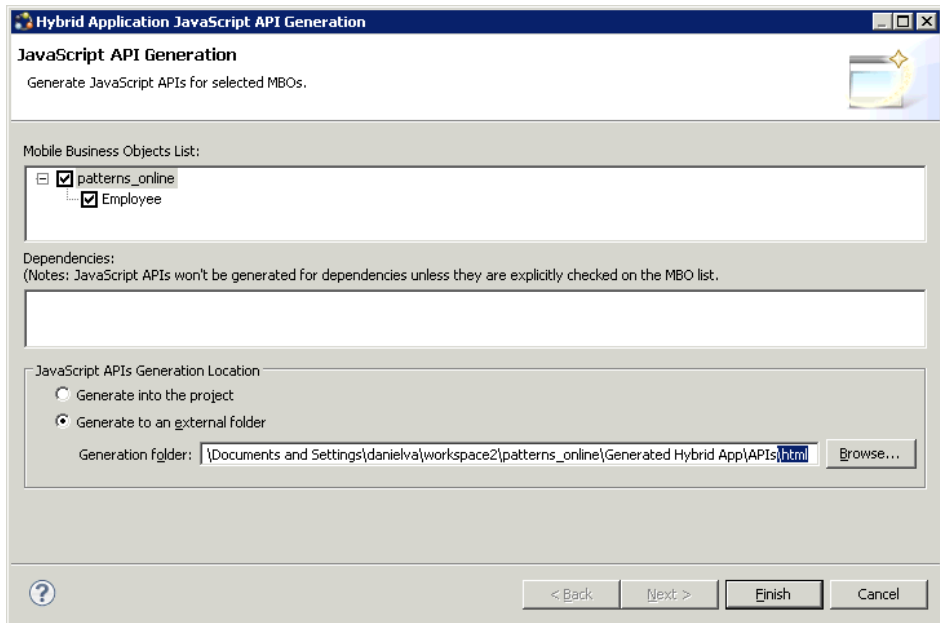
Prerequisites

Complete the procedure in *Defining Load Arguments from Mapped Propagate to Attributes* on page 107 so that you have an MBO with the required attributes.

Task

This section describes how to invoke the Employee's `findByParameter` method.

1. Right-click on the mobile application project and choose **Generate Hybrid App API**.
2. Select the Employee MBO, choose **Generate to an external folder**, and add `\html` to end of the folder name.



3. Right-click on the generated **html** folder and select **New > Other > General > File**.
4. Enter `online.html` for the file name.
5. Open the `online.html` file and add this code:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <meta name="HandheldFriendly" content="True" />
    <meta http-equiv="PRAGMA" content="NO-CACHE" />
    <meta name="viewport" content="initial-scale =
1.0 ,maximum-scale = 1.0" />
    <script src="js/PlatformIdentification.js"></script>
    <script src="js/hwc-api.js"></script>
    <script src="js/hwc-comms.js"></script>
    <script src="js/hwc-utils.js"></script>
    <script src="js/WorkflowMessage.js"></script>
    <script src="js/HybridApp.js"></script>

    <script>
      function findEmp() {
        var deptID = document.getElementById("deptID").value;
        emp = new Employee();
        emp.deptIdLP = deptID;
        employee_findByParameter(emp,
"supusername=supAdmin&suppassword=s3pAdmin", "onError");
      }

      function onError(e) {
        alert("An error occurred");
      }
    </script>
  </head>
</html>
```

```
    }  
  
    hwc.processDataMessage = function  
(incomingDataMessageValue) {  
        if (incomingDataMessageValue.indexOf("<M>") != 0) {  
            alert("An error occurred! " +  
incomingDataMessageValue);  
        }  
        var workflowMessage = new  
WorkflowMessage(incomingDataMessageValue);  
        var values = workflowMessage.getValues();  
        var empList = values.getData("Employee");  
        var firstEmp = empList.value[0];  
        var firstName =  
firstEmp.getData("Employee_emp_fname_attribKey").value;  
        alert("The name of the first employee is " + firstName);  
    }  
}</script>  
</head>  
<body>  
<form>Dept Id: <input type="text" value="100" id="deptID"/></  
form><br>  
<button id="findEmpsButton" onclick="findEmp()">Find</  
button>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
<button id="closeWorkflow" onclick="hwc.close()">Close</  
button>  
</body>  
</html>
```

Five of the included files are from <SMP_HOME>\MobileSDK22\HybridAp\API folder. The file named HybridApp.js is generated based on the operations and object queries of the MBOs selected in the Generate Hybrid App API wizard. When the Find button is clicked, the department ID is retrieved and set on the employee object, which is an input parameter to the method named `employee_findByDeptId` in `HybridApp.js`. Once the result returns from SAP Mobile Server, it is passed into the method `processDataMessage` where the first employee's name is shown.

6. Navigate to `SMP_HOME\MobileSDK22\HybridApp\PackagingTool` and double-click the `packagingtool.bat` file if you are using a 32-bit JDK, or `packagingtool64.bat` if you are using a 64-bit JDK.
7. Click **Browse** to enter the file path for your project and click **OK**.
8. Select **Project > New**.
9. Fill in `Patterns_Online` and the location of where the generated files currently exist (the same location specified as the `Generation` folder above) for the Project name and Web root directory.
10. Fill in the MBO package name and version to match the deployed package.
11. Specify the files to include in the Hybrid App for each supported platform.
Only the selected files appear in the `manifest.xml` file.

12. Click **Generate** to generate a deployable Hybrid App package.
13. Deploy and assign the Hybrid App package using SAP Control Center.

Implementing Server Notification for Hybrid App Clients

Configure matching rules for MBO-related data on SAP Mobile Server.

Prerequisites

Complete the procedure in *Defining the Mobile Business Object* on page 111 so that you have an MBO with the required attributes.

Task

Any data changes matching these rules trigger a notification from SAP Mobile Server to the Hybrid App client. This section describes how to write HTML, JavaScript, and modify the `WorkflowClient.xml` to display the results of a server notification.

1. Right-click on the mobile application project and choose **Generate Hybrid App API**.
2. Select the Sales MBO, choose **Generate to an external folder**, and add `\html` to end of the folder name.
3. Right-click on the generated **html** folder and select **New > Other > General > File**.
4. Enter `notification.html` for the file name.
5. Open the `notification.html` file and add this code:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <meta name="HandheldFriendly" content="True" />
    <meta http-equiv="PRAGMA" content="NO-CACHE" />
    <meta name="viewport" content="initial-scale =
1.0 ,maximum-scale = 1.0" />
    <script src="js/PlatformIdentification.js"></script>
    <script src="js/hwc-api.js"></script>
    <script src="js/hwc-comms.js"></script>
    <script src="js/hwc-utils.js"></script>
    <script src="js/WorkflowMessage.js"></script>
    <script src="js/HybridApp.js"></script>
    <script>
      hwc.processDataMessage = function
(incomingDataMessageValue) {
        if (incomingDataMessageValue.indexOf("<M>") != 0) {
          alert("An error occurred! " +
incomingDataMessageValue);
        }
        var workflowMessage = new
WorkflowMessage(incomingDataMessageValue);
        var values = workflowMessage.getValues();
        var salesOrderList = values.getData("Sales_order");
        var salesOrder = salesOrderList.value[0];
        var salesOrderId =
```



```

salesOrder.getData("Sales_order_id_attribKey").value;
    var custId =
salesOrder.getData("Sales_order_cust_id_attribKey").value;
    alert("The customer id for sales order " + salesOrderId
+ " is " + custId);
    }
</script>
</head>
<body onload="hwc.onHybridAppLoad_CONT()">
    <h3>Server Notification Sample</h3>
    <button id="closeHybridApp" onclick="hwc.close()">Close</
button>
</body>
</html>

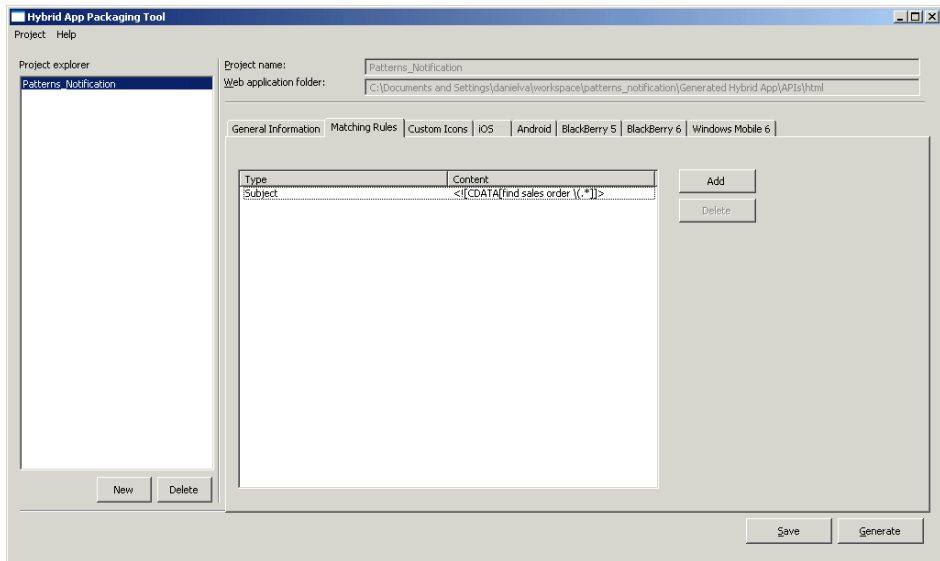
```

Five of the included files are from the <SMP_HOME>\MobileSDK22\HybridApp\API folder. The file named HybridApp.js is generated based on the operations and object queries of the MBOs selected in the Generate Hybrid App API wizard. In the onload event, the method hwc.onHybridAppLoad_CONT() is called. For server-initiated applications this returns the data message associated with this instance of the server-initiated application as a parameter to hwc.processDataMessage(). In processDataMessage, some of the data is extracted from the application message and displayed.

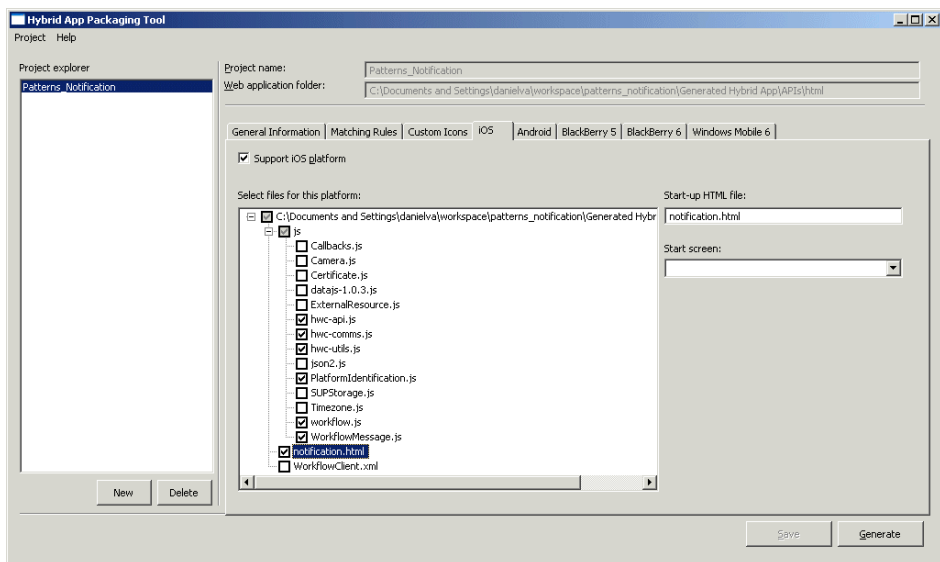
6. Navigate to SMP_HOME\MobileSDK22\HybridApp\PackagingTool and double-click the packagingtool.bat file.
7. Click **Browse** to enter the file path for your project and click **OK**.
8. Select **Project > New**.
9. Fill in Patterns_Notification and the location of where the generated files currently exist (the same location specified as the Generation folder above) for the Project name and Web root directory.
10. Fill in the MBO package name and version to match the deployed package.
11. Specify a matching rule for the subject:


```
<![CDATA[find sales order \(.*)]]>
```

Develop Hybrid Apps Using Third-party Web Frameworks



12. Specify the files to include in the Hybrid App for each supported platform.
Only the selected files will appear in the `manifest.xml` file.



13. Open the generated `WorkflowClient.xml` file and update the Notifications section:

```
<Notifications>
  <Notification type="onEmailTriggered" targetscreen="Salesorder"
    asyncRequestErrorScreen="" errorNotificationSubjectLine=""
    errorNotificationFromLine="" asyncRequestErrorlogs=""
    asyncRequestErrorLogMessage=""
```

```

asyncRequestErrorLogMessageAsList="">
  <Transformation>
    <Rule type="regex-extract" source="subject"
workflowKey="order_id" workflowType="number" beforeMatch="find
sales order \" afterMatch="\" format="\"/>
  </Transformation>
  <Methods>
    <Method name="findByParameter" type="search"
mbo="Sales_order" package="patterns_notification:1.0">
      <InputBinding opname="findByParameter" optype="none">
        <Value sourceType="Key" workflowKey="order_id"
contextVariable="" paramName="order_id" attribName="id"
mboType="int" convertToLocalTime="false"/>
      </InputBinding>
      <OutputBinding generateOld="true">
        <Mapping workflowKey="Sales_order" workflowType="list"
mboType="list">
          <Mapping workflowKey="Sales_order_id_attribKey"
workflowType="number" attribName="id" mboType="int"/>
          <Mapping workflowKey="Sales_order_cust_id_attribKey"
workflowType="number" attribName="cust_id" mboType="int"/>
          <Mapping workflowKey="Sales_order_order_date_attribKey"
workflowType="date" attribName="order_date" mboType="date"/>
          <Mapping
workflowKey="Sales_order_fin_code_id_attribKey"
workflowType="text" attribName="fin_code_id" mboType="string"/>
          <Mapping workflowKey="Sales_order_region_attribKey"
workflowType="text" attribName="region" mboType="string"/>
        </Mapping>
      </OutputBinding>
    </Method>
  </Methods>
</Notification>
</Notifications>

```

14. Save and close the file.

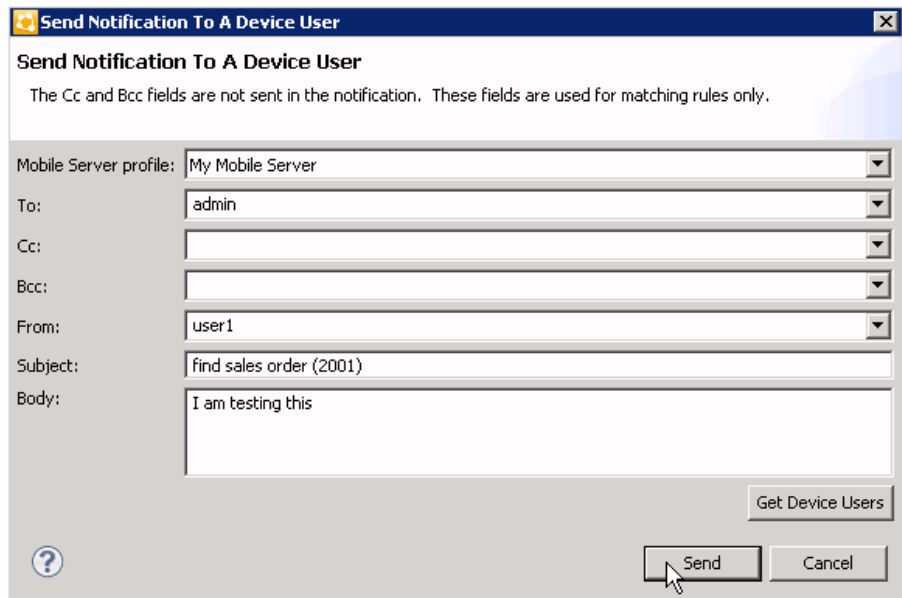
15. In the Hybrid App Packaging Tool, click **Generate** to create a deployable package.

16. Login in to SAP Control Center to deploy and assign the Hybrid App package.

17. Send a notification to the device.

Typically this is triggered by a database trigger or by the EIS sending a DCN. You can also use the Send a Notification wizard in the Hybrid App designer.

- a) In the Hybrid App designer, click **Flow Design**.
- b) Right-click in the Flow Design page and select **Send a notification**.



Implementing the Cached Data Pattern for MBO-based Hybrid Apps

For access to cached data, define a menu action and bind it to the `findByDeptId` object query.

Prerequisites

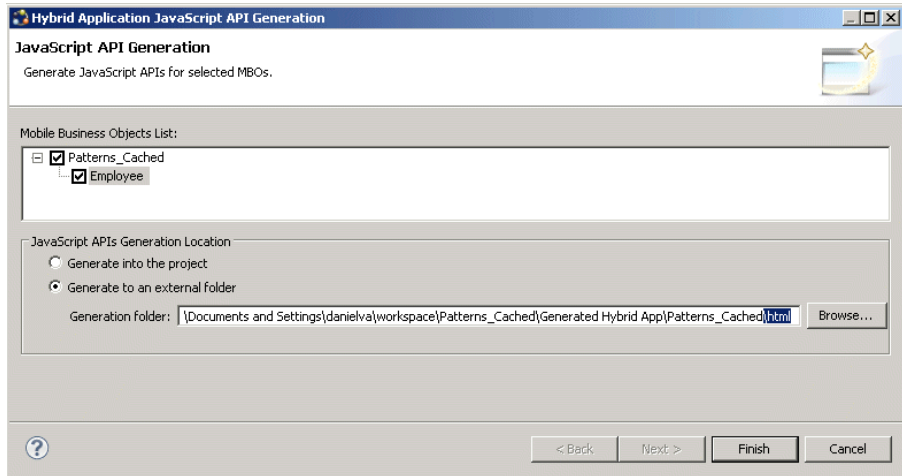
Complete the procedure in *Defining the Mobile Business Object* on page 117 so that you have an MBO with the required attributes.

Task

Using cached data is efficient when access to cached data is sufficient to meet business needs. For example, it may be sufficient to refresh the cache once a day for noncritical MBO data that changes infrequently.

1. Generate the Hybrid App API:

- a) Right-click the mobile application project and choose **Generate Hybrid App API**.
- b) In the tree view, select the **Employee** MBO, which contains the `findByDeptId` object query.
- c) Choose **Generate to an external folder** and add `"\html"` to end of the folder name.



By default, the wizard creates a Generated Hybrid App folder under the project and a sub folder named APIs.

d) Click **Finish**.

2. Right-click the `html` folder and choose **New > Other > General > File**, and enter `cached.html` for the file name.
3. Copy and paste the following contents into the `cached.html` file:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <meta name="HandheldFriendly" content="True" />
    <meta http-equiv="PRAGMA" content="NO-CACHE" />
    <meta name="viewport" content="initial-scale =
1.0 ,maximum-scale = 1.0" />
    <script src="js/PlatformIdentification.js"></script>
    <script src="js/hwc-comms.js"></script>
    <script src="js/hwc-utils.js"></script>
    <script src="js/WorkflowMessage.js"></script>
    <script src="js/HybridApp.js"></script>

    <script>
      function findByDept() {
        var deptID = document.getElementById("deptID").value;
        emp = new Employee();
        emp.deptIDLp = deptID;
        employee_findByDeptId(emp,
"supusername=supAdmin&suppassword=s3pAdmin", "onError");
      }

      function onError(e) {
        alert("An error occurred");
      }

      hwc.processDataMessage = function
```

```
(incomingDataMessageValue) {  
    var workflowMessage = new  
WorkflowMessage(incomingDataMessageValue);  
    var values = workflowMessage.getValues();  
    var empList = values.getData("Employee");  
    var firstEmp = empList.value[0];  
    var firstName =  
firstEmp.getData("Employee_emp_fname_attribKey").value;  
    alert("The name of the first employee is " + firstName);  
}  
</script>  
</head>  
<body>  
<form>Dept Id: <input type="text" value="100" id="deptID"/></  
form><br>  
    <button id="findByDeptButton" onclick="findByDept()">Find</  
button>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
    <button id="closeWorkflow" onclick="hwc.close()">Close</  
button>  
</body>  
</html>
```

4. Open the packaging tool to create a deployable ZIP file of the Hybrid App by double-clicking on `packagingTool.bat`, which is located in `<SMP_HOME>\MobileSDK22\HybridApp\PackagingTool\`.
5. Enter a location for the generated ZIP file, for example, `c:\patterns`.
6. Choose **Project > New**.
7. Fill in `Patterns_Cached` and the location where the generated files currently exist for the Project name and Web root directory.
8. Fill in the MBO package name and version to match the deployed package.
9. Specify the files to include in the Hybrid App for each supported platform.
Only the selected files appear in the `manifest.xml` file.
10. Click **Deploy** to create a deployable package.
11. Log in to SAP Control Center to deploy the Hybrid App package and assign the Hybrid App to an application connection.
12. Review the contents of the `cached.html` file.

The first four included files are from the Mobile SDK located in the <SMP_HOME>\MobileSDK22\HybridAp\API folder. The last file is generated based on the operations and object queries of the MBOs selected when you generated the Hybrid App API.

When you click the **Find** button, the department ID is retrieved and set on the employee object, which is an input parameter to the method named `employee_findByDeptId` in `HybridApp.js`. Once the result returns from the SUP server, it is passed into the method `processDataMessage`, where the first employee's name is shown.

Generating JavaScript MBO Access API

Generate JavaScript API for MBOs to use in your custom code.

The generated API automatically includes, and utilizes, the Container APIs, along with the message manipulation APIs from the AppFramework portion of the Mobile SDK. The wizard also generates the `WorkflowClient.xml` file, which is required to support those functions.

Note: The generated `WorkflowClient.xml` file does not include a completed Notification, so if you want a server-initiated Hybrid App, you must do this by hand.

1. In WorkSpace Navigator, right-click the Mobile Application project for which to generate the JavaScript, and choose **Generate Hybrid App API**.
2. In the tree view, select the MBOs for which to generate the JavaScript.
3. Accept the default location for the files, or specify the location for the generated files and click **Finish**.

By default, the wizard creates a `Generated Hybrid App` folder under the project, and a subfolder named `APIs`.

Generated Hybrid App Files

Examine the generated files.

- `WorkflowClient.xml` – this file establishes the mapping between the XML messages and JSON calls to and from the SAP Mobile Server server.

Note: The generated `WorkflowClient.xml` does not include a completed notification, so if you want the Hybrid App to be server-initiated, you must write the Notification section. *See [Creating Notifications to Make the Hybrid App Server-Initiated](#).*

- `WorkflowMessage.js` – defines some convenient functions for accessing incoming application messages.
- `Workflow.js` – contains functions map to the MBO's operations and object queries. The contents depend on the MBOs you select when you run the wizard, since the wizard generates only the JavaScript API functions for the selected MBOs.
- These files are static, container related, commonly used JavaScript libraries and are copied from the `<SMP_HOME>\UnwiredPlatform\Mobile SDK HybridApp\API\Container` folder.
 - `Callbacks.js` – Hybrid Web Container callback functions.
 - `SUPStorage.js` – client side key/value storage.
 - `hwc-comms.js` – native container functions that are invoked from the custom code.
 - `Camera.js` – functions for accessing the device's native camera functionality.
 - `Timezone.js` – utility functions for getting the local time.

- `hwc-utils.js` – native Hybrid Web Container utility functions.
- `Certificate.js` – functions for processing certificates.
- `json2.js` – functions for processing JSON data.
- `ExternalResource.js` – functions for accessing external resources.
- `datajs-1.0.2.js` – functions for communicating through an OData protocol.
- `PlatformIdentification.js` – utility functions for checking the platform.
- `hwc-api.js` – native Hybrid Web Container functions that allow users access to Hybrid App metadata and notifications from the custom code.

HybridApp.js

In the `HybridApp.js` file, helper JavaScript structures are generated for the selected MBOs, and for the MBOs that have one-to-one, or one-to-many relationships.

Functions against selected MBO operations and object queries are also generated.

This is an example of the generated JavaScript for the Department MBO and Employee MBO in which the Department MBO has a one-to-many relationship with the Employee MBO.

```
/**
 * Returns The constructor of an mbo structure. This is helper
 function for creating MBO's operations or namedQuery input
 structure
 * @param attributes The parameters of an mbo operation, separated by
 one space. If the parameters map to MBO's attributes, use attributes
 name instead.
 * @param children The relationship names of an mbo operation's
 parameters or the array type of parameters, separated by one
 space.
 */
function makeClass(attributes, children) {
    var attributeNames = attributes.split(' ');
    var attributeCount = attributeNames.length;
    var childrenNames = children.split(' ');
    var childrenCount = childrenNames.length;

    function constructor() {
        for (var i = 0; i < attributeCount; i++)
        {
            var name = attributeNames[i];
            var subAttr = null;

            //If the name contains . which should be structure,
            while( name.indexOf('.') > 0 )
            {
                var part = name.substring( 0,
name.indexOf('.'));
                if ( subAttr )
                {
                    subAttr.part = new Object();
                    subAttr = subAttr.part;
                }
            }
        }
    }
}
```



```

        }else
        {
            this[part]= new Object();
            subAttr = this[part];
        }
        name = name.substring( name.indexOf('.')+1,
name.length);
    }

    if ( subAttr )
    {
        subAttr[name]= new Object();
    }else {
        this[name] = new Object();
    }
}

for (var i = 0; i < childrenCount; i++) {
    this[childrenNames[i]] =[];
    this['OldValue_' + childrenNames[i]] =[];
}

this['__state'] ="";
this['pks'] = {};

var self = this;

this['pks'].put = function(pkName, pkValue ) {
    self['pks'][pkName] = pkValue ;
}
}
return constructor;
}

```

Set the "__state" field to "add," "delete," or "update" to add or delete an MBO, or to update a child MBO to a parent MBO, respectively.

Use the "pks" field to set values for operation parameters that have personalization keys.

This example shows the JavaScript structures generated for a Department MBO and Employee JavaScript:

```

/**
 * Returns Department MBO structure.
 * Used by JavaScript functions of
department_create_submit,department_create_onlineRequest,department
_update_submit,department_update_onlineRequest,department_delete_su
bmit,department_delete_onlineRequest,department_findAll,department_
findByPrimaryKey
 * @param dept_id The "dept_id" is attribute field of MBO
Department
 * @param dept_name The "dept_name" is attribute field of MBO
Department
 * @param dept_head_id The "dept_head_id" is attribute field of MBO
Department
 * @param Employee is MBO Employee javaScript structure array which

```

```

representing the MBO Department to MBO Employee one to many
relationship
*/
Department = makeClass( "dept_id dept_name dept_head_id",
"Employee" );
/**
 * Returns Employee MBO structure.
 * Used by JavaScript functions of
employee_create_submit,employee_create_onlineRequest,employee_updat
e_submit,employee_update_onlineRequest,employee_delete_submit,emplo
yee_delete_onlineRequest,employee_findAll,employee_findByPrimaryKey

 * @param emp_id The "emp_id" is attribute field of MBO Employee
 * @param manager_id The "manager_id" is attribute field of MBO
Employee
 * @param emp_fname The "emp_fname" is attribute field of MBO
Employee
 * @param emp_lname The "emp_lname" is attribute field of MBO
Employee
 * @param dept_id The "dept_id" is attribute field of MBO Employee
 * @param street The "street" is attribute field of MBO Employee
 * @param city The "city" is attribute field of MBO Employee
 * @param state The "state" is attribute field of MBO Employee
 * @param zip_code The "zip_code" is attribute field of MBO
Employee
 * @param phone The "phone" is attribute field of MBO Employee
 * @param status The "status" is attribute field of MBO Employee
 * @param ss_number The "ss_number" is attribute field of MBO
Employee
 * @param salary The "salary" is attribute field of MBO Employee
 * @param start_date The "start_date" is attribute field of MBO
Employee
 * @param termination_date The "termination_date" is attribute field
of MBO Employee
 * @param birth_date The "birth_date" is attribute field of MBO
Employee
 * @param bene_health_ins The "bene_health_ins" is attribute field of
MBO Employee
 * @param bene_life_ins The "bene_life_ins" is attribute field of MBO
Employee
 * @param bene_day_care The "bene_day_care" is attribute field of MBO
Employee
 * @param sex The "sex" is attribute field of MBO Employee
 */
Employee = makeClass( "emp_id manager_id emp_fname emp_lname dept_id
street city state zip_code phone status ss_number salary start_date
termination_date birth_date bene_health_ins
bene_life_insbene_day_care sex" , "" );

```

If there is one parameter that does not map to the MBO's attribute, the JavaScript structure for the MBO's function input parameters is generated. This example shows an MBO called Banks where the dataSource is an SAP® object. In addition to the Banks JavaScript structure, the BANK_LIST and Banks_getList JavaScript structures are also generated.

```

/***** DEFINITION OF MBO JAVASCRIPT STRUCTURE *****/
/*****/
/**
 * Returns BANK_LIST structure
 * @param BANK_CTRY The "BANK_CTRY" is the parameter field of
BANK_LIST.
 * @param BANK_NAME The "BANK_NAME" is the parameter field of
BANK_LIST.
 */
BANK_LIST = makeClass("BANK_CTRY BANK_NAME" , "" )
/**
 * Returns Banks_getList structure. Used by functions of
banks_getList submit and banks_getList_onlineRequest
 * @param BANK_CTRY The "BANK_CTRY" is the parameter field of the
operation of getList.
 * @param BANK_LIST The "BANK_LIST" is the array parameter field of
the operation of getList.
 */
Banks_getList = makeClass( "BANK_CTRY", "BANK_LIST" );

/**
 * Returns Banks MBO structure.
 * Used by JavaScript functions of banks_findAll,banks_getByName
 * @param BANK_CTRY The "BANK_CTRY" is attribute field of MBO Banks
 * @param BANK_KEY The "BANK_KEY" is attribute field of MBO Banks
 * @param BANK_NAME The "BANK_NAME" is attribute field of MBO Banks
 * @param CITY The "CITY" is attribute field of MBO Banks
 * @param BAPI_BANK_GETLIST_BANK_LIST1 is MBO
BAPI_BANK_GETLIST_BANK_LIST1 javaScript structure array which
representing the MBO Banks to MBO BAPI_BANK_GETLIST_BANK_LIST1 one to
many relationship
 */
Banks = makeClass( "BANK_CTRY BANK_KEY BANK_NAME CITY" ,
"BAPI_BANK_GETLIST_BANK_LIST1" );

```

Global variables are generated for each MBO operation. You can reference these global variables in your code when you process incoming data to check which action was performed for the incoming message.

```

/*
 * Global variables for Customer actions
 */
Customer.createAction = "Customer_create";
Customer.updateAction = "Customer_update";
Customer.deleteAction = "Customer_delete";
Customer.findAllAction = "Customer_findAll";
Customer.findByPrimaryKeyAction = "Customer_findByPrimaryKey" ;

```

Two versions of JavaScript functions are generated for the MBO's create, read, update, delete operations. For example, for a **create** operation there is a **create_submit** function and **create_onlinerequest** function generated. This example shows the generated JavaScript function for the Department **create** operation:

```

/**
 * Returns void. This is submit operation, therefore no message will
 * be sent back to user by the hybrid web container
 * @param departmentObj which is the instance of Department
 * JavaScript structure. Values should be set for this instance.
 * @param credInfo, It is string value , should be something look
 * like "supusername=username&suppassword=password".
 * @param keepOpen, If this set to true, the Hybrid App will be kept
 * open, otherwise, the hybrid web container will close the Hybrid
 * App.
 */

function department_create_submit(departmentObj, credInfo,
keepOpen)
{
    //Collect values from departmentObj customerObj and fill the
    action parameters
    var keys = ["Department_create_dept_id_paramKey",
"Department_create_dept_name_paramKey",
"Department_create_dept_head_id_paramKey"];
    var types = ["int", "string", "int"];
    var objValues = [departmentObj.dept_id, departmentObj.dept_name,
departmentObj.dept_head_id];

    var workflowMessageToSend = new WorkflowMessage("");

    workflowMessageToSend.setHeader("");

workflowMessageToSend.setRequestAction( "Department_create");

    createMessageValues( workflowMessageToSend.getValues(), keys,
types, objValues );

    if ( departmentObj.Employee && departmentObj.Employee.length >
0 )
    // we have list object array
    {
        var department_employees = new MessageValue();
        department_employees.key = "Department_employees";
        department_employees.isNull = false;
        department_employees.type = "LIST";

        var employeekeys = ["Employee_emp_id_attribKey",
"Employee_manager_id_attribKey", "Employee_emp_fname_attribKey",
"Employee_emp_lname_attribKey", "Employee_dept_id_attribKey",
"Employee_street_attribKey", "Employee_city_attribKey",
"Employee_state_attribKey", "Employee_zip_code_attribKey",
"Employee_phone_attribKey", "Employee_status_attribKey",
"Employee_ss_number_attribKey", "Employee_salary_attribKey",
"Employee_start_date_attribKey",
"Employee_termination_date_attribKey",
"Employee_birth_date_attribKey",
"Employee_bene_health_ins_attribKey",
"Employee_bene_life_ins_attribKey",
"Employee_bene_day_care_attribKey", "Employee_sex_attribKey"];
    }
}

```

```

        var employeetypes = ["int", "int", "string", "string", "int",
"string", "string", "string", "string", "string", "string",
"string", "decimal", "DateTime", "DateTime", "DateTime", "string",
"string", "string", "string"];

        var employeeValues = [];

        for( var employeei = 0 ; employeei <
departmentObj.Employee.length ; employeei ++ )
        {
            var employeeelc = new MessageValueCollection();
            employeeelc.key = guid();
            employeeelc.parent = "Department_employees";
            employeeelc.parentValue = department_employees
            employeeelc.state =
departmentObj.Employee[employeei].__state;

            var employeeObjValues = [];

            employeeObjValues.push( departmentObj.Employee[employeei].emp_id);

            employeeObjValues.push( departmentObj.Employee[employeei].manager_id);

            employeeObjValues.push( departmentObj.Employee[employeei].emp_fname
);

            employeeObjValues.push( departmentObj.Employee[employeei].emp_lname
);

            employeeObjValues.push( departmentObj.dept_id);
            employeeObjValues.push( departmentObj.Employee[employeei].street);

            employeeObjValues.push( departmentObj.Employee[employeei].city);

            employeeObjValues.push( departmentObj.Employee[employeei].state);

            employeeObjValues.push( departmentObj.Employee[employeei].zip_code
);

            employeeObjValues.push( departmentObj.Employee[employeei].phone);

            employeeObjValues.push( departmentObj.Employee[employeei].status);

            employeeObjValues.push( departmentObj.Employee[employeei].ss_number

```

```

);

employeeObjValues.push( departmentObj.Employee[employeei].salary);

    employeeObjValues.push( departmentObj.Employee[employeei].start_date);

employeeObjValues.push( departmentObj.Employee[employeei].termination_date);

employeeObjValues.push( departmentObj.Employee[employeei].birth_date);

employeeObjValues.push( departmentObj.Employee[employeei].bene_health_ins);

employeeObjValues.push( departmentObj.Employee[employeei].bene_life_ins);

employeeObjValues.push( departmentObj.Employee[employeei].bene_day_care);

    employeeObjValues.push( departmentObj.Employee[employeei].sex);


        createMessageValues( employeeelc ,employeekeys ,
employeeetypes, employeeObjValues );


        //Find this Employee old values if it has.
        for( var oldValueemployeei = 0 ; oldValueemployeei <
departmentObj.OldValue_Employee.length ;
            oldValueemployeei ++ )
        {
            if
( departmentObj.OldValue_Employee[ oldValueemployeei ].emp_id ===
            departmentObj.Employee[ employeei].emp_id
            )
            {
                var oldValue_employeekeys =
["_old.Department.emp_id", "_old.Department.manager_id",
"_old.Department.emp_fname", "_old.Department.emp_lname",
"_old.Department.dept_id", "_old.Department.street",
"_old.Department.city", "_old.Department.state",
"_old.Department.zip_code", "_old.Department.phone",
"_old.Department.status", "_old.Department.ss_number",
"_old.Department.salary", "_old.Department.start_date",
"_old.Department.termination_date", "_old.Department.birth_date",
"_old.Department.bene_health_ins", "_old.Department.bene_life_ins",
"_old.Department.bene_day_care", "_old.Department.sex"];
                var oldValue_employeeetypes = ["INT", "INT",
"STRING", "STRING", "INT", "STRING", "STRING", "STRING",
"STRING", "STRING", "STRING", "DECIMAL", "DATE", "DATE", "DATE",
"STRING", "STRING", "STRING", "STRING"];
                var oldValue_employeeValues = [];

```

```
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].emp_id);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].manager_id);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].emp_fname);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].emp_lname);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].dept_id);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].street);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].city);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].state);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].zip_code);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].phone);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].status);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].ss_number);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].salary);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].start_date);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].termination_date);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].birth_date);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].bene_health_ins);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].bene_life_ins);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
```

```

lueemployeei].bene_day_care);

oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldVa
lueemployeei].sex);

createMessageValues( employeeelc,oldValue_employeekeys ,
oldValue_employeeetypes, oldValue_employeeValues );

                break;
            }
        }
// end of old values --->

        employeeValues.push( employeeelc);

    }

    department_employees.setValue(employeeValues);

    workflowValues.add( department_employees.getKey(),
department_employees);

    }
    hwc.doSubmitWorkflow_CONT( credInfo,
workflowMessageToSend.serializeToString(),workflowMessageToSend.get
HasFileMessageValue());
}
/**
 * Returns void. This is an onlineRequest operation, therefore the
message will be sent back to the user by the Hybrid Web Container.
Handle the result in the function
customAfterDataReceived(incomingWorkflowMessage)defined in
Custom.js.
 * @param departmentObj, which is the instance of Department
JavaScript structure. Values should be set for this instance.
 * @param credInfo, which is a string value, and should look like
"supusername=username&suppassword=password".
 * @param errorCallback, name of the function to be called if an
online request fails.
 */

function department_create_onlineRequest(departmentObj, credInfo ,
errorCallback)
{
    var keys = ["Department_create_dept_id_paramKey",
"Department_create_dept_name_paramKey",
"Department_create_dept_head_id_paramKey"];
    .....
    ....
    ..

```


WorkflowClient.xml

The WorkflowClient.xml file defines all of an application's action mappings that correspond to selected MBO operations and named queries.

Below is part of an example of the generated WorkflowClient.xml for the create operation on the Department MBO. Since the department has a one-to-many relationship to the Employee MBO, all input mappings for Department MBO and Employee MBO are also defined.

```
<?xml version="1.0" encoding="utf-8"?>
<Workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="WorkflowClient.xsd" >
  <Globals>
    <DefaultScreens activation="" credentials=""/>
  </Globals>
  <Triggers>
    <Actions>
      <Action name="Department_create" sourcescreen=""
targetscreens="" errorscreens="">
        <Methods>
          <Method type="replay" mbo="Department"
package="apiTestttDepartmentOneToMany:1.0"
showCredScreenOnAuthFailure="true" >
            <InputBinding optype="create" opname="create"
generateOld="false">
              <Value sourceType="Key"
workflowKey="Department_create_dept_id_paramKey"
paramName="dept_id" attribName="dept_id" mboType="int"/>
              <Value sourceType="Key"
workflowKey="Department_create_dept_name_paramKey"
paramName="dept_name" attribName="dept_name" mboType="string"/>
              <Value sourceType="Key"
workflowKey="Department_create_dept_head_id_paramKey"
paramName="dept_head_id" attribName="dept_head_id" mboType="int"/>
            >
              <Value sourceType="Key"
workflowKey="Department_employees" relationShipName="employees"
mboType="list">
                <InputBinding actiontype="create" optype="create"
opname="create" generateOld="true">
                  <Value sourceType="Key"
workflowKey="Employee_emp_id_attribKey" contextVariable=""
paramName="emp_id" attribName="emp_id" mboType="int"/>
                  <Value sourceType="Key"
workflowKey="Employee_manager_id_attribKey" contextVariable=""
paramName="manager_id" attribName="manager_id" mboType="int"/>
                  <Value sourceType="Key"
workflowKey="Employee_emp_fname_attribKey" contextVariable=""
paramName="emp_fname" attribName="emp_fname" mboType="string"/>
                  <Value sourceType="Key"
workflowKey="Employee_emp_lname_attribKey" contextVariable=""
paramName="emp_lname" attribName="emp_lname" mboType="string"/>
                  <Value sourceType="Key"
```

```

workflowKey="Employee_dept_id_attriKey" contextVariable=""
paramName="dept_id" attriName="dept_id" mboType="int"/>
    <Value sourceType="Key"
workflowKey="Employee_street_attriKey" contextVariable=""
paramName="street" attriName="street" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_city_attriKey" contextVariable=""
paramName="city" attriName="city" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_state_attriKey" contextVariable=""
paramName="state" attriName="state" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_zip_code_attriKey" contextVariable=""
paramName="zip_code" attriName="zip_code" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_phone_attriKey" contextVariable=""
paramName="phone" attriName="phone" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_status_attriKey" contextVariable=""
paramName="status" attriName="status" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_ss_number_attriKey" contextVariable=""
paramName="ss_number" attriName="ss_number" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_salary_attriKey" contextVariable=""
paramName="salary" attriName="salary" mboType="decimal"/>
    <Value sourceType="Key"
workflowKey="Employee_start_date_attriKey" contextVariable=""
paramName="start_date" attriName="start_date" mboType="date"/>
    <Value sourceType="Key"
workflowKey="Employee_birth_date_attriKey" contextVariable=""
paramName="birth_date" attriName="birth_date" mboType="date"/>
    <Value sourceType="Key"
workflowKey="Employee_bene_health_ins_attriKey" contextVariable=""
paramName="bene_health_ins" attriName="bene_health_ins"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_bene_life_ins_attriKey" contextVariable=""
paramName="bene_life_ins" attriName="bene_life_ins"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_bene_day_care_attriKey" contextVariable=""
paramName="bene_day_care" attriName="bene_day_care"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_sex_attriKey" contextVariable=""
paramName="sex" attriName="sex" mboType="string"/>
    </InputBinding>
    <InputBinding optype="none">
    <Value sourceType="Key"
workflowKey="Employee_emp_id_attriKey" attriName="emp_id"
mboType="int"/>
    <Value sourceType="Key"
workflowKey="Employee_manager_id_attriKey" attriName="manager_id"
mboType="int"/>
    <Value sourceType="Key"

```

```

workflowKey="Employee_emp_fname_attribKey" attribName="emp_fname"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_emp_lname_attribKey" attribName="emp_lname"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_dept_id_attribKey" attribName="dept_id"
mboType="int"/>
    <Value sourceType="Key"
workflowKey="Employee_street_attribKey" attribName="street"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_city_attribKey" attribName="city"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_state_attribKey" attribName="state"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_zip_code_attribKey" attribName="zip_code"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_phone_attribKey" attribName="phone"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_status_attribKey" attribName="status"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_ss_number_attribKey" attribName="ss_number"
mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_salary_attribKey" attribName="salary"
mboType="decimal"/>
    <Value sourceType="Key"
workflowKey="Employee_start_date_attribKey" attribName="start_date"
mboType="date"/>
    <Value sourceType="Key"
workflowKey="Employee_birth_date_attribKey" attribName="birth_date"
mboType="date"/>
    <Value sourceType="Key"
workflowKey="Employee_bene_health_ins_attribKey"
attribName="bene_health_ins" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_bene_life_ins_attribKey"
attribName="bene_life_ins" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_bene_day_care_attribKey"
attribName="bene_day_care" mboType="string"/>
    <Value sourceType="Key"
workflowKey="Employee_sex_attribKey" attribName="sex"
mboType="string"/>
</InputBinding>
<InputBinding actiontype="update" optype="update"
opname="update" generateOld="true">
    <Value sourceType="Key"
workflowKey="Employee_manager_id_attribKey" contextVariable=""
paramName="manager_id" attribName="manager_id" mboType="int"/>

```

```

        <Value sourceType="Key"
workflowKey="Employee_emp_fname_attribKey" contextVariable=""
paramName="emp_fname" attribName="emp_fname" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_emp_lname_attribKey" contextVariable=""
paramName="emp_lname" attribName="emp_lname" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_dept_id_attribKey" contextVariable=""
paramName="dept_id" attribName="dept_id" mboType="int"/>
        <Value sourceType="Key"
workflowKey="Employee_street_attribKey" contextVariable=""
paramName="street" attribName="street" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_city_attribKey" contextVariable=""
paramName="city" attribName="city" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_state_attribKey" contextVariable=""
paramName="state" attribName="state" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_zip_code_attribKey" contextVariable=""
paramName="zip_code" attribName="zip_code" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_phone_attribKey" contextVariable=""
paramName="phone" attribName="phone" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_status_attribKey" contextVariable=""
paramName="status" attribName="status" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_ss_number_attribKey" contextVariable=""
paramName="ss_number" attribName="ss_number" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_salary_attribKey" contextVariable=""
paramName="salary" attribName="salary" mboType="decimal"/>
        <Value sourceType="Key"
workflowKey="Employee_start_date_attribKey" contextVariable=""
paramName="start_date" attribName="start_date" mboType="date"/>
        <Value sourceType="Key"
workflowKey="Employee_birth_date_attribKey" contextVariable=""
paramName="birth_date" attribName="birth_date" mboType="date"/>
        <Value sourceType="Key"
workflowKey="Employee_bene_health_ins_attribKey" contextVariable=""
paramName="bene_health_ins" attribName="bene_health_ins"
mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_bene_life_ins_attribKey" contextVariable=""
paramName="bene_life_ins" attribName="bene_life_ins"
mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_bene_day_care_attribKey" contextVariable=""
paramName="bene_day_care" attribName="bene_day_care"
mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_sex_attribKey" contextVariable=""
paramName="sex" attribName="sex" mboType="string"/>
        <Value sourceType="Key"
workflowKey="Employee_emp_id_attribKey" contextVariable=""

```

```
paramName="emp_id" attribName="emp_id" mboType="int"/>
    </InputBinding>
    <InputBinding actiontype="delete" optype="delete"
opname="delete" generateOld="true">
    </InputBinding>
    </Value>

    </InputBinding>
    <OutputBinding/>
    </Method>
    </Methods>
</Action>
```

By default, the MBO has two named queries—`FindById` and `FindAll`. The method, input and output binding keys, and all of the dependency's key bindings are generated.

```
<Action name="Department_findByPrimaryKey" sourcescreen=""
targetscreen="" errorscreen="">
    <Methods>
        <Method name="findByPrimaryKey" type="search"
mbo="Department" package="apiTestttDepartmentOneToMany:1.0"
showCredScreenOnAuthFailure="true" >
            <InputBinding optype="none" opname="findByPrimaryKey"
generateOld="true">
                <Value sourceType="Key"
workflowKey="Department_dept_id_attribKey" paramName="dept_id"
attribName="dept_id" mboType="int"/>
            </InputBinding>
            <OutputBinding generateOld="true">
                <Mapping workflowKey="Department_dept_id_attribKey"
workflowType="number" attribName="dept_id" mboType="int"/>
                <Mapping workflowKey="Department_dept_name_attribKey"
workflowType="text" attribName="dept_name" mboType="string"/>
                <Mapping workflowKey="Department_dept_head_id_attribKey"
workflowType="number" attribName="dept_head_id" mboType="int"/>
                <Mapping workflowKey="Department_employees"
workflowType="list" relationshipName="employees" mboType="list">
                    <Mapping workflowKey="Employee_emp_id_attribKey"
workflowType="number" relationshipName="employees"
attribName="emp_id" mboType="int"/>
                    <Mapping workflowKey="Employee_manager_id_attribKey"
workflowType="number" relationshipName="employees"
attribName="manager_id" mboType="int"/>
                    <Mapping workflowKey="Employee_emp_fname_attribKey"
workflowType="text" relationshipName="employees"
attribName="emp_fname" mboType="string"/>
                    <Mapping workflowKey="Employee_emp_lname_attribKey"
workflowType="text" relationshipName="employees"
attribName="emp_lname" mboType="string"/>
                    <Mapping workflowKey="Employee_dept_id_attribKey"
workflowType="number" relationshipName="employees"
attribName="dept_id" mboType="int"/>
                    <Mapping workflowKey="Employee_street_attribKey"
workflowType="text" relationshipName="employees" attribName="street"
mboType="string"/>
                    <Mapping workflowKey="Employee_city_attribKey"
```

```

workflowType="text" relationshipName="employees" attribName="city"
mboType="string"/>
    <Mapping workflowKey="Employee_state_attribKey"
workflowType="text" relationshipName="employees" attribName="state"
mboType="string"/>
    <Mapping workflowKey="Employee_zip_code_attribKey"
workflowType="text" relationshipName="employees"
attribName="zip_code" mboType="string"/>
    <Mapping workflowKey="Employee_phone_attribKey"
workflowType="text" relationshipName="employees" attribName="phone"
mboType="string"/>
    <Mapping workflowKey="Employee_status_attribKey"
workflowType="text" relationshipName="employees" attribName="status"
mboType="string"/>
    <Mapping workflowKey="Employee_ss_number_attribKey"
workflowType="text" relationshipName="employees"
attribName="ss_number" mboType="string"/>
    <Mapping workflowKey="Employee_salary_attribKey"
workflowType="number" relationshipName="employees"
attribName="salary" mboType="decimal"/>
    <Mapping workflowKey="Employee_start_date_attribKey"
workflowType="date" relationshipName="employees"
attribName="start_date" mboType="date"/>
    <Mapping workflowKey="Employee_birth_date_attribKey"
workflowType="date" relationshipName="employees"
attribName="birth_date" mboType="date"/>
    <Mapping
workflowKey="Employee_bene_health_ins_attribKey"
workflowType="text" relationshipName="employees"
attribName="bene_health_ins" mboType="string"/>
    <Mapping
workflowKey="Employee_bene_life_ins_attribKey" workflowType="text"
relationshipName="employees" attribName="bene_life_ins"
mboType="string"/>
    <Mapping
workflowKey="Employee_bene_day_care_attribKey" workflowType="text"
relationshipName="employees" attribName="bene_day_care"
mboType="string"/>
    <Mapping workflowKey="Employee_sex_attribKey"
workflowType="text" relationshipName="employees" attribName="sex"
mboType="string"/>
    </Mapping>
    </OutputBinding>
  </Method>
</Methods>
</Action>

```

Note: By default, the <Notifications> section of the generated WorkflowClient.xml is empty, so you must write the <Notification> section for a server-initiated Hybrid App.

Creating Notifications to Make the Hybrid App Server-Initiated

To make the Hybrid App server-initiated, you must modify the `WorkflowClient.xml` file and create a notification.

By default, the `<Notifications>` section is empty.

1. Create a notification.

Each notification has two child nodes—Transformation and Methods.

2. Create a notification node, for example:

```
<Notifications>
  <Notification type="onEmailTriggered" targetscreens=" ">
    </Notification>
  </Notifications>
```

You can simply copy the Methods from the appropriate object query (for example, `findByPrimaryKey`) that is generated automatically in the `WorkflowClient.xml` file, for example:

```
<Notifications>
  <Notification type="onEmailTriggered" targetscreens=" ">
    <Methods>
      <Method name="findByPrimaryKey" type="search"
mbo="Department" package="apiTestDepartmentOneToMany:1.0"
showCredScreenOnAuthFailure="true" >
        <InputBinding optype="none" opname="findByPrimaryKey"
generateOld="true">
          <Value sourceType="Key"
workflowKey="Department_dept_id_attribKey" paramName="dept_id"
attribName="dept_id" mboType="int"/>
        </InputBinding>
        <OutputBinding generateOld="true">
          <Mapping workflowKey="Department_dept_id_attribKey"
workflowType="number" attribName="dept_id" mboType="int"/>
          [...]
        </OutputBinding>
      </Method>
    </Methods>
  </Notification>
</Notifications>
```

3. Create a Transformation node.

You must manually write the Transformation section. The contents depend on how many input parameters the object query has. For each input parameter, you need a corresponding Rule node as a child of the Transformation node. The `workflowKey` of the Rule node corresponds to the `InputBinding`'s `Value` for that input parameter. For example:

```
<Notifications>
  <Notification type="onEmailTriggered" targetscreens=" ">
    <Transformation>
  </Transformation>
    <Methods>
```

```

        <Method name="findByPrimaryKey"
            type="search" mbo="Department"
            package="apiTesttDepartmentOneToMany:1.0"
            showCredScreenOnAuthFailure="true" >
            <InputBinding optype="none" opname="findByPrimaryKey"
                generateOld="true">
                <Value sourceType="Key"
                    workflowKey="Department_dept_id_attriKey"
                    paramName="dept_id" attriName="dept_id"
                    mboType="int"/>
            </InputBinding>
            <OutputBinding
                generateOld="true">
                <Mapping
                    workflowKey="Department_dept_id_attriKey"
                    workflowType="number" attriName="dept_id"
                    mboType="int"/>
                [...]
            </OutputBinding>
            </Method>
        </Methods>
    </Notification>
</Notifications>

```

4. For each input parameter in the object query, create a corresponding Rule and make sure the workflowKey of the Rule matches the Value of the InputBinding. For example:

```

<Notifications>
    <Notification type="onEmailTriggered" targetscreen=" " >
        <Transformation>
            <Rule type="regex-extract"
                source="subject" workflowKey="dept_id"
                workflowType="number" beforeMatch="dept_id ="
                " afterMatch="" format=""/>
            </Transformation>
        <Methods>
            <Method name="findByPrimaryKey"
                type="search" mbo="Department"
                package="apiTesttDepartmentOneToMany:1.0"
                showCredScreenOnAuthFailure="true" >
                <InputBinding optype="none" opname="findByPrimaryKey"
                    generateOld="true">
                    <Value sourceType="Key"
                        workflowKey="dept_id"
                        paramName="dept_id" attriName="dept_id" mboType="int"/>
                </InputBinding>
                <OutputBinding
                    generateOld="true">
                    <Mapping
                        workflowKey="Department_dept_id_attriKey"
                        workflowType="number" attriName="dept_id"
                        mboType="int"/>
                    [...]
                </OutputBinding>
            </Method>
        </Methods>
    </Notification>
</Notifications>

```



```
</Notification>
</Notifications>
```

5. Save the file.

Processing Responses From the Server

There are a couple of approaches for handling callback functions.

If you want to use the JavaScript APIs generated by the wizard, for online request functions, you must implement the function:

```
hwc.processDataMessage = function processDataMessage
(incomingDataMessageValue, noUI, loading, fromActivationFlow,
dataType) {

    // for example,
    // var workflowMessage = new
WorkflowMessage(incomingWorkflowMessage);

    //if ( workflowMessage.getRequestAction() ==
Customer.findByPrimaryKeyAction ){
    //so this workflow message is returned by calling
customer_findByPrimaryKey function

    //TODO; do whatever you want to do with the return data....

}
```

You can choose, instead, to take advantage of the other functions in the `SMP_HOME\UnwiredPlatform\MobileSDK<version>\` folder, specifically the files under the `AppFramework` folder. In these, the incoming and outgoing messages, how they are bound to the UI, and how navigation works are handled by the functions defined in the `API.js` and `Utils.js` files. You can add your custom code into your own JavaScript file. You must still create the UI and do so in a way that is compatible with the `AppFramework`.

Error Handling

Usually, errors come from the Hybrid Web Container or from the back-end server side.

For online requests, when the error comes from the Hybrid Web Container, handle it in the `errorCallback` function, for example:

```
department_create_onlineRequest(dep1,
                                "",
                                function(errorMessage) {
                                    //TODO: error occurred...
                                }
);
```

An error message passed as an incoming Hybrid App message in the user-defined function of `processDataMessage` is another type of error that comes from the back-end server. The `incomingDataMessageValue` should be similar to this:

```
<M><H><S>...<S/> ..<V k="ErrorLogMessage" t="T">ERROR:.....</V><V
k=ErrorLogMessageAsList> ....</V>..</M>

hwc.processDataMessage = function processDataMessage
(incomingDataMessageValue, noUI, loading, fromActivationFlow,
dataType) {

    //// var workflowMessage = new
WorkflowMessage(incomingWorkflowMessage);

    //if ( workflowMessage.getRequestAction() ==
Customer.findByPrimaryKeyAction ){
    //    var detailErrorMsg =
workflowMessage.getValues().getData("ErrorLogMessage").getValue();
    // }

}
```

URL Parameters

When writing your own HTML and JavaScript, when the document is loaded, these URL parameters are present.

You can find an example of how to use these URL parameters in the `onHybridAppLoad()` function in the `Utils.js` file.

URL parameter	Description
loglevel	Current device log level.
screenToShow	Name of the screen to show.
supusername	User name of the current Hybrid App (if available).
lang	Current language of the device.
isalreadyprocessed	Indicates whether or not the Hybrid App message has been processed. The JavaScript can, for example, choose to show all controls as read-only if it has already been processed but viewed again.
loadtransformdata	Indicates that the JavaScript should request the transform data (contents of the e-mail message) from the Hybrid Web Container using the <code>loadtransformdata</code> query type. For information about the query types, see the topic <i>Calling the Hybrid Web Container</i> .

URL parameter	Description
ignoretransformsreen	Indicates that the JavaScript should ignore the <code>RequestScreen</code> tag in the transform data (contents of the message). This is set to true when the screen to show is either the Activation or Credentials screen.

Develop OData-based Hybrid Apps

The Hybrid App SDK includes the open source Datajs JavaScript library, which you can include as part of your application to access OData stores.

This library, along with the rest of the Hybrid App JavaScript API, is in `<SMP_HOME>\MobileSDK<version>\HybridApp\API\Container\Datajs-1.x.x.js`. As of this writing, the latest version of Datajs is 1.0.3.

The Datajs library is used to fetch the data used in your Hybrid App. This data can be displayed in your Hybrid App using a variety of UI frameworks such as jQuery Mobile, Sencha, or your favorite Web-based UI framework. Using OData in Hybrid Apps is similar to using the Rest API, described in *Developer Guide: REST API Applications*.

If the back-end OData service has support, you can use the Datajs library to read, modify, and delete data using standard HTTP methods (POST, PUT, DELETE, and so on).

The basic steps for developing an OData-based Hybrid App are:

1. Add the `<SMP_HOME>\UnwiredPlatform\MobileSDK<version>\HybridApp\API\Containers\datajs-1.0.3.js` to your Hybrid App.
2. Create a Hybrid App user interface with your preferred UI framework.
3. Use the Datajs library for create, read, update, and delete operations to the OData or HTTP end point and bind it to the UI.
4. Use the packaging tool to generate the `manifest.xml` file and Hybrid App ZIP package.
5. Use the Deploy Wizard in SAP Control Center to deploy the Hybrid App ZIP file.

Connect to an OData Source

The Datajs JavaScript library supports reading and writing to an OData service using both the JSON and ATOM-based formats.

The endpoint is an HTTP based URI exposed by the server.

You can use the `ODATA.read` API with a URI to read data from a server. To add, update, or delete data, the `ODATA.request` API can be used along with a POST, PUT, or DELETE method.

You can see examples at <http://datajs.codeplex.com/wikipage?title=OData%20Code%20Snippets&referringTitle=Using%20OData>

In your Hybrid App, you can connect to the Proxy endpoint exposed by SAP Mobile Platform using the Datajs library. This gives administrators and developers control over the endpoint as only white listed endpoints are accessible from the Hybrid App and also restricts who is able to access the endpoint based on built in SAP Mobile Platform security mechanisms.

When using Datajs to access an OData service from the Hybrid Web Container, you must employ POST tunneling to use the PUT, MERGE, and DELETE methods. There is an explanation of how to use POST tunneling with Datajs here: <http://datajs.codeplex.com/wikipage?title=Frequently%20Asked%20Questions#post-tunneling>.

Creating a Proxy Connection (Whitelisting)

Create a new connection in SAP Control Center to allow a proxy connection (authenticated or anonymous) through SAP Mobile Platform.

Note: When you set the proxy property with the endpoint address in the application template (either as part of the application creation or editing the application template created for that application), a proxy connection is generated automatically.

1. In the left navigation pane, expand the Domains folder, and select the default domain.
2. Select **Connections**.
3. In the right administration pane, select the **Connections** tab, and click **New**.
4. Enter a unique Connection pool name.
The Connection pool must have the same name as the application ID.
5. Select the Proxy **Connection pool** type.
6. Select the appropriate template for the data source target from the **Use template** menu.
7. Set the **Address** property by clicking the corresponding cell and entering the address of the proxy endpoint. For example, `http://odata.example.com/v2/Catalog/`
8. Configure the following optional properties:

Note: You must manually enter the `EnableHttpProxy` and `EnableUrlRewrite` properties; these properties are not pre-populated in the menu. To access an external service, you must configure the `http.proxyHost` and `http.proxyPort` properties during server configuration in **SAP Control Center > Servers > Server Configuration > General > User Options**. If you set or change the setting for `http.proxyHost` and `http.proxyPort`, you must restart the services using the stop/start service scripts. For more information, see *Administer > SAP Mobile Server > Configuring SAP Mobile Server to Securely Communicate With an HTTP Proxy* in *SAP Control Center for SAP Mobile Platform*.

- **Allow Anonymous Access** – Enables anonymous authentication to SAP Mobile Platform.

- **Certificate Alias** – Client SSL certificate stored in the SAP Mobile Platform keystore to be forwarded to the EIS.
- **Username** – Username passed to the EIS.
- **Password** – Password passed to the EIS.
- **EnableHttpProxy** – Enables Internet proxy support in the proxy connector. EnableHttpProxy defaults to false. Set explicitly to `true` to enable.
- **EnableUrlRewrite** – Enables the proxy component to rewrite URLs (or URIs) embedded in a response, with SUP URLs that causes the client requests to be directed back to the SAP Mobile Server (proxy component), rather than to the back end server. EnableUrlRewrite defaults to true. Set explicitly to `false` to disable.
- **ProxyHost** – Host name of the proxy server.
- **ProxyPort** – Port number.

Note:

- On a proxy connection, if the header for X-SUP-BACKEND-URL is not NULL, or EnableUrlRewrite is false then no URL rewrite occurs for either the request or response content.
 - To access the external services, you must configure `EnableHttpProxy = True`, `ProxyHost = proxy`, `ProxyPort = 8080` in the connection pool.
 - In REST services, the proxy URL is fetched from the application ID which is sent from the client device. The same application name is also present in the connection pool. This proxy URL is used for request/response.
-

9. Click **OK** to register the connection pool.

Datajs OData Client Authentication in Hybrid Apps

Several authentication schemes are available when accessing a protected OData service through an SAP Mobile Platform proxy, from a Hybrid App, in JavaScript using Datajs.

- **Basic authentication** – Provide a username and password to login. This method is available when connecting through HTTP and one-way HTTPS.
- **SSO token** – Provide an SSO token to login. This method is available when connecting through HTTP and HTTPS and a token validation service is available and configured.
- **X.509 Mutual authentication through intermediary** – Provide a forwarded client certificate to login using the `SSL_CLIENT_CERT` header name containing forwarded a PEM-encoded client certificate. This method is available only through an appropriately configured HTTPS listener. The certificate forwarder must have the "SUP Impersonator" role to be authorized for this type of login. The certificate of the actual "SUP Impersonator" user cannot be used as a regular user certificate.

In each case, if common additional JavaScript is required for every `OData.read` or `OData.request` call, this is best implemented in a Datajs custom HTTP client. This is a wrapper and extension of the `OData.defaultHttpClient` using the JavaScript proxy pattern. See <http://datajs.codeplex.com/wikipage?title=Custom%20OData%20httpClient>

Basic Authentication

The Datajs JavaScript library internally uses the XMLHttpRequest (XHR) object to handle the underlying HTTP or HTTPS requests/responses on the client.

The XHR API's open method optionally accepts user name and password credentials passed through parameters. Likewise, the Datajs' request object can take user and password members that map to those parameters. If credentials are not passed and basic authentication is required, the client is challenged with HTTP status 401. If credentials are passed to the XHR object, internally it does not automatically send them on the first request. It submits the credentials only if challenged. If this standard procedure is all that is required from the calling OData script, normally additional script can be avoided.

The below sample script shows possible alternative approaches for handling a 401 status manually, or, in cases where the authentication needs to be centralized.

```
/**
 * Sybase Hybrid App version 2.2
 *
 * Datajs.SSO.js
 * This file will not be regenerated, and it is expected that the user
 * may want to
 * include customized code herein.
 *
 * The template used to create this file was compiled on Mon Jul 9
 * 19:54:04 CST 2012
 *
 * Copyright (c) 2012 Sybase Inc. All rights reserved.
 */

// Capture datajs' current http client object.
var oldClient = OData.defaultHttpClient;

var sso_username = "";
var sso_password = "";
var sso_session = "";
var sso_token = "";

// Creates new client object that will attempt to handle SSO
// authentication, specifically SiteMinder login,
// in order to gain access to a protected URL.
var ssoClient = {
    request: function (request, success, error) {

        // For basic authentication, XMLHttpRequest.open method can
        // take varUser and varPassword parameters.
        // If the varUser parameter is null ("") or missing and the
        // site requires authentication, the
        // component displays a logon window. Although this method
        // accepts credentials passed via parameter,
        // those credentials are not automatically sent to the server
        // on the first request. The varUser and
        // varP      assword parameters are not transmitted unless the
```

```

server challenges the client for credentials
    // with a 401 - Access Denied response. But SiteMinder may
require additional steps, so save for
    // later...
    if (request.user != undefined && request.password !=
undefined) {
        sso_username = request.user;
        sso_password = request.password;
    }

    var onSuccess = function (data, response) {
        // Browser control will automatically cache cookies, with
possible token, for next time, so
        // parsing Set-Cookie in HTTP response headers unnecessary
here.
        //var setCookieHeader = response.headers["Set-Cookie"];
        //var setCookies = [];
        //parseSetCookies(setCookieHeader, setCookies);

        //for(var i=0; i < setCookies.length; i++)
        //{
        //    if (setCookies[i].substr(0, 9) === "SMSESSION")
        //        sso_session = setCookies[i];
        //    else if (setCookies[i].substr(0, 9) === "MYSAPSSO2")
        //        sso_token = setCookies[i];
        //}

        // Call original success
        alert("Calling original success");
        success(data, response);
    }

    var onError = function (sso_error) {
        if (sso_error.response.statusCode == 0) {
            // Attempt to parse error from response.body, e.g. sent
from SAP NetWeaver as HTML page.
            if (sso_error.response.body.indexOf("401") != -1 &&
(sso_error.response.body.indexOf("Unauthorized") !=
-1 ||
sso_error.response.body.indexOf("UNAUTHORIZED") !=
-1)) {
                alert("SSO challenge detected");
                sso_error.response.statusCode = 401;
            }
        }

        // Ensure valid response. Expecting either HTTP status 401
for SMCHALLENGE or 302 for redirection.
        if (sso_error.response.statusCode != 401 &&
sso_error.response.statusCode != 302) {
            alertText(sso_error.response.statusText);
            error(sso_error);
            return;
        }

        // 401 may include SMCHALLENGE=YES in Set-Cookie, so need

```

```

to return along with Authorization
    // credentials to acquire SMSESSION cookie.
    if (sso_error.response.statusCode === 401) {
        // Browser control will automatically cache cookies,
        with possible token, for next time,
        // so parsing Set-Cookie in HTTP response headers
        unnecessary here.
        //var setCookieHeader =
        sso_error.response.headers["Set-Cookie"];
        //var setCookies = [];
        //parseSetCookies(setCookieHeader, setCookies);

        // Append existing headers.
        var newHeaders = [];
        if (request.headers) {
            for (name in request.headers) {
                newHeaders[name] = request.headers[name];
            }
        }
        // Browser control should include SMCHALLENGE cookie.
        //newHeaders["Cookie"] = "SMCHALLENGE=YES";
        var enc_username = window.btoa(sso_username);
        var enc_password = window.btoa(sso_password);
        var basic_auth = "Basic " + enc_username + ":" +
enc_password;
        newHeaders["Authorization"] = basic_auth;

        // Redo the OData request for the protected resource.
        var newRequest = {
            headers: newHeaders,
            requestUri: request.requestUri,
            method: request.method,
            user: sso_username,
            password: sso_password
        };

        oldClient.request(newRequest, onSuccess, error);
    }

    // 302 indicates that the requested information is located
    at the URI specified in the Location
    // header. The default action when this status is received
    is to follow the Location header
    // associated with the response. When the original request
    method was POST, the redirected request
    // will use the GET method.
    if (sso_error.response.statusCode === 302) {
        // Get the redirection location.
        var siteminder_url =
sso_error.response.headers["Location"];

        // Open a connection to the redirect and load the login
        form.
        // That screen can be used to capture the required form
        fields.
    
```



```

        var httpRedirect = getXMLHttpRequest();

        httpRedirect.onload = function () {

            if (this.status < 200 || this.status > 299) {
                alert("Error: " + this.status);
                alertText(this.statusText);
                error({ message: this.statusText });
                return;
            }

            var sm_form_response = this.responseXML;
            var siteminder_tags = {};

            getSiteMinderTags(sm_form_response,
siteminder_tags);

            // Create the form data to post back to SiteMinder.
            Two ContentTypes are valid for sending
            // POST data. Default is application/x-www-form-
            urlencoded and form data is formatted
            // similar to typical querystring. Forms submitted
            with this content type are encoded as
            // follows: Control names and values are escaped.
            Space characters are replaced by `+',
            // reserved characters are escaped as described in
            [RFC1738], section 2.2:
            // non-alphanumeric characters are replaced by `
            %HH', representing ASCII code of character.
            // Line breaks are represented as CRLF pairs (i.e.,
            `%0D%0A'). Control names/values are
            // listed in order they appear in document. Name is
            separated from value by '=' and name/
            // value pairs are separated from each other by `&'.
            Alternative is multipart/form-data.
            //var formData = new FormData();
            var postData = "";

            for (var inputName in siteminder_tags) {
                if (inputName.substring(0, 2).toLowerCase() ===
"sm") {
                    postData += inputName + "=" +
encodeURIComponent(siteminder_tags[inputName]) + "&";
                    // formData.append(inputName,
siteminder_tags[inputName]);
                }
            }
            postData += "postpreservationdata=&";
            postData += "USER=" +
encodeURIComponent(sso_username) + "&";
            postData += "PASSWORD=" +
encodeURIComponent(sso_password);

            // Submit data back to SiteMinder.
            var httpLogin = getXMLHttpRequest();

```

```

        httpLogin.onload = function () {

            if (this.status < 200 || this.status > 299) {
                alert("Error: " + this.status);
                alertText(this.statusText);
                error({ message: this.statusText });
                return;
            }

            // Browser control should cache required cookies
            // so no need to parse HTTP response
            // headers.
            //var sm_cookie response = this.response;
            //var setCookieHeader =
            this.getResponseHeader("Set-Cookie");
            //var setCookies = [];
            //parseSetCookies(setCookieHeader, setCookies);

            // Locate the URI to access next.
            var newUrl = this.getResponseHeader("Location");

            // Append existing headers.
            var newHeaders = [];
            if (request.headers) {
                for (name in request.headers) {
                    newHeaders[name] = request.headers[name];
                }
            }
            // Browser control should include SMSESSION
            //cookie.
            //newHeaders["Cookie"] = setCookieHeader;

            // Redo the OData request for the protected
            //resource.

            var newRequest = {
                headers: newHeaders,
                requestUri: newUrl,
                method: request.method,
                user: sso_username,
                password: sso_password
            };

            oldClient.request(newRequest, onSuccess, error);
        }

        httpLogin.open("POST", siteminder_url, true);
        httpLogin.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
        httpLogin.withCredentials = "true";
        httpLogin.send(postData);
        //httpLogin.send(formData);
    }

    httpRedirect.open("GET", siteminder_url, true);
    httpRedirect.responseType = "document";
    httpRedirect.send();

```

```

    }
}

// Call back into the original http client.
var result = oldClient.request(request, success, onError);
return result;
}
};

// Parses Set-Cookie from header into array of setCookies.
function parseSetCookies(setCookieHeader, setCookies) {

    if (setCookieHeader == undefined)
        return;

    var cookieHeaders = setCookieHeader.split(", ");

    // verify comma-delimited parse by ensuring '=' within each token
    var len = cookieHeaders.length;
    if (len > 0) {
        setCookies[0] = cookieHeaders[0];
    }
    var i, j;
    for (i = 1, j = 0; i < len; i++) {
        if (cookieHeaders[i]) {
            var eqdex = cookieHeaders[i].indexOf('=');
            if (eqdex != -1) {
                var semidex = cookieHeaders[i].indexOf(';');
                if (semidex == -1 || semidex > eqdex) {
                    setCookies[++j] = cookieHeaders[i];
                }
            }
            else {
                setCookies[j] += ", " + cookieHeaders[i];
            }
        }
        else {
            setCookies[j] += ", " + cookieHeaders[i];
        }
    }
}

// Parses response HTML document and returns array of INPUT tags.
function getSiteMinderTags(response, tags) {

    var inputs = new Array();
    inputs = response.getElementsByTagName("input");

    // get the 'input' tags
    for (var i = 0; i < inputs.length; i++) {
        var element = inputs.item(i).outerHTML;
        var value = "";

        // filter out inputs with type=button
        var stridex = element.indexOf("type=");

```

```
        if (strindex != -1) {
            var typ = element.substring(strindex + 5);
            strindex = typ.indexOf(' ');
            typ = typ.substring(0, strindex);

            if (typ.toLowerCase() === "button") {
                continue;
            }
        }

        strindex = element.indexOf("value=")
        if (strindex != -1) {
            value = element.substring(strindex + 6);
            strindex = value.indexOf(' ');
            value = value.substring(0, strindex);
        }

        tags[inputs.item(i).name] = value;
    }
}

function alertText(error) {

    var txt = JSON.stringify(error);
    alert("Error:\n" + txt);

    var length = txt.length;
    var sectionLength = 300;
    var index = Math.floor(length / sectionLength);
    for (i = 0; i <= index; i++) {
        var start = i * sectionLength;
        var end = (i + 1) * sectionLength;
        var segLength = sectionLength;
        if (end > length) segLength = length - start;
        alert(txt.substr(start, segLength));
    }
}

// Can either pass ssoClient explicitly, or set it globally for the
// page as the default:
OData.defaultHttpClient = ssoClient;
```

Authentication Against an OData Source

Hybrid Apps pass user name and password information using HTTP basic authentication, by setting the Authorization HTTP header.

It is recommended to use this in combination with SSL/TLS, otherwise user names and passwords are passed in cleartext. For example:

```
var strUsername = "odata";
var strPassword = "password";
var oHeaders = {};
oHeaders['Authorization'] = "Basic " + btoa(strUsername + ":" +
strPassword);
```

```

var request = {
    headers : oHeaders, // object that contains HTTP headers as
name value pairs
    requestUri : sUrl, // OData endpoint URI
    method : "GET"
};

OData.read( request, function (data, response) {

    // do something with the response
},
function( err ) {
    // handle error reading the data
});

```

SSO Token, Including SAP SSO2 and SiteMinder/Network Edge

As in basic authentication, the Data.js JavaScript library internally uses the XmlHttpRequest (XHR) object to handle the underlying HTTP or HTTPS requests/responses on the client.

From the XHR object's API, Data.js uses `setRequestHeader()` and `getAllResponseHeaders()` to send and read the HTTP headers in the request and response. For Single Sign-On and Network Edge authentication, issuers of SSO tokens, including SAP SSO2 logon tickets (MYSAPSSO2), as well as SiteMinder tokens (SMCHALLENGE, SMSESSION, and so on) normally use the "Set-Cookie" field in the HTTP header to send the token to the client, and expect the "Cookie" in the header to receive the token from the client.

However, these specific headers are omitted from JavaScript access. See the W3C spec (<http://www.w3.org/TR/XMLHttpRequest/>). Instead, these headers are designed to be controlled by the user agent, in this case the browser control hosted by the Hybrid Web Container, to protect the client from rogue sites. According to the W3C spec it is the job of the user agent to support HTTP state management: to persist, discard, and send cookies, as received in the Set-Cookie response header, and sent in the Cookie header, as applicable. One possible exception allows cookie handling in JavaScript to set up a CORS request on the client and server, using the XHR's "withCredentials" property.

Considering the reliance on the Hybrid Web Container-hosted browser control to handle the required SSO tokens, it is important to note the same origin policy surrounding automatic cookie management. That means from the client's perspective, the domain from where the cookie-based token originates must be the same as where it needs to be redirected to access the protected OData endpoint, such as the SAP NetWeaver Gateway, while authenticated. For the domain to be the same to the client, the URL pattern specifying transport protocol, servername, domain, and port number must match between token issuer and endpoint. This should be possible using proxy mappings in the Relay Server or reverse proxy.

Regarding the SiteMinder component of Network Edge, its Policy Server supports a variety of authentication schemes, including Basic Authentication and HTML Forms-based

Authentication. The sample script below demonstrates an approach to handling a Basic 401 challenge from SiteMinder, as well as possible Forms authentication, involving HTTP status 302 indicating redirection. The script involving cookie handling is commented out and just informational, since this is managed by the user agent as described previously.

```
/**
 * SAP Hybrid App version 2.2
 *
 * Dataajs.SSO.js
 * This file will not be regenerated, and it is expected that the user
 * may want to
 * include customized code herein.
 *
 * The template used to create this file was compiled on Mon Jul 9
 * 19:54:04 CST 2012
 *
 * Copyright (c) 2012 SAP Inc. All rights reserved.
 */

// Capture dataajs' current http client object.
var oldClient = OData.defaultHttpClient;

var sso_username = "";
var sso_password = "";
var sso_session = "";
var sso_token = "";

// Creates new client object that will attempt to handle SSO
// authentication, specifically SiteMinder login,
// in order to gain access to a protected URL.
var ssoClient = {
    request: function (request, success, error) {

        // For basic authentication, XMLHttpRequest.open method can
        // take varUser and varPassword parameters.
        // If the varUser parameter is null ("") or missing and the
        // site requires authentication, the
        // component displays a logon window. Although this method
        // accepts credentials passed via parameter,
        // those credentials are not automatically sent to the server
        // on the first request. The varUser and
        // varPassword parameters are not transmitted unless the
        // server challenges the client for credentials
        // with a 401 - Access Denied response. But SiteMinder may
        // require additional steps, so save for
        // later...
        if (request.user != undefined && request.password !=
            undefined) {
            sso_username = request.user;
            sso_password = request.password;
        }

        var onSuccess = function (data, response) {
            // Browser control will automatically cache cookies, with
            // possible token, for next time, so

```

```

        // parsing Set-Cookie in HTTP response headers unnecessary
here.
        //var setCookieHeader = response.headers["Set-Cookie"];
        //var setCookies = [];
        //parseSetCookies(setCookieHeader, setCookies);

        //for(var i=0; i < setCookies.length; i++)
        //{
        //    if (setCookies[i].substr(0, 9) === "SMSESSION")
        //        sso_session = setCookies[i];
        //    else if (setCookies[i].substr(0, 9) === "MYSAPSSO2")
        //        sso_token = setCookies[i];
        //}

        // Call original success
        alert("Calling original success");
        success(data, response);
    }

    var onError = function (sso_error) {
        if (sso_error.response.statusCode == 0) {
            // Attempt to parse error from response.body, e.g. sent
            from SAP NetWeaver as HTML page.
            if (sso_error.response.body.indexOf("401") != -1 &&
                (sso_error.response.body.indexOf("Unauthorized") !=
-1 ||
-1)) {
                sso_error.response.body.indexOf("UNAUTHORIZED") !=
-1)) {
                    alert("SSO challenge detected");
                    sso_error.response.statusCode = 401;
                }
            }

            // Ensure valid response. Expecting either HTTP status 401
            for SMCHALLENGE or 302 for redirection.
            if (sso_error.response.statusCode != 401 &&
                sso_error.response.statusCode != 302) {
                alertText(sso_error.response.statusText);
                error(sso_error);
                return;
            }

            // 401 may include SMCHALLENGE=YES in Set-Cookie, so need
            to return along with Authorization
            // credentials to acquire SMSESSION cookie.
            if (sso_error.response.statusCode === 401) {
                // Browser control will automatically cache cookies,
                with possible token, for next time,
                // so parsing Set-Cookie in HTTP response headers
                unnecessary here.
                //var setCookieHeader =
sso_error.response.headers["Set-Cookie"];
                //var setCookies = [];
                //parseSetCookies(setCookieHeader, setCookies);

```

```

        // Append existing headers.
        var newHeaders = [];
        if (request.headers) {
            for (name in request.headers) {
                newHeaders[name] = request.headers[name];
            }
        }
        // Browser control should include SMCHALLENGE cookie.
        //newHeaders["Cookie"] = "SMCHALLENGE=YES";
        var enc_username = window.btoa(sso_username);
        var enc_password = window.btoa(sso_password);
        var basic_auth = "Basic " + enc_username + ":" +
enc_password;
        newHeaders["Authorization"] = basic_auth;

        // Redo the OData request for the protected resource.
        var newRequest = {
            headers: newHeaders,
            requestUri: request.requestUri,
            method: request.method,
            user: sso_username,
            password: sso_password
        };

        oldClient.request(newRequest, onSuccess, error);
    }

    // 302 indicates that the requested information is located
    at the URI specified in the Location
    // header. The default action when this status is received
    is to follow the Location header
    // associated with the response. When the original request
    method was POST, the redirected request
    // will use the GET method.
    if (sso_error.response.statusCode === 302) {
        // Get the redirection location.
        var siteminder_url =
sso_error.response.headers["Location"];

        // Open a connection to the redirect and load the login
        form.
        // That screen can be used to capture the required form
        fields.

        var httpRedirect = getXMLHttpRequest();

        httpRedirect.onload = function () {

            if (this.status < 200 || this.status > 299) {
                alert("Error: " + this.status);
                alertText(this.statusText);
                error({ message: this.statusText });
                return;
            }

            var sm_form_response = this.responseXML;
            var siteminder_tags = {};

```



```

        getSiteMinderTags(sm_form_response,
siteminder_tags);

        // Create the form data to post back to SiteMinder.
        Two ContentTypes are valid for sending
        // POST data. Default is application/x-www-form-
        urlencoded and form data is formatted
        // similar to typical querystring. Forms submitted
        with this content type are encoded as
        // follows: Control names and values are escaped.
        Space characters are replaced by `+',
        // reserved characters are escaped as described in
        [RFC1738], section 2.2:
        // non-alphanumeric characters are replaced by `
        %HH', representing ASCII code of character.
        // Line breaks are represented as CRLF pairs (i.e.,
        `%0D%0A'). Control names/values are
        // listed in order they appear in document. Name is
        separated from value by '=' and name/
        // value pairs are separated from each other by `&'.
        Alternative is multipart/form-data.
        //var formData = new FormData();
        var postData = "";

        for (var inputName in siteminder_tags) {
            if (inputName.substring(0, 2).toLowerCase() ==
"sm") {
                postData += inputName + "=" +
encodeURIComponent(siteminder_tags[inputName]) + "&";
                // formData.append(inputName,
siteminder_tags[inputName]);
            }
        }
        postData += "postpreservationdata=" +
        postData += "USER=" +
        encodeURIComponent(sso_username) + "&";
        postData += "PASSWORD=" +
        encodeURIComponent(sso_password);

        // Submit data back to SiteMinder.
        var httpLogin = getXMLHttpRequest();

        httpLogin.onload = function () {

            if (this.status < 200 || this.status > 299) {
                alert("Error: " + this.status);
                alertText(this.statusText);
                error({ message: this.statusText });
                return;
            }

            // Browser control should cache required cookies
            so no need to parse HTTP response
            // headers.
            //var sm_cookie_response = this.response;
    
```

```

        //var setCookieHeader =
this.getResponseHeader("Set-Cookie");
        //var setCookies = [];
        //parseSetCookies(setCookieHeader, setCookies);

        // Locate the URI to access next.
var newUrl = this.getResponseHeader("Location");

        // Append existing headers.
var newHeaders = [];
        if (request.headers) {
            for (name in request.headers) {
                newHeaders[name] = request.headers[name];
            }
        }
        // Browser control should include SMSESSION
cookie.
        //newHeaders["Cookie"] = setCookieHeader;

        // Redo the OData request for the protected
resource.
        var newRequest = {
            headers: newHeaders,
            requestUri: newUrl,
            method: request.method,
            user: sso_username,
            password: sso_password
        };

        oldClient.request(newRequest, onSuccess, error);
    }

    httpLogin.open("POST", siteminder_url, true);
    httpLogin.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
    httpLogin.withCredentials = "true";
    httpLogin.send(postData);
    //httpLogin.send(formData);
    }

    httpRedirect.open("GET", siteminder_url, true);
    httpRedirect.responseType = "document";
    httpRedirect.send();

    }
}

// Call back into the original http client.
var result = oldClient.request(request, success, onError);
return result;
}
};

// Parses Set-Cookie from header into array of setCookies.
function parseSetCookies(setCookieHeader, setCookies) {

```

```

    if (setCookieHeader == undefined)
        return;

    var cookieHeaders = setCookieHeader.split(", ");

    // verify comma-delimited parse by ensuring '=' within each token
    var len = cookieHeaders.length;
    if (len > 0) {
        setCookies[0] = cookieHeaders[0];
    }
    var i, j;
    for (i = 1, j = 0; i < len; i++) {
        if (cookieHeaders[i]) {
            var eqdex = cookieHeaders[i].indexOf('=');
            if (eqdex != -1) {
                var semidex = cookieHeaders[i].indexOf(';');
                if (semidex == -1 || semidex > eqdex) {
                    setCookies[++j] = cookieHeaders[i];
                }
                else {
                    setCookies[j] += ", " + cookieHeaders[i];
                }
            }
            else {
                setCookies[j] += ", " + cookieHeaders[i];
            }
        }
    }
}

// Parses response HTML document and returns array of INPUT tags.
function getSiteMinderTags(response, tags) {

    var inputs = new Array();
    inputs = response.getElementsByTagName("input");

    // get the 'input' tags
    for (var i = 0; i < inputs.length; i++) {
        var element = inputs.item(i).outerHTML;
        var value = "";

        // filter out inputs with type=button
        var stridex = element.indexOf("type=");
        if (stridex != -1) {
            var typ = element.substring(stridex + 5);
            stridex = typ.indexOf(' ');
            typ = typ.substring(0, stridex);

            if (typ.toLowerCase() === "button") {
                continue;
            }
        }

        stridex = element.indexOf("value=");
        if (stridex != -1) {
            value = element.substring(stridex + 6);

```

```
        strindex = value.indexOf(' ');
        value = value.substring(0, strindex);
    }

    tags[inputs.item(i).name] = value;
}

function alertText(error) {

    var txt = JSON.stringify(error);
    alert("Error:\n" + txt);

    var length = txt.length;
    var sectionLength = 300;
    var index = Math.floor(length / sectionLength);
    for (i = 0; i <= index; i++) {
        var start = i * sectionLength;
        var end = (i + 1) * sectionLength;
        var segLength = sectionLength;
        if (end > length) segLength = length - start;
        alert(txt.substr(start, segLength));
    }
}

// Can either pass ssoClient explicitly, or set it globally for the
page as the default:
OData.defaultHttpClient = ssoClient;
```

Server Certificate Validation Over HTTPS

In this pattern, which uses the `CertificateAuthenticationLoginModule`, the server sends the client a certificate with which to authenticate itself.

The client uses the certificate to authenticate the identity the certificate claims to represent. An SSL-enabled client goes through these steps to authenticate a server's identity:

1. Is today's date within the valid period?
2. Is the issuing certificate authority (CA) a trusted one? Each SSL-enabled client maintains a list of trusted CA certificates. This list determines which server certificates the client accepts. Validation continues if the distinguished name (DN) of the issuing CA matches the DN of a certificate authority on the client's list of trusted certificate authorities.
3. Does the issuing certificate authority's public key validate the issuer's digital signature?
4. Does the domain name in the server's certificate match the domain name of the server itself?
5. The server is authenticated. The client proceeds with the SSL handshake. If the client does not get to step 5 for any reason, the server that is identified by the certificate cannot be authenticated, and the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established.

Similar to cookie-based tokens, certificate authentication is also outside the scope of pure JavaScript which has no access to certificates, and similarly falls under the control of the user agent, in this case again the browser control, and its interface directly with the user.

X.509 SSO Authentication

For certificate based SSO authentication, due to the restriction from handling certificates in pure JavaScript, a native counterpart on the device must be interfaced, such as the Hybrid Web Container, using its existing `Certificate.js`.

In this sample script, a Datajs custom HTTP client is used to encapsulate the client certificate component of certificate based SSO. You can provision signed certificate from a local file, a server, or from Afaria, based on the device platform, using the existing Certificate API. You can choose to set the results of the API call as the password.

```
/**
 * SAP Hybrid App version 2.2
 *
 * Datajs.Certificate.js
 * This file will not be regenerated, and it is expected that the user
may want to
 * include customized code herein.
 *
 * The template used to create this file was compiled on Mon Aug 23
16:43:02 CST 2012
 *
 * Copyright (c) 2012 SAP Inc. All rights reserved.
 */

// Capture datajs' current http client object.
var oldClient = OData.defaultHttpClient;

var cert_username = "";
var cert_password = "";

// Creates new client object that will attempt to handle Certificate
authentication.
var certClient = {
    request: function (request, success, error) {

        if (request.requestUri.substr(0, 8) === "https://")
        {
            if (request.password != undefined)
            {
                // The following script gets the signed certificate data
for the first
                // p12 file found on the sdcard
                var certStore = CertificateStore.getDefault();
                var certPaths =
certStore.listAvailableCertificatesFromFileSystem("/sdcard/",
"p12");
                var cert =
certStore.getSignedCertificateFromFile(certPaths[0],
request.password);
            }
        }
    }
}
```

```

        var cert_username = cert.subjectCN;
        var cert_password = cert.signedCertificate;

        // Redo the OData request for the protected resource
        var newRequest = {
            headers : request.headers,
            requestUri : request.requestUri,
            method : request.method,
            user : cert_username,
            password : cert_password
        };

        // Call back into the original http client.
        return oldClient.request(newRequest, success, error);
    }

    return oldClient.request(request, success, error);
}

};

// Can either pass certClient explicitly, or set it globally for the
page as the default:
OData.defaultHttpClient = certClient;

```

When sending a forwarded client certificate through an intermediary, set the value to “SSL_CLIENT_CERT” in the XHR’s HTTP request header, as shown in this example:

```

/**
 * SAP Hybrid App version 2.2
 *
 * Datajs.Certificate.js
 * This file will not be regenerated, and it is expected that the user
may want to
 * include customized code herein.
 *
 * The template used to create this file was compiled on Mon Aug 23
16:43:02 CST 2012
 *
 * Copyright (c) 2012 SAP Inc. All rights reserved.
 */

// Capture datajs' current http client object.
var oldClient = OData.defaultHttpClient;

// Creates new client object that will attempt to handle Certificate
authentication.
var certClient = {
    request: function (request, success, error) {

        if (request.requestUri.substr(0, 8) === "https://")
        {
            if (request.user != undefined && request.password !=
undefined)

```

```

        {
            // The following script gets the signed certificate
            data for the first
            // p12 file found on the sdcard
            var certStore = CertificateStore.getDefault();
            var certPaths = certStore.listAvailableCertificatesFromFileSystem("/sdcard/", "p12");
            var cert = certStore.getSignedCertificateFromFile(certPaths [0] ,
            request.password);

            // Append existing headers.
            var newHeaders = [];
            if (request.headers) {
                for (name in request.headers) {
                    newHeaders[name] = request.headers[name];
                }
            }
            //
            newHeaders["SSL_CLIENT_CERT"] = cert.signedCertificate;

            // Redo the OData request for the protected resource
            var newRequest = {
                headers : newHeaders,
                requestUri : request.requestUri,
                method : request.method,
                user : request.user,
                password : request.password
            };

            // Call back into the original http client.
            return oldClient.request(newRequest, success, error);
        }

        return oldClient.request(request, success, error);
    }
};

// Can either pass certClient explicitly, or set it globally for the
page as the default:
OData.defaultHttpClient = certClient;

```

Implementing Push

The backend OData source can proactively send notifications to Hybrid Apps.

SAP Mobile Platform enables this by exposing an HTTP based push interface `http://supserver:port/notifications/ApplicationConnectionID`.

The Hybrid App must inform the backend of its `ApplicationConnectionID`, usually on startup. You can obtain this by using the `hwc.getApplicationConnectionID()` JavaScript API. The backend service exposes an endpoint where said

ApplicationConnectionID can be sent when the Hybrid App starts up or "subscribes."
When the push notification is received, it can be handled in native code or JavaScript.

Enabling the Datajs Library on Windows Mobile

To enable the datajs-`<version>.js` library on Window Mobile 6.0 and Windows Mobile 6.1, you must add some custom code into the file where the Hybrid App is first launched.

For Windows Mobile 6.5, you need only to include the datajs-`<version>.js` library in your HTML file.

1. Open the JavaScript file where the Hybrid App is first launched, for example, Custom.js, which is located in `<SMP_HOME>\MobileSDK<version>\HybridApp\API\AppFramework`.

2. Add this code:

```
///Begin, This code enable datajs library on Windows 6.0 and
Windows6.1
window.oldActiveXObject = window.ActiveXObject;
window.ActiveXObject = function(id) {
try{ return new window.oldActiveXObject(id); }
catch (exception) {
if(isWindowsMobile()){
try{
if(id == "Msxml2.XMLHTTP.6.0" || id == "Msxml2.XMLHTTP.3.0")
{ return new window.oldActiveXObject("Microsoft.XMLHTTP"); }
if(id == "Msxml2.DOMDocument.6.0" || id == "Msxml2.DOMDocument.
3.0"){ return new window.oldActiveXObject("Microsoft.XMLDOM"); }
}
catch(e){ throw e; }
}
throw exception;
}
};
//End
```

3. Save the file.
4. Rebuild the Hybrid App project.

Hybrid Web Container and Hybrid App JavaScript APIs

The container and framework JavaScript APIs provide functionality that the Hybrid Apps can access.

Hybrid Web Container JavaScript APIs

The files where the Hybrid Web Container JavaScript APIs are defined are located in `<SMP_HOME>\MobileSDK<version>\HybridApp\API\Container`. The generated JavaScript API reference documents are located in `<SMP_HOME>\MobileSDK<version>\HybridApp\API\API`.

Note: The detail of the individual APIs is not available if you are viewing this document from DocCommentXchange (<http://dcx.sybase.com>) or in PDF format. You can access this information by going to Product Documentation: access <http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories>, then navigate to the current version of this topic.

Class	Description	Defined in
<code>hwc.CallbackSet ()</code>	Use for event handlers that are asynchronous.	<code>Callbacks.js</code>
<code>hwc.CertificateStore</code>	Create a user interface in HTML and JavaScript that uses X.509 certificates as the Hybrid App credentials.	<code>Certificate.js</code>
<code>hwc.ConnectionSettings</code>	The JavaScript class for the Hybrid Web Container connection settings manages the connection between applications and the server.	<code>hwc-api.js</code>
<code>hwc.CustomIcon</code>	The JavaScript class for the Hybrid Web Container custom icon, lists custom icons.	<code>hwc-api.js</code>
<code>hwc.e2eTrace</code>	Allows for an end to end trace of data communication from the client to the back-end.	<code>hwc-api.js</code>
<code>hwc.getExternalResource</code>	Access resources on external HTTP servers.	<code>ExternalResource.js</code>
<code>hwc.getCurrentLocale</code>	The date/time functions allow you to extract and format the date and time for the Hybrid App.	<code>Timezone.js</code>
<code>hwc.getPicture</code>	Provides access to the device's default camera application or device's photo library for retrieving a picture asynchronously.	<code>Camera.js</code>
<code>hwc.HybridApp</code>	Javascript class for the Hybrid App object. Lists installed Hybrid Apps.	<code>hwc-api.js</code>

Class	Description	Defined in
<code>hwc.LogEntry</code>	Javascript class for LogEntry object.	<code>hwc-api.js</code>
<code>hwc.MediaCache</code>	Used within the JavaScript to wrap the source of an image element. Fetches media content from a cache or the server using a URI.	<code>hwc-api.js</code>
<code>hwc.MenuItemCollection</code>	Represents a collection of menu items.	<code>hwc-comms.js</code>
<code>hwc.Message</code>	This is the class to encapsulate an incoming message object. When a new message arrives, a notification is sent to users through custom code.	<code>hwc-api.js</code>
<code>hwc.MessageFilter</code>	This is the class to encapsulate a filter for messages.	<code>hwc-api.js</code>
<code>hwc.perf</code>	The performance library allows you to instrument your application code and collect performance counters when executing the application on the device. Results are reported in a log file on the SD-card (BlackBerry and Android), or in the sandbox (iOS). The results can also be read in the domain log by calling <code>GetTrace</code> for the application connection in SAP Control Center.	<code>hwc-api.js</code>
<code>hwc.SUPStorage</code>	Constructs a new storage area identified by a storage key.	<code>SUPStorage.js</code>
<code>Resources</code>	Access localized string resources.	<code>Resources.js</code>

Hybrid App Framework JavaScript APIs

The files where the Hybrid App framework JavaScript APIs are defined are located in `<SMP_HOME>\MobileSDK<version>\HybridApp\API\AppFramework`.

Class	Description	Defined in
<code>doOnlineRequest</code>	Allows an operation or object query to be invoked.	<code>API.js</code>
<code>MessageValue</code>	Message value object that stores a key-value pair from a message sent to or from the server and the Hybrid App.	<code>WorkflowMessage.js</code>
<code>MessageValueCollection</code>	Message value collection object that stores a container node from a message sent to or from the server and the Hybrid App.	<code>WorkflowMessage.js</code>
<code>WorkflowMessage</code>	Access the Hybrid App message data functions.	<code>WorkflowMessage.js</code>

MBO Access JavaScript API Samples

This section shows some sample JavaScript APIs that access MBOs.

Calling a Create Function

1. Create a JavaScript object, in this case, `Department`.

```
var dept1 = new Department();
```

2. Set the values for all the fields. The fields names map to the `Department` MBO create operation's parameter name.

```
dept1.dept_id = "800";
dept1.dept_name="Dept";
dept1.dept_head_id="888";
```

3. Call the create online request function.

```
department_create_onlineRequest(dept1,
    "supusername=supAdmin&suppassword=s3pAdmin",
    function() { alert("error occurred");});
```

4. For an online request, you should implement the `hwc.processDataMessage` function, for example:

```
hwc.processDataMessage = function
processDataMessage(incomingWorkflowMessage, noUI, loading,
fromActivationFlow, dataType) {
    if
    ( (incomingWorkflowMessage.indexOf("<XmlWidgetMessage>") === 0)
```

```
||
(incomingWorkflowMessage.indexOf("<XmlWorkflowMessage>") === 0)
|| (incomingWorkflowMessage.indexOf("<M>")
=== 0)) {
    var workflowMessage = new
WorkflowMessage(incomingWorkflowMessage);

if ( workflowMessage.getRequestAction() ==
Department.createAction ){
    alert("Department id=" +
workflowMessage.getValues().getData('Department_create_dept_id_pa
ramKey').getValue() + " has been created!");
}else if ( workflowMessage.getRequestAction()
==Sales_order.findAllAction){
    alert("Return Item count =" +
workflowMessage.getValues().getData('Sales_order').value.length )
;
//By default database it should return 54 items.
    }
}else{
    alert("TODO: Please fix me,
incomingWorkflowMessage="+ incomingWorkflowMessage);
}
}
```

Calling an Update Function With Old Arguments

1. Set old arguments values:

```
var oldDep = new Department();
oldDep.dept_id = "800";
oldDep.dept_name="Dept";
oldDep.dept_head_id="888";
```

2. Set the new values:

```
var newDep = new Department();
newDep.dept_id = "800";
newDep.dept_name="DeptUpdated";
newDep.dept_head_id="777";
```

3. Call the update submit function:

```
department_update_submit(newDep, oldDep, "", true );
```

Passing a Personalization Key Value

1. Create Sales_order object:

```
var sales_order = new Sales_order();
```

2. Set the onload personalization key value:

```
sales_order.pks.put(Sales_rep_PK_pkKey, "667");
```

3. Call the findAll online request:

```
sales_order_findAll( sales_order , "", function() {});
}
```

4. In the process workflowMessage function, to process incoming message, add:

```
hwc.processDataMessage=function processDataMessag
(incomingWorkflowMessage, noUI, loading, fromActivationFlow,
dataType) {

    if
    ( (incomingWorkflowMessage.indexOf("<XmlWidgetMessage>") === 0)
    ||
    (incomingWorkflowMessage.indexOf("<XmlWorkflowMessage>") === 0)
    || (incomingWorkflowMessage.indexOf("<M>")
    === 0)) {

        var workflowMessage = new
WorkflowMessage(incomingWorkflowMessage);
if ( workflowMessage.getRequestAction() ==
Sales_order.findAllAction){
            alert("Return Item count =" +
workflowMessage.getValues().getData('Sales_order').value.length )
;    //By default database it should return 54 items.
        }
    }else{

        alert("TODO: Please fix me,
incomingWorkflowMessage="+ incomingWorkflowMessage);
    }

}
```

Calling a Create Function on MBOs With a One to Many Relationship

1. Create a new Department:

```
var dep = new Department();
dep.dept_id="2";
dep.dept_name="My Dep";
dep.dept_head_id="1";
```

2. Create a new employee:

```
var empl = new Employee();
empl.emp_id = "1";
empl.manager_id = "2";
empl.emp_fname = "Yan";
empl.emp_lname= "Gong";
empl.street = "King Street";
empl.city="Waterloo";
empl.state = "ON";
empl.zip_code = "n2v3l4";
empl.phone="518-8836863";
empl.status="A";
empl.ss_number="024601768"
empl.salary = "324234";
empl.start_date="1996-12-30";
empl.termination_date = "1999-12-20";
empl.birth_date = "1956-12-20";
empl.bene_health_ins = "Y";
empl.bene_life_ins = "Y";
```

```
emp1.bene_day_care="Y";
emp1.sex="F";
```

3. Create a second employee:

```
var emp2 = new Employee();
emp2.emp_id = "2";
emp2.manager_id = "2";
emp2.emp_fname ="Yan2";
emp2.emp_lname= "Gong2";
emp2.street ="King Street";
emp2.city="Waterloo";
emp2.state ="ON";
emp2.zip_code ="n2v3l4";
emp2.phone="518-8836863";
emp2.status="A";
emp2.ss_number="024601768"
emp2.salary ="324234";
emp2.start_date="1996-12-30";
emp2.termination_date ="1999-12-20";
emp2.birth_date ="1956-12-20";
emp2.bene_health_ins ="Y";
emp2.bene_life_ins ="Y";
emp2.bene_day_care="Y";
emp2.sex="F";
```

4. Add the two employees to Department:

```
dep.Employee.push( emp1 );
dep.Employee.push( emp2 );
```

5. Call department create online request, it would create a new department and two new employees entries in the database:

```
department_create_onlineRequest(dep,
                                "",      function() {});
```

Calling a Delete Function on MBOs With a One to Many Relationship

To delete an MBO and its children, you need to find the MBO instance online request and, from the processDataMessage function, after the online request, you need to find each child's surrogate key value from the incoming message, create a child JavaScript instance, then add the child JavaScript instance to the parent JavaScript instance. Subsequently, when the delete function is called on the parent instance, the children are also deleted. The details of this are shown in this example in **bold** font.

1. Call the department_findByPrimaryKey online request to find the department instance:

```
function deleteDepartment() {

    var dep = new Department();
    dep.dept_id="2";

    alert("before delete Deeparatment and its children
Employee, we need to call findByPrimaryKey first.")
    department_findByPrimaryKey( dep, "" , function(error)
{alert(error)});
```

```
    }
```

2. In the `processDataMessage` function, find the `surrogatekey` value for each `Employee` and create `Employee` instance and add it to department instance:

```
hwc.processDataMessage = function
processDataMessage(incomingWorkflowMessage, noUI, loading,
fromActivationFlow, dataType) {

    alert( "incomingMessage="+ incomingWorkflowMessage);
    if
    ( (incomingWorkflowMessage.indexOf("<XmlWidgetMessage>") === 0)
      ||
    (incomingWorkflowMessage.indexOf("<XmlWorkflowMessage>") === 0)
      || (incomingWorkflowMessage.indexOf("<M>") ===
0)) {

        var workflowMessage = new
WorkflowMessage(incomingWorkflowMessage);

        if ( workflowMessage.getRequestAction() ===
Department.findByPrimaryKeyAction){

            var employees =
workflowMessage.getValues().getData('Department_employees').value
;

            if
            ( workflowMessage.getValues().getData('Department_dept_id_attribK
ey').getValue() == '2'){
                var dep = new Department();
                dep.dept_id="2";

                for( var i = 0; i < employees.length ; i++ ) {
                    var emp = new Employee();
                    emp.emp_id =
employees[i].getData('Employee_emp_id_attribKey').getValue();
                    //set surrogateKey for employ
                    emp._surrogateKey
=employees[i].getData('_surrogateKey').getValue();
                    dep.Employee.push( emp );
                    dep.OldValue_Employee.push( emp );
                }
            }
        }
    }
```

3. Call `department_delete_onlineRequest` to delete the department and all of its children:

```
department_delete_onlineRequest( dep, "", function( error)
{ alert(error)});

    }

    .....
}
```

MediaCache Examples

```
var resourceUrl = "http://someserver/someimage.jpg";
document.write("<img src=\"" + MediaCache.getUrl(resourceUrl,
hwc.MediaCache.Policy.CACHE_FIRST) + "\" />");

var oImg=document.createElement("img");
oImg.setAttribute('src', MediaCache.getUrl('http://someserver/
someimage.jpg'));
document.body.appendChild(oImg);
```

Null Value Support

Null data values are represented in `MessageValue` objects, belonging to a `MessageValue` collection that is created from the data message sent by the server.

Note: Null data values are not supported on the Windows Mobile platform.

This document refers to example HTML. You can see the example HTML by downloading the *hybridapp_null_value.zip* file and extracting the *hybridapp_null_value.html* file.

In the example, `MessageValueCollection` is referenced by `var values = myDataMessage.getValues();` and `myDataMessage` is created in the `onHybridAppLoad` method.

A specific `MessageValue` is referenced by `values.getData(string key)`. If you use the Hybrid App designer, the IDE manages keys for you but with JavaScript API, you have to implement key management in the code yourself.

This example follows the IDE style of giving each control an ID and using that as the key for data that will be used in that control:

```
<input class="right" type="number"
id="Nullvaluetest_int_value2_attribKey"
smp_allows_null="true" smp_valuechanged="false"
onchange="inputChanged(this)"/>
```

So if you want the `MessageValue` object corresponding to that control:

```
var value = values.getData("Nullvaluetest_int_value2_attribKey");
```

Once you have the `MessageValue` object you can see if it is null with:

```
var isNull = value.getNullAttribute();
```

Null Values and HTML

HTML usually puts an empty string into a control if it is assigned a null value. If the control is not changed but you get the data from it for its `MessageValue` object, the `MessageValue` object will have an empty string as its value instead of `NULL`. This means the `NullAttribute` is not set properly unless you set it yourself.

When using null values, keep in mind that the contents of the control do not tell you whether it should be null. This can cause bad data on the server. Putting an empty string into a number type `MessageValue` can throw a formatting exception on the server, so when using JavaScript API, you are responsible for maintaining null values.

The Sample HTML

This section references the `hybridapp_null_value.html` file to show examples of how to implement null values.

- **Recognizing NULL values** – The example uses the same techniques as an Hybrid App generated with the designer to keep track of data values, keys, controls and null-ness.

Controls that allow null have a special attribute that identifies it is okay to be NULL:

```
<input class="right" type="number"
id="Nullvaluetest_int_value2_attribKey"
smp_allows_null="true" smp_valuechanged="false"
onchange="inputChanged(this)"/>
```

This example processes the incoming data message and checks control attributes for null friendliness and issues an alert message for a null value in the wrong place.

```
hwc.processDataMessage = function(incomingDataMessageValue, noUI,
loading, fromActivationFlow, dataType)
```

- **Handling input to NullValue controls** – This example uses an event handler to recognize user input:

```
<input class="right" type="number"
id="Nullvaluetest_int_value2_attribKey"
smp_allows_null="true" smp_valuechanged="false"
onchange="inputChanged(this)"/>
```

`inputChanged` uses another special attribute to indicate that the user has put something in the control and it is no longer null.

- **Setting a value to NULL** – In the HTML example, `setKeyValueNull` and `setControlValueNull` show how to set a value to null while managing the control attributes and the `MessageValue` null attribute.
- **Sending data to the server** – In the example HTML, `doUpdate` uses the `getUpdatedValue` method to set the right value in the `newNVT` object. `getUpdatedValue` checks the control attributes and the `MessageValue` null attribute to decide what to send to the server.
- **Creating data with null values** – In the example HTML, `doCreate` and `doCreate2` show two ways of creating a record with null values.

Calling the Hybrid Web Container

It is easiest to learn how to call the Hybrid Web Container by examining the `API.js` and `Utils.js` files, which are located in `<SMP_HOME>\MobileSDK<version>\HybridApp\API\AppFramework`.

Making calls to the Hybrid Web Container is platform-dependent, as shown in this example:

```
if (isWindowsMobile()) {
    var xmlhttp = getXMLHttpRequest();
```

```
        xmlhttp.open("POST", "/sup.amp?
querytype=setscreentitle&version=2.0", false);
        xmlhttp.send("title=" + encodeURIComponent(screenTitle));
    }
    else if (isIOS()) {
        var xmlhttpReq = getXMLHttpRequest();
        xmlhttpReq.open("GET", "http://localhost/sup.amp?
querytype=setscreentitle&version=2.0&title=" +
encodeURIComponent(screenTitle), true);
        xmlhttpReq.send("");
    }
    else if (isAndroid()) {
        var request = "http://localhost/sup.amp?
querytype=setscreentitle&version=2.0&title=" +
encodeURIComponent(screenTitle);
        _WorkflowContainer.getData(request);
    }
    else { //must be BlackBerry
        var xmlhttp = getXMLHttpRequest();
        xmlhttp.open("POST", "http://localhost/sup.amp?
querytype=setscreentitle&version=2.0", false);
        xmlhttp.send("title=" + encodeURIComponent(screenTitle));
    }
}
```

From a high-level perspective, these are the query types used for calling the Hybrid Web Container.

setscreentitle

Sets the native screen title on the Hybrid Web Container.

close

Closes the native Hybrid Web Container (Windows Mobile only).

addMenuItem

Adds a single menu item to the Hybrid Web Container.

removeallmenuitems

Removes all the menu items from the Hybrid Web Container.

clearrequestcache

Clears the entire Online Request cache for the current Hybrid App.

clearrequestcacheitem

Clears a single Online Request cache entry for the current Hybrid App.

logtoworkflow

Logs a message to the AMPHostLog.txt (mocalog.txt for iOS) on the device. You can retrieve this log file remotely from SAP Control Center.

showcertpicker

Shows a native platform certificate picker on the device for selecting certificate credentials.

showInBrowser

On iOS, this function shows the URL in the Hybrid Web Container in a separate browser instance. On all other platforms, this launches the native Web browser in another window with the given URL.

showattachment

Using third party file viewers, this function displays an attachment that has previously been downloaded using the `downloadattachment` querytype in a separate window.

Note: On iOS, the attachment is shown within the Hybrid Web Container.

showlocalattachment

Using third party file viewers, this function displays an attachment that was included as part of the Hybrid App .zip package, in a separate window.

Note: On iOS, the attachment is shown within the Hybrid Web Container.

rmi

This function executes an online request to the SAP Mobile Server synchronously, in other words, a network connection must be available. This can indicate results should be cached for future access (in which case a network connection does not need to be available).

downloadattachment

Requests an attachment to be downloaded from the SAP Mobile Server through an object query. A network connection is required for this operation. This operation occurs asynchronously, and the calling JavaScript is notified when it is complete.

submit

Submits the current `MessageValueCollection` to the SAP Mobile Server for processing by the server plug-in. This operation occurs asynchronously. If a network connection is not available when this operation is performed, the request is queued up and executed the next time a network connection is available.

alert

Shows a message box in native code (iOS and Android platforms only).

loadtransformdata

Requests the Hybrid Web Container for the transform data (the contents of the e-mail message) for the current message.

addallmenuitems

Instructs the Hybrid Web Container to add the supplied list of menu items.

formredirect

Notifies the Hybrid Web Container that a screen navigation is occurring, and to update credentials in the credentials cache, if required.

AttachmentViewer and Image Limitations

There are some limitations on the size of the attachments and images that you can include as part of the Hybrid App message.

These limitations vary by platform.

Platform	Size Limit
iOS	Large attachments can produce longer processing times.
Android	Large attachments can produce longer processing times. There is a 1MB limit for attachments on Android devices.
Windows Mobile	The maximum size of a JavaScript variable for Windows Mobile is 2MB, which allows for more memory. Warning messages are shown if the script continues for a long time, which can cause the memory to run out.
BlackBerry 5.0 and BlackBerry 6.0	On BlackBerry 5.0, the maximum size of a JavaScript variable is 500KB and on BlackBerry 6.0 and later, the maximum size of a JavaScript variable is 2MB. The maximum size must be larger than the attachment and the rest of the Hybrid App message. If the attachment is Base64-encoded, also allow for an increase in the attachment size.

Note: When accessing very large binary (image) data in the mobile business object associated with the Hybrid App, ensure that the attribute set in the mobile business object is a **BigBinary** datatype, rather than Binary.

Package Hybrid Apps

Package the files for the Hybrid App so that you can deploy them to the server.

Packaging Hybrid Apps Using the Packaging Tool

Use the packaging tool to package existing files into a Hybrid App package.

1. Navigate to <SMP_HOME>\MobileSDK22\HybridApp\PackagingTool and double-click the `packagingtool.bat` file if you are using a 32-bit JDK, or `packagingtool64.bat` if you are using a 64-bit JDK.
2. Click **Browse** to enter the filepath for the output directory where your projects are located, and click **OK**.
All of the projects stored in the output directory you set appear in the Project Explorer list box.
3. (Optional) Select a project to see the details of the project in the right pane. You can make changes to any of the General Information properties and click **Save**.
4. (Optional) To create a new project:
 - a) Click **New** at the bottom of the Project Explorer list box.
 - b) Enter a project name.
 - c) Click **Browse** to select a folder for the Web application folder from the local hard disk.

The Web root folder is the location of your HTML files, typically, with any dependent HTML, JavaScript, CSS, images, and so on, files being in the same folder or subfolders. The `WorkflowClient.xml` file should also be in the Web application folder.

Note: The Web application folder cannot be a subfolder of the workspace, and the workspace folder cannot be a subfolder of the Web application folder.

- d) Click **OK**.
The new project name is added to the Project Explorer, and a project file is created in the workspace folder with the `.pkgproj` extension. The project will have a separate folder under the workspace to store all temporary files for deployment.
5. (Optional) To remove a project from Project Explorer, select the project to remove and click **Delete** at the bottom of the Project Explorer list box.
6. Set the configuration information for the project in the General Information tab.
 - Module name – the name of the Hybrid App on the server. The default value is the project name. This is required.
 - Module version – this can be any number. The default value is 1. It is required.
 - Module description – (optional) enter description text.
 - Display name – (optional) the display name.
 - Client icon – the default value is 48. It is required.
 - MBO package name – if the Hybrid App uses MBOs, input the MBO package name.
 - MBO package version – enter the version for the MBO package.

- Invokable on client – a boolean value to determine whether the Hybrid App can be invoked from the client. The default value is true.
 - Processed Messages
 - Mark as read – the default value is false.
 - Delete – the default value is true.
 - Cache key – (optional) the key to represent the cache.
 - Activation key – (optional) define the key to use.
 - Shared storage key – (optional) enter the shared storage key.
 - SAP Mobile Platform server information – the manifest.xml file may require hard-coded credentials for logging in to SAP Mobile Server.
 - User name – enter the user name for logging into SAP Mobile Server.
 - Simple password – enter the password for logging into SAP Mobile Server.
 - Certificate – enter the certificate information for logging into SAP Mobile Server.
7. Click the applicable platform tab to choose files for packaging.
- Five platforms are available: Android, BlackBerry 5, BlackBerry 6, iOS, and Windows Mobile 6. For each platform, you can choose whether to include the specific platform in the package, the files needed for the platform, the HTML files for the the platform, and the start screen to show for this platform.
- The start screen is the screen to show by default for the selected platform. The html (or htm) file in the HTML File for the Start Page textbox is parsed and all screens are then listed. If the file is not an html file or there is no screen defined in the file, the start screen textbox is empty.
8. (Optional) Click the **Matching Rules** tab to add matching rules.
- Matching rules describe the collection of rules that are used to determine if a given server notification will be sent to the application for processing. Each matching rule describes the field to test (for example, Subject), and the regular expression to test against for matches.
9. (Optional) Click **Custom Icon** to add a custom icon for the Hybrid App package.
- When you add a custom icon, the manifest.xml file is updated when you generate the package.
10. (Optional) Click **Client Variables** to add client variables for data that is associated with a particular client and application and that must be saved between user sessions.
11. Click **Generate**.
- Configuration files are created and packaged in a ZIP file and placed in the output directory you specified.

Refreshing the Packaging Tool Treeview

Refresh the treeview to reflect the latest changes to the package.

There are several ways to refresh the treeview.

- Exit the packaging tool and restart it. All the new files appear in the treeview automatically.

- Switch to another project, then switch back.
- Click the **Support <xxx> platform** checkbox, uncheck it and then check it. When you set the **Support <xxx> platform** checkbox to true, the treeview is refreshed to get the latest files from the Web app folder.

Packaging Hybrid Apps Manually

While using the packaging tool is the easiest way to package Hybrid Apps, it is also possible to create a Hybrid App package without the tool.

Hybrid AppPackage Files

To build a Hybrid App package manually, you should first familiarize yourself with its contents.

This section describes the contents of the Hybrid App package—which files are required, and what the contents of those files should be. Particular attention is paid to the contents of the `manifest.xml` and `WorkflowClient` XML files, along with the Web application files (HTML, JavaScript, CSS), most specifically the public API functions available to you.

The Web Application Files

A Hybrid App package contains Web application files.

When developing a Hybrid App package manually:

- Include HTML files that follow the same general pattern as the files generated when using the Hybrid App Designer to generate the Hybrid App package.
- Use the `API.js`, `Callbacks.js`, `Camera.js`, `Certificate.js`, `ExternalResource.js`, `SUPStorage.js`, and `Timezone.js` files to communicate with the Hybrid Web Container. These files are in the `<SMP_HOME>\MobileSDK22\HybridApp\API\Container` and `<SMP_HOME>\MobileSDK22\HybridApp\API\AppFramework` directories.
- Use `WorkflowMessage.js` to view and manipulate the Hybrid App messages. This file is located in `<SMP_HOME>\MobileSDK22\HybridApp\API\AppFramework`

HTML Format

This is a commonly used HTML format.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <meta name="HandheldFriendly" content="True" />
    <meta http-equiv="PRAGMA" content="NO-CACHE" />
    <link rel="stylesheet" href="css/MyStylesheet.css"
type="text/css" />
    [...]
    <script src="js/API.js"></script>
    <script src="js/Utils.js"></script>
```

```

        <script src="js/WorkflowMessage.js"></script>
        <script src="js/MyJavaScript.js"></script>
        [...]
        <script>
[...]
```

</script>
 </head>
 <body onload="onHybridAppLoad();">
 <div id=Screen1KeyScreenDiv" smp_screen_title="Screen1Title"
 style="display: none"
 smp_menuitems="NativeMenu1Key,NativeMenu1DisplayName,NativeMenu2Key
 ,NativeMenu2DisplayName" smp_okaction="myOKAction()">
 [...]

 <form style="margin: 0px;" name="Screen1KeyForm"
 id="Screen1KeyForm" onSubmit="return false;" autocomplete="on">
 [...]

Manifest.xml File

The manifest.xml file describes how the contents of the Hybrid App package .zip file are organized.

This file must reside at the root of the Hybrid App ZIP package. This shows the outline of what the manifest.xml file contains.

Manifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<Manifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AMPManifest.xsd">
  <ModuleName>...</ModuleName>
  <ModuleVersion>...</ModuleVersion>
  <ModuleDesc>...</ModuleDesc>
  <ModuleDisplayName>...</ModuleDisplayName>
  <ClientIconIndex>...</ClientIconIndex>
  <InvokeOnClient>...</InvokeOnClient>
  <PersistAppDomain>...</PersistAppDomain>
  <MarkProcessedMessages>...</MarkProcessedMessages>
  <DeleteProcessedMessages>...</DeleteProcessedMessages>
  <ProcessUpdates>...</ProcessUpdates>
  <CredentialsCache>...</CredentialsCache>
  <RequiresActivation>...</RequiresActivation>
  < SharedStorage key> ... </ SharedStorage >

```



```

<TransformPlugin>
<File shared="true">WorkflowClient.dll</File>
<Class>Sybase.UnwiredPlatform.WorkflowClient.Transformer</Class>
</TransformPlugin>
- <ResponsePlugin>
<File shared="true">WorkflowClient.dll</File>
<Class>Sybase.UnwiredPlatform.WorkflowClient.Responder</Class>
</ResponsePlugin>

<ClientWorkflows>
  <WindowsMobileProfessional>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </WindowsMobileProfessional>
  <BlackBerry>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </BlackBerry>
  <BlackBerry6>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </BlackBerry6>
  <Android>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </Android>
  <iPhone>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </iPhone>

```

```
</ClientWorkflows>

<ContextVariables>
  <ContextVariable>
    <Name>...</Name>
    <Value>...</Value>
    <Certificate>...</Certificate>
    <Password>...</Password>
  </ContextVariable>
</ContextVariables>

<Metadata version="1">
  <Item>
    <Name>Key1</Name>
    <Value>Value1</Value>
  </Item>
  <Item>
    <Name>Key2</Name>
    <Value>Value2</Value>
  </Item>
</Metadata>
<MatchRules>
  <SubjectRegExp>...</SubjectRegExp>
  <ToRegExp>...</ToRegExp>
  <FromRegExp>...</FromRegExp>
  <CCRegExp>...</CCRegExp>
  <BodyRegExp>...</BodyRegExp>
</MatchRules>
</Manifest>
```

ModuleName

```
<ModuleName>SampleActivitiesModule</ModuleName>
```

The `ModuleName` defines the name of the Hybrid App package.

ModuleVersion

```
<ModuleVersion>2</ModuleVersion>
```

The `ModuleVersion` defines the version of the Hybrid App package.

ModuleDesc

```
<ModuleDesc>AMP Sample - Activities Collection</ModuleDesc>
```

The `ModuleDesc` provides a short description of the Hybrid App package.

ModuleDisplayName

```
<ModuleDisplayName>Activities Sample</ModuleDisplayName>
```

The name of the Hybrid App package that is displayed to the user in the Hybrid App list on the device for Hybrid Apps that are client-invoked. When the Hybrid App package is deployed, you can override the `DisplayName` specified here with one of your own choosing.

ClientIconIndex

```
<ClientIconIndex>35</ClientIconIndex>
```

The index of the icon associated with the Hybrid App package. This icon is shown beside the e-mail message in the device's Inbox listing instead of the regular e-mail icon. When the Hybrid App package is deployed, you can override the icon that is specified here with one of your own choosing.

InvokeOnClient

```
<InvokeOnClient>1</InvokeOnClient>
```

Specifies whether this Hybrid App can be used without an associated e-mail. 1 = true, 0 = false. If 1 is specified, the Hybrid App is shown in the Hybrid App list on the device and can be used without the context of an e-mail message.

PersistAppDomain

```
<PersistAppDomain>1</PersistAppDomain>
```

States whether this Hybrid App uses a persistent application domain when the .NET assembly plugin is loaded. 1 = true, 0 = false. By default, it is set to false, meaning an application domain is created and removed every time the plugin is loaded.

MarkProcessedMessages

```
<MarkProcessedMessages>1</MarkProcessedMessages>
```

Indicates whether a Hybrid App message shows a visual indication in the Inbox after it has been processed (1 = true, 0 = false).

Note: When a Hybrid App message shows a visual indication that it has been processed, the visual indication disappears if the device is re-registered, or if the device user performs a Refresh All Data action.

DeleteProcessedMessages

```
<DeleteProcessedMessages>1</DeleteProcessedMessages>
```

Indicates whether a Hybrid App message is deleted from the mobile device's Inbox after it has been processed (1 = true, 0 = false).

Note: You cannot set both `DeleteProcessedMessages` and `MarkProcessedMessages` to true (1). To set `MarkProcessedMessages` to true, you must set `DeleteProcessedMessages` to false (0) as shown:

```
<MarkProcessedMessages>1</MarkProcessedMessages>
<DeleteProcessedMessages>0</DeleteProcessedMessages>
```

ProcessUpdates

```
<ProcessUpdates>1</ProcessUpdates>
```

Indicates whether Hybrid App messages associated with this Hybrid App package that are already delivered to the device can be updated from the server with modified content. (1 = true, 0 = false). By default, this is set to false (0).

CredentialsCache

```
<CredentialsCache key="activity_credentials">1</  
CredentialsCache>
```

Specifies whether a Hybrid App requires credentials (1 = true, 0 = false). Different Hybrid Apps can specify different credentials keys. Hybrid Apps with the same credentials key share that set of credentials. In the case of shared credentials, they are requested only once by the Hybrid App that is launched first.

RequiresActivation

```
<RequiresActivation key="shared_credentials_key">1</  
RequiresActivation>
```

Specifies whether a Hybrid App requires activation (1 = true, 0 = false). If set to true, the screen defined in the `ActivationScreen` tag is displayed the very first time the Hybrid App is launched, before the default screen is displayed.

If the Activation Screen contains credentials controls (and the Hybrid App requires credentials), the values are updated to the Credentials Cache automatically, without further prompting, with the specified Credentials Screen.

Different Hybrid Apps can specify different activation keys. Hybrid Apps with the same activation key share their activation status. For example, if Hybrid App A and Hybrid App B both specify an activation key of AB (using the key attribute on the `RequiresActivation` tag), when Hybrid App A gets activated, it also activates Hybrid App B so that when Hybrid App B is invoked for the very first time, its activation screen is not seen; it goes directly to the default screen.

TransformPlugin

```
<TransformPlugin> <File/> <Class/> </TransformPlugin>
```

(Optional) If this is defined, the `ResponsePlugin` tag must also be defined. Describes the server module implemented as a .NET assembly that implements the `IMailProcessor` interface. This module is responsible for processing the intercepted e-mail message before it gets delivered to the device.

Inner tags

```
<File shared="true">WorkflowClient.dll</File>
```

 The path, including the filename of the assembly that implements the `IMailProcessor` interface. The path is relative to

the Hybrid App ZIP package. If the shared property is present and set to true, the DLL is located in the <UnwiredPlatform_InstallDir>\Servers\MessagingServer\bin folder (installed by an external process) and all Hybrid Apps using that DLL will use the same version of the DLL. If the shared property is not present, or is present and is set to false, each Hybrid App will use its own version of that DLL in the Hybrid App's own folder.

<Class>Sybase.UnwiredPlatform.WorkflowClient.Transformer</Class> The .NET Type in the assembly that implements the IMailProcessor interface.

ResponsePlugin

<ResponsePlugin> <File/> <Class/> </ResponsePlugin>

(Optional) If this is defined, the TransformPlugin tag must also be defined. Describes the server module implemented as a .NET assembly that implements the IResponseProcessor interface. This module is responsible for processing the response from the device.

Inner tags

<File shared="true">WorkflowClient.dll</File> The path, including the filename, of the assembly that implements the IResponseProcessor interface. The path is relative to the Hybrid App .zip package. If the shared property is present and set to true, the DLL is expected to be located in the <UnwiredPlatform_InstallDir>\Servers\MessagingServer\bin folder (installed by an external process), and all Hybrid Apps using that DLL will use the same version of the DLL. If the shared property is not present, or is present and set to false, each Hybrid App will use its own version of that DLL in the Hybrid App's own folder.

<Class>Sybase.UnwiredPlatform.WorkflowClient.Responder</Class> The .NET Type in the assembly that implements the IResponseProcessor interface.

ClientWorkflows

```
<ClientWorkflows>
  <WindowsMobileProfessional>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </WindowsMobileProfessional>
  <BlackBerry>
    <HTMLWorkflow>
      <File>...</File>
      <HtmlFiles>
        <HtmlFile>...</HtmlFile>
        <HtmlFile>...</HtmlFile>
      </HtmlFiles>
    </HTMLWorkflow>
  </BlackBerry>
```

```
<BlackBerry6>
  <HTMLWorkflow>
    <File>...</File>
    <HtmlFiles>
      <HtmlFile>...</HtmlFile>
      <HtmlFile>...</HtmlFile>
    </HtmlFiles>
  </HTMLWorkflow>
</BlackBerry6>
<iPhone>
  <HTMLWorkflow>
    <File>...</File>
    <HtmlFiles>
      <HtmlFile>...</HtmlFile>
      <HtmlFile>...</HtmlFile>
    </HtmlFiles>
  </HTMLWorkflow>
</iPhone>
<Android>
  <HTMLWorkflow>
    <File>...</File>
    <HtmlFiles>
      <HtmlFile>...</HtmlFile>
      <HtmlFile>...</HtmlFile>
    </HtmlFiles>
  </HTMLWorkflow>
</Android>
</ClientWorkflows>
```

This section of the `manifest.xml` file describes the supported device platforms for the Hybrid App and the corresponding client module to use for each platform.

Inner tags

- `<WindowsMobileProfessional>...</WindowsMobileProfessional>` – Windows Mobile Professional device support
- `<iPhone>...</iPhone>` – iOS device support
- `<BlackBerry>...</BlackBerry>` – BlackBerry 5.0 device support
- `<BlackBerry6>...</BlackBerry6>` – BlackBerry 6.0 device support
- `<Android>...</Android>` – Android device support

`<File>...</File>`

Contains a reference to an XML file. That XML file should have contents similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<widget>
  <screens src="html/myAndroidHybridApp.html"
default="Start_Screen">
    <screen key="html/myAndroidHybridApp.html">
      </screen>
    </screens>
</widget>
```

The referenced HTML file must be present in the list of `HtmlFiles` tags that follow and must also be present in the Hybrid App .zip package.

```
<HtmlFile>...</HtmlFile>
```

Indicates that the named file (`html/js/API.js`, `html/myAndroidHybridApp.html`) will be used on the specified platform. The referenced file must be present in the Hybrid App .zip package.

ContextVariables

```
<ContextVariables>...</ContextVariables>
```

Describes the collection of context variables that will be made available to the methods in the `IMailProcessor` and `IResponseProcessor` interfaces. When the Hybrid App package is deployed by the administrator, the Display Name that is specified here can be overridden with one of their own choosing.

```
<ContextVariables >
<ContextVariable>
<Name/>
<Value/>
<Certificate/>
<Password/>
</ContextVariable>
```

Describes a context variable that will be made available to the methods in the `IMailProcessor` and `IResponseProcessor` interfaces. When administrators deploy a Hybrid App package, they have the ability to override the value of the context variable that is specified here.

It is good practice for developers of Hybrid Apps to provide sufficient documentation so that administrators can knowledgeably edit a context variable's value as necessary. Context variables are a good place to store configuration information that will likely change between development and production environments.

Inner tags

`<Name>OutputFolder</Name>` The name of the context variable. This is the key used to retrieve the value of the context variable in the methods defined in the `IMailProcessor` and `IResponseProcessor` interface.

Note: The value of the `<Name>` tag supports single-byte characters only.

`<Value>C:\ActivitiesSampleOutput</Value>` The value of the context variable. When administrators deploy a Hybrid App, they have the ability to override the value of the context variable that is specified here.

Note: The value of the `<Value>` tag supports single-byte, double-byte, or both, characters.

`<Certificate>>false</Certificate>` Indicates whether this context variable is a Base64 string representation of an X.509 certificate. If this value is set to true, SAP Control Center displays a dialog specific to selecting an X.509 certificate.

`<Password>false</Password>` Indicates whether this context variable is a password. If set to true, the value is displayed as asterisks in the SAP Control Center console.

Client Variables

You can define client variables on the server side and retrieve it on the client side by using either native API or JavaScript API. In the JavaScript API, you can call the `hwc.getClientVariables(moduleid, version)` method to retrieve the client variables.

An optional metadata element in manifest.xml is used to specify clientvariables information. It has a version attribute of integer type to identify and keep track of metadata changes. You can set any positive integer value as the initial version. After the Hybrid App is deployed, each time the metadata gets updated, the version number is increased by one.

```
<Metadata version="1">
  <Item>
    <Name>Key1</Name>
    <Value>Value1</Value>
  </Item>
  <Item>
    <Name>Key2</Name>
    <Value>Value2</Value>
  </Item>
</Metadata>
```

You can update the client variables for a Hybrid App in SAP Control Center, and the change will be pushed to the already deployed clients. The client variables received on the client side are treated as read-only. The client cannot update the client variables.

Similar to server side Hybrid App context variables, client variables are stored as name/value pairs. Both name and value are string type, and the name is case sensitive. The maximum length of the client variable key name is 256 in ANSI code (not Unicode). Although the name is case sensitive, it cannot have the same item names that differ only by case. The metadata item key name cannot be an empty string. The object of a complex type needs to be serialized to string values to set the value.

Note: Due to a limitation on Windows Mobile platforms, the total length of all the client variables (keys and values) cannot exceed 2000 characters.

If the client side variables are updated, the change is applied the next time the Hybrid App is opened.

Similar to context variables, when the Hybrid App package is deployed in SAP Control Center with the option of "Replace," the updated client variables for the old Hybrid App package are not automatically passed to the new Hybrid App package.

MatchRules

```
<MatchRules>...</MatchRules>
```


Describes the collection of match rules that are used to determine if a message is sent to a TransformPlugin server module for processing. When administrators deploy a Hybrid App, they have the ability to Add, Delete, and override the Match Rules that are specified here.

<MatchRule>... </MatchRule> Describes a single match rule.

Note: The value of the <MatchRule> tag supports double-byte characters.

Inner tags

<SubjectRegExp>...</SubjectRegExp> The value to test for against the "Subject" line of a message.

<ToRegExp>...</ToRegExp> The value to test for against the "To" line of a message.

<FromRegExp>...</FromRegExp> The value to test for against the "From" line of a message.

<CCRegExp>...</CCRegExp> The value to test for against the "CC" line of a message.

<BodyRegExp>...</BodyRegExp> The value to test for against the <Body> text of a message.

WorkflowClient.xml File

The WorkflowClient.xml file contains metadata that specifies how to map the data in the Hybrid App message to and from calls to Mobile Business Object (MBO) operations and object queries.

WorkflowClient.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Workflow xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="WorkflowClient.xsd" >
  <Globals>
    <DefaultScreens activation="..." credentials="..." />
  </Globals>
  <Triggers>
    <Actions>
      <Action name="..." sourcescreen="..." targetscreen="..."
errorscreen="...">
        <Methods>
          <Method type="replay" mbo="..." package="..." >
            <InputBinding optype="..." opname="..."
generateOld="...">
              <Value sourceType="..." workflowKey="..." paramName="..."
mboType="..." />
              <Value sourceType="..." workflowKey="..."
relationshipName="..." mboType="list">
                <InputBinding optype="delete" opname="..." generateOld="...">
                  <Value sourceType="..." workflowKey="..." paramName="..."
attribName="..." mboType="..." />
                </InputBinding>
                <InputBinding optype="update" opname="..." generateOld="...">
                  <Value sourceType="..." workflowKey="..." paramName="..."
```

```

attribName="..." mboType="..."/>
    </InputBinding>
    <InputBinding optype="create" opname="..." generateOld="...">
        <Value sourceType="..." workflowKey="..." paramName="..."
attribName="..." mboType="..."/>
    </InputBinding>
    </Value>
    </InputBinding>
    <OutputBinding generateOld="...">
        <Mapping workflowKey="..." workflowType="..." attribName="..."
mboType="..."/>
    <Mapping workflowKey="..." workflowType="list"
mboType="list">
    <Mapping workflowKey="..." workflowType="..." attribName="..."
mboType="..."/>
    </Mapping>
    </OutputBinding>
    </Method>
    </Methods>
    </Action>
    </Actions>
    <Notifications>
        <Notification type="onEmailTriggered"
targetscreen="...">
            <Transformation>
                <Rule type="regex-extract" source="..." workflowKey="..."
workflowType="..." beforeMatch="..." afterMatch="..." format="..."/>
            </Transformation>
            <Methods>
                <Method name="..." type="..." mbo="..." package="...">
                    <InputBinding opname="..." optype="...">
                        <Value sourceType="..." workflowKey="..." paramName="..."
attribName="..." mboType="..."/>
                    </InputBinding>
                    <OutputBinding generateOld="...">
                        <Mapping workflowKey="..." workflowType="..." attribName="..."
mboType="..."/>
                    <Mapping workflowKey="..." workflowType="list"
mboType="list">
                        <Mapping workflowKey="..." workflowType="..."
attribName="..." mboType="..."/>
                    </Mapping>
                    </OutputBinding>
                </Method>
            </Methods>
        </Notification>
    </Notifications>
</Triggers>
</Workflow>

```

Globals

```

<Globals> <DefaultScreens activation="Introduction"
credentials="Authentication"/> </Globals>

```

Describes the global information for the Hybrid App metadata.

Inner tags

`<DefaultScreens activation="..." credentials="..." />` contains two optional attributes—activation and credentials—that allow you to specify the screens to use for activation and credential requests.

Triggers

`<Triggers> <Actions> ... </Actions> <Notifications> ... </Notifications> </Triggers>`

Describes the conditions under which MBO operations and/or object queries run and, where appropriate, what to return to the device.

Inner tags

`<Actions> ... </Actions>` Contains the description for one or more MBO operations and/or object queries to execute when an online request or submit action is received from the client.

`<Notifications> ... </Notifications>` Contains the description of, at most, one way to extract values from an incoming server notification, execute an MBO object query, and send that notification on to the device.

Action

`<Action name="Online_Request" sourcescreen="Reports_Create" targetscreen="OnReportsCreateSuccess" errorscreen="OnReportsCreateFailure"> ... </Action>`

Describes the conditions under which MBO operations and/or object queries run and, where appropriate, what to return to the device.

Table 1. Attributes

Attribute	Description
name	The name of the action, which typically corresponds to the key of the menuitem that invoked the action.
sourcescreen	The screen from where the action was invoked.
targetscreen	This attribute is optional. The screen to which the client will return, by default, if the MBO operation/object query succeeds. If left unspecified, the client application remains on the current screen. This attribute is applicable only to online request actions.

Attribute	Description
errorscreen	This attribute is optional. The screen to which the client will return, by default, if the MBO operation/object query fails. If left unspecified, the client application remains on the current screen. This attribute is applicable only to online request actions.
<ul style="list-style-type: none"> errorlogskey errorlogmessagekey errorlogmessageaslistkey 	The keys used to fill any error log messages.

Inner tags

`<Methods> ... </Methods>` Contains the description for one or more MBO operations and/or object queries to be executed when this online request or submit action is received from the client.

Method

`<Method type="replay" mbo="Reports" package="testReports:1.0"> ... </Method>`

Describes the conditions under which MBO operations and/or object queries run and, where appropriate, what to return to the device.

Table 2. Attributes

Attribute	Description
type	The type of method to invoke. For object queries, this must be search . For operations, it must be replay .
mbo	The name of the mobile business object (MBO).
package	The Hybrid App package name and version of the MBO, separated by a colon, for example, <code><package_name>:<mbo_version></code> .

Inner tags

`<InputBinding> ... </InputBinding>` Contains the description of how to map the key values to the parameters of one or more of the MBO operations and/or object queries to be executed when this online request or submit action is received from the client.

`<OutputBinding> ... </OutputBinding>` Contains the description of how to map the response from the object query to key values.

InputBinding

```
<InputBinding optype="create" opname="create"
generateOld="false"> ... </InputBinding>
```

Contains the MBO operation to invoke and how to map the key values to the parameters of that operation.

Table 3. Attributes

Attribute	Description
optype	The type of MBO operation to invoke. Must be one of these types: <ul style="list-style-type: none"> • none • create • update • delete • other
opname	The name of the MBO operation to invoke.
generatedOld	A boolean that indicates whether or not to generate old value keys.

Inner tags

<Value> ... </Value> Contains the description of where to obtain the parameter values of the MBO operations to be executed when this online request or submit action is received from the client.

Value

```
<Value sourceType="Key"
workflowKey="Reports_type_id_attribKey" attribName="id"
mboType="int"/>
```

Describes how to obtain the parameter value or attribute value from the Hybrid App message.

Table 4. Attributes

Attribute	Description
sourceType	<p>The source of the data. Must be one of these types:</p> <ul style="list-style-type: none"> • Key • BackEndPassword • BackEndUser • DeviceId • DeviceName • DeviceType • UserName • MessageId • ModuleName • ModuleVersion • QueueId • ContextVariable
workflowKey	If the sourceType is Key , the name of the key in the Hybrid App message from which to obtain the value.
contextVariable	If the sourceType is ContextVariable , the name of the context variable from which to obtain the value.
paramName	If present, the name of the parameter the value is supplying.
pkName	If present, the name of the personalization key the value is supplying.
attribName	If present, the name of the attribute name the value is supplying. This value may, or may not, be present together with paramName.
parentMBO	The name of the parent MBO, if any.
relationshipName	The name of the relationship, if any.

Attribute	Description
mboType	<p>The type of the value in MBO terms. Must be one of these types:</p> <ul style="list-style-type: none"> • string • char • date • datetime • time • int • byte • short • long • decimal • boolean • binary • float • double • list • integer • structure
array	A boolean that indicates whether or not the value is an array. The default is false.
length	The length of the parameter/attribute/personalization key.
precision	The precision of the parameter/attribute/personalization key.
scale	The scale of the parameter/attribute/personalization key.
convertToLocalTime	A boolean that indicates whether or not to convert the value to a local time before passing it to the MBO. The default is false.

Inner tags

`<InputBinding> ... </InputBinding>` If the mboType is “list,” it will be necessary to specify child input bindings to indicate which MBO operations to invoke when a child is updated, deleted, or created.

OutputBinding

```
<OutputBinding generateOld="true"> ... </OutputBinding>
```

Contains a series of mappings that indicate how to map the results of the object query to the Hybrid App message.

Table 5. Attributes

Attribute	Description
generatedOld	A boolean that indicates whether or not to generate old value keys.

Inner tags

`<Mapping> ... </Mapping>` Contains the description of how to map the results of the object query to a key in the Hybrid App message.

Mapping

```
<Mapping workflowKey="Department_dept_id_attrbKey"
workflowType="number" attrbName="dept_id" mboType="int"/>
```

Describes how to fill a key's value in the Hybrid App message from the results of the object query.

Table 6. Attributes

Attribute	Description
workflowKey	The name of the key in the Hybrid App message to fill with the results of the object query.
workflowType	The type of the data in the Hybrid App message. Must be one of these types: <ul style="list-style-type: none"> • text • number • boolean • datetime • date • time • list • choice
attrbName	If present, the name of the attribute name to which the key is mapped.

Attribute	Description
hardCodedValue	If the workflowType is not choice, and attrib-Name is not present, the hard-coded value to which the key is mapped.
keyWorkflowKey	If the workflowType is choice, the key to which to map the dynamic display names of the choice.
valueWorkflowKey	If the workflowType is choice, the key to which to map the dynamic values of the choice.
assumeLocalTime	A boolean to indicate whether or not to assume that the values coming back from the object query are in local time or not. The default is false.
array	A boolean that indicates whether or not the value is an array. The default is false.
mboType	<p>The type of the value in MBO terms. Must be one of these types:</p> <ul style="list-style-type: none"> • string • char • date • datetime • time • int • byte • short • long • decimal • boolean • binary • float • double • list • integer • structure
relationshipName	The name of the relationship, if any.

Inner tags

`<Mapping> ... </Mapping>` If the mboType is list, you must specify child mappings to indicate how to map the attributes of child MBO instances to keys in the Hybrid App message.

Notification

```
<Notification type="onEmailTriggered" targetscreen="dept"> ...
</Notification>
```

Describes how to formulate the Hybrid App message for the given notification type and which screen to open on the device when that Hybrid App message is opened.

Table 7. Attributes

Attribute	Description
type	The type of the notification. Must be onEmail-Triggered.
targetscreen	The screen to which the client will be opened if the object query succeeds.
errorscreen	The screen to which the client will be opened, by default, if the object query fails.
<ul style="list-style-type: none"> errorlogskey errorlogmessagekey errorlogmessageaslistkey 	The keys to use to fill any error log messages.

Inner tags

`<Transformation> ... </Transformation>` Contains the description for one or more rules that dictate how to extract values from the server notification and map it to a key in the Hybrid App message.

`<Methods> ... </Methods>` Contains the description for one or more object queries to be executed when this online request or submit action is received from the client.

Rule

```
<Rule type="regex-extract" source="subject" workflowKey="ID"
workflowType="number" beforeMatch="Purchase order request \("
afterMatch="\)" is ready for review" format=""/>
```

Describes how to extract a value from the server notification and map it to a key in the Hybrid App message.

Table 8. Attributes

Attribute	Description
type	The type of the rule. Must be regex-extract .

Attribute	Description
source	<p>The source of the data to be extracted. Must be one of these sources:</p> <ul style="list-style-type: none"> • body • subject • from • to • cc • receivedDate • custom1, custom2, custom3, custom4, custom5, custom6, custom7, custom8, custom9, or custom10
workflowKey	The name of the key in the Hybrid App message to fill with the value extracted from the server notification.
workflowType	<p>The type of the data in the Hybrid App message. Must be one of these data types:</p> <ul style="list-style-type: none"> • text • number • boolean • datetime • date • time • list • choice
assumeLocalTime	A boolean to indicate whether or not to assume that the values coming back from the object query are in local time or not. The default is false.
beforeMatch	A regular expression used to indicate where the value starts.
afterMatch	A regular expression used to indicate where the value ends.
format	If the workflowType is datetime or time, the C# formatting string to be passed to DateTime.ParseExact when converting the value to a datetime.

The Look and Feel XML Files

Each device platform (WindowsMobile Professional, BlackBerry, BlackBerry6, iOS, and Android) provides a `<File>...</File>` tag, which refers to an XML file in the Hybrid App ZIP package.

The contents are similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<widget>
  <screens src="html/myAndroidhybridapp.html"
default="Start_Screen">
    <screen key="html/myAndroidhybridapp.html">
    </screen>
  </screens>
</widget>
```

Different platforms can share the same look and feel XML file, or they can use different XML files, depending on the application design. Different XML files can refer to the same HTML file, or to different HTML files, depending, again, on the application design.

When a Hybrid App package is generated using the Hybrid App Designer, with the **Optimized for appearance** option selected in Preferences, three look and feel XML files are generated: `hybridapp.xml`, `hybridapp_Custom.xml`, and `hybridapp_JQM.xml`.

Using Third-party Files

To load external JavaScript and CSS files dynamically when creating a Hybrid App package manually:

Add the path of the third-party JavaScript or CSS files to the `manifest.xml` file, in the device platform section. For example:

```
<BlackBerry>
<HTMLWorkflow>
<File>TokenSI_CustomLookAndFeel.xml</File>
<HtmlFiles>
<HtmlFile>html/css/bb/some-3rd-part.css</HtmlFile>
<HtmlFile>html/css/bb/checkbox.css</HtmlFile>
<HtmlFile>html/css/bb/datepicker.css</HtmlFile>
<HtmlFile>html/css/bb/editBox.css</HtmlFile>
<HtmlFile>html/css/bb/img/btn_check_off.png</HtmlFile>
<HtmlFile>html/css/bb/img/btn_check_on.png</HtmlFile>
<HtmlFile>html/css/bb/img/btn_radio_off.png</HtmlFile>
```

Deploying a Hybrid App Package with the Deploy Wizard

Use the Deploy wizard to make Hybrid App packages available on SAP Mobile Server.

If you are deploying to a target domain, replicate the value in the context variable. The domain deployment target must match the context variable defined. If the developer has used an incorrect context variable (for example, one used for testing environments), you can change the value assigned to one that is appropriate for production deployments.

1. In the left navigation pane of SAP Control Center, click **Hybrid Apps**.
2. From the **General** tab, click **Deploy**.
3. Click **Browse** to locate the Hybrid App package.
4. Select the file to upload and click **Open**.
5. Select the deployment mode:
 - **New** – deploys an SAP Mobile Server package and its files for the first time.
If the uploaded file does not contain an SAP Mobile Server, or an SAP Mobile Server with the same name and version is already deployed to SAP Mobile Server, you see an error message.
 - **Update** – installs a new SAP Mobile Server package with the original package name and assigns a new, higher version number than the existing installed SAP Mobile Server package. SAP recommends that you use this deployment mode for major new changes to the SAP Mobile Server package.
During the update operation, SAP Mobile Server:
 - Acquires a list of assigned application connections from the original package.
 - Installs and assigns the package a new version number.
 - Prompts the administrator to specify application connection assignments from the acquired list of assigned application connections.
 - Preserves existing notifications.
 - Preserves the previous SAP Mobile Server package version.
 - **Replace** – replaces an existing SAP Mobile Server package with a new package, but maintains the same name and version. SAP recommends that you use the replace deployment mode for minor changes and updates to the SAP Mobile Server package, or during initial development.

During the replace operation, SAP Mobile Server:

- Acquires a list of assigned application connections for each user of the original package.
- Uninstalls the original package.
- Installs the new package with the same name and version.

Develop Hybrid Apps Using Third-party Web Frameworks

- Assigns the original application connections list to the new package, thus preserving any application connection assignments associated with the original package.

The package is added to the list of deployed packages, which are sorted by Display Name.

Next

Configure the deployed package if you want it to have a different set of properties in a production environment.

Develop a Hybrid App Using the Hybrid App Designer

Hybrid Apps support the occasionally connected user and addresses the replication and synchronization issues those users present for the back-end system.

A Hybrid App application requires an integration module on the server side, which is implemented by a static set of logic that processes Hybrid App-specific metadata to map keys to and from mobile business object attributes, personalization keys, and parameters. This integration module processes the notifications identified by matching rules configured for the server-initiated starting point and also processes the responses sent to the server from the device.

You can develop Hybrid Apps that work on these platforms:

- Android
- Apple iOS
- BlackBerry
- Windows Mobile Professional

The Hybrid App Designer provides UI controls that make development of Hybrid Apps fast and easy. For information about using the Hybrid App Designer to design and develop Hybrid Apps, see online help, *SAP Mobile WorkSpace - Hybrid App Package Development*.


See *Supported Hardware and Software* for supported version levels.

Deploy the Hybrid App Package to SAP Mobile Server

Use the Hybrid App generation wizard to generate the Hybrid App package and deploy it to SAP Mobile Server to make it available for device clients.

Generating Hybrid App Files and Deploying a Package

Use the Hybrid App Package Generation wizard to generate a Hybrid App package, or to generate Hybrid App files that you can deploy to specific devices.

1. In the Hybrid App Designer, click the  code generation icon on the toolbar.
Alternatively, right-click in the Flow Design or Screen Design page and select **Generate Hybrid App Package**.
2. Specify the Mobile Server profile.
3. Choose the option to either generate a package or generate files for one or more specific platforms. Specify the required parameters, and click **Finish** to generate the files and close the wizard.

Note: The files to be generated are listed in the File Order tab of the Flow Design properties view for the application. You can optionally add or remove files or change the order in which they are loaded in the running application. See *Flow Design Application Properties* for more information.

The generated files are created in your project, visible in Workspace Navigator under Generated Hybrid App.

4. Deploy the Hybrid App to an appropriate device or simulator.

See the *Developer Guide: Hybrid Apps* for information about how to configure devices or simulators for the Hybrid App Package.

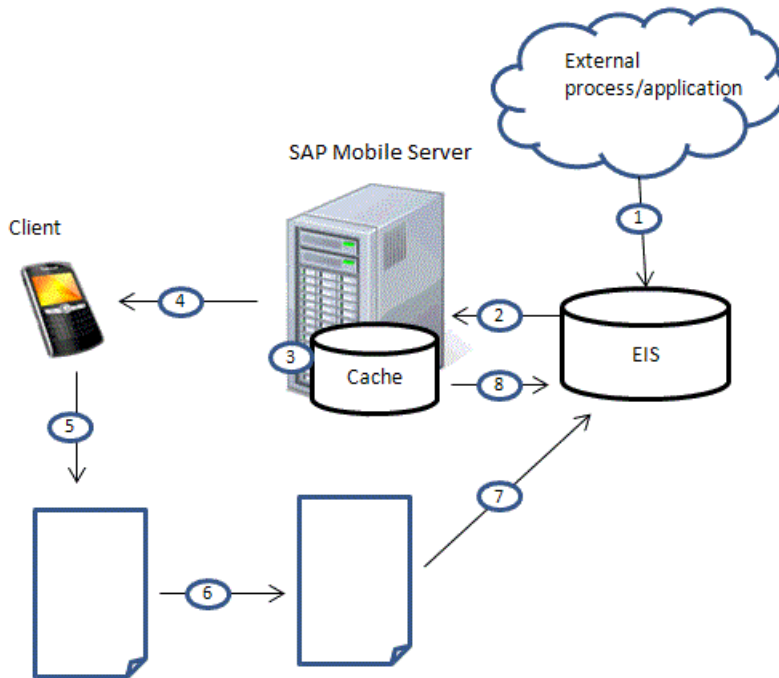
See *SAP Control Center for SAP Mobile Platform* documentation for information about managing devices, Hybrid App assignments, and users.

Hybrid App Patterns

The Hybrid Web Container allows you to create lightweight applications that implement various business solutions. These are some of the primary Hybrid App and the SAP Mobile Platform patterns (models):

- Online lookup – the client retrieves data directly from the EIS. This pattern typically uses a client-initiated starting point.
- Server notification – the enterprise information system (EIS) notifies SAP Mobile Platform of data changes and SAP Mobile Platform sends notifications to subscribed devices based on the rules.
- Cached data – the client retrieves data from the SAP Mobile Server cache. This pattern typically uses a client-initiated starting point.

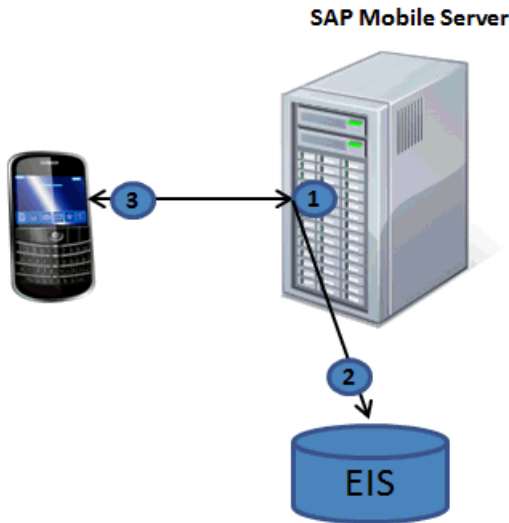
These patterns are not mutually exclusive. You can create applications that combine patterns in various ways to meet business needs. For example:



1. An external process or application updates EIS data.
2. The changed data triggers a data change notification (DCN), which is sent to SAP Mobile Server, or a message from another client updates mobile business object (MBO) data contained on SAP Mobile Server.
3. The DCN could be programmed to update MBO data.
4. SAP Mobile Server notifies the client that some action needs to be taken.
5. The client views the message.
6. The client opens a screen to perform the required action. The form may, for example, call an object query to return cached data or online data, call an MBO operation, or perform some other action.
7. The client sends an update to SAP Mobile Server.
8. SAP Mobile Server updates the EIS.

Online Lookup

This pattern provides direct interaction between the data requester (client) and the enterprise information system (EIS), supplying real-time EIS data or cached data.



While the server notification and cached data patterns are flexible regarding MBO definition and cache group policy, the online lookup pattern must have at least one `findByParameter` and use the Online cache group policy:

1. The client requests data using the `findByParameter` object query.
2. Since the MBO associated with the object query is in a cache group that uses an Online policy, SAP Mobile Server retrieves the requested data directly from the EIS and not the cache.
3. Online data is returned to the client.

In this example, online data retrieval by the client is triggered when the user selects the menu item that calls the `findByParameter` object query.

Implementing Online Lookup for Hybrid Apps

Define an MBO with at least one load argument that maps to a propagate-to attribute, add the MBO to a cache group that uses an Online policy, then define the Hybrid App that calls the `findByParameter` object query to return real-time results from the EIS.

Defining Load Arguments from Mapped Propagate to Attributes

Create an MBO with at least one load argument, map as propagate to attributes, then assign the MBO to a cache group that uses an Online policy.

1. From SAP Mobile WorkSpace, create an MBO that has at least one load argument. For example, you could define an Employee MBO as:

```
SELECT emp_id, emp_fname, emp_lname, dept_id
FROM sampledб.дба.employee WHERE dept_id = :deptIdLP
```
2. In the MBO Properties view, select the **Attributes > Load Arguments** tab, map each load argument to be used as an operation load argument for the Hybrid App package to a Propagate to Attribute. This example requires you to map the deptIdLP load argument to the empDeptId attribute. You must also verify that data types are INT and the default value is a valid INT.
3. Set the Online cache group policy for the MBO.
 - a) Add the MBO to a cache group that uses the Online cache group policy. For example, create a new cache group named CacheGroupOnline and set the policy to **Online**.
 - b) Drag and drop the MBO to CacheGroupOnline.The findByParameter object query is automatically generated based on all load arguments that have propagate-to attributes:
4. Deploy the project that contains the MBO to SAP Mobile Server.

Binding the findByParameter Object Query to a Menu Action

For synchronous, online data access, define an Online Request menu action and bind it to the findByParameter object query.

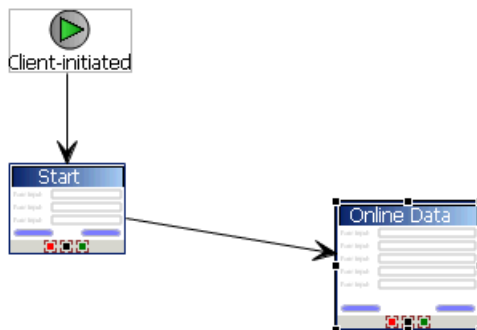
Prerequisites

You must have propagate-to attributes mapped to MBO load parameters, and the deployed MBO must use an Online cache group policy. SAP Mobile Platform services must be running.

Task

1. From SAP Mobile WorkSpace, launch the Hybrid App Designer.
2. From the Flow Design screen, double-click the screen for which you are defining a mapping to open it in the Screen Design tab.

For example, you can have a client-initiated starting point with a Start screen that connects to the Online Data screen.



3. Highlight the menu item you want to map, or create a new menu item.
4. Define a Submit action that invokes the `findByParameter` object query:
 - a) From the General tab, select **Online Request** as the Type.
 - b) In the Details section, select **Search** to locate the MBO that contains the `findByParameter` object query.
 - c) Click the **General** tab, select **Invoke object query** and select **`findByParameter`**.

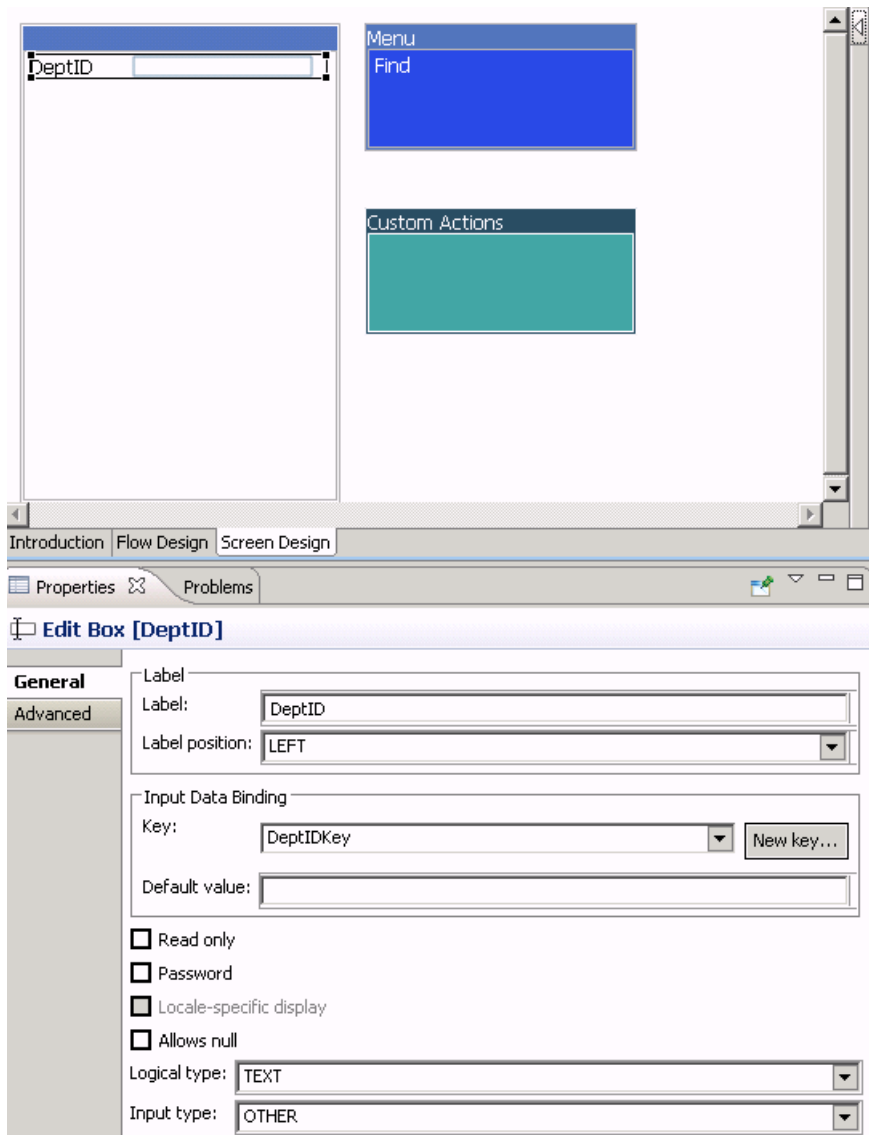
If you select the Parameter Mappings tab, you see all the load parameters defined for the MBO and used to generate the `findByParameter` object query. In addition to Key, you can map parameters to `BackEndPassword`, `BackEndUser`, `DeviceId`, `DeviceName`, `DeviceType`, `UserName`, `MessageId`, `ModuleName`, `ModuleVersion`, and `QueueId`.

Unmapped parameters can get their value from the default value, if specified, or from the personalization key value they are mapped to, if that is specified. If the key is unmapped, and the parameter has no default value and is not mapped to a personalization key value, the parameter value is empty (NULL for string, 0 for numeric, and so on).

Defining the Control that Contains the `findByParameter` Object Query Parameter

Add a control to pass the load argument to SAP Mobile Server. Define a screen that displays the results returned from the EIS.

1. Define a control that passes the load argument to SAP Mobile Server from the screen (named Online Data) that contains the menu item (named Find) that invokes the `findByParameter` object query:
 - a) Select an **EditBox** control and click in the control area.
 - b) Name the EditBox `DeptId`.
 - c) From the Properties view, select **New key** and name it `DeptIdKey`. Click **OK**.

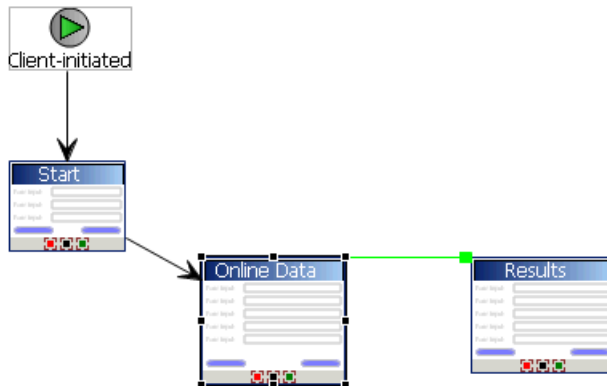


2. Select the **Find** menu item, and from the Parameter Mappings tab, map parameters to input keys defined for the controls. For example, map the deptIDLP parameter to the DeptIdKey key.
3. Define a screen that displays the results of the findByParameter object query:
 - a) From the Flow Design window, add a new Screen and name it **Results**. Select the Screen Design tab.
 - b) Drag and drop a **Listview** control onto the control area.

Develop a Hybrid App Using the Hybrid App Designer

- c) Select the Flow Design tab and double-click the **Online Data** screen to open it.
- d) Select the **Find** menu item, and in the Properties view, specify **Results** as the success screen.

The Online Data screen now sends successful results returned by the EIS to the Results screen. The Flow Design window indicates the connection between the screens.



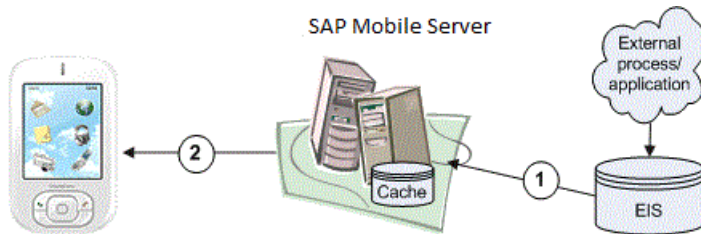
- 4. Configure the Results screen to display the results. In this example, the Emp MBO, contains three attributes: Id, empName, and empDeptId. Create a Listview with a cell for each attribute to display the results returned from the EIS as a list:
 - a) From the Flow Design window, double-click the **Results** screen to display it in the Screen Design window.
 - b) Select the control area, select the General tab in the Properties view, and for the Input Data Binding Key select **<MBOName>** (where MBOName is the name of the MBO).
 - c) Select the **Cell** tab, then click **Add** to add cell line 0.
 - d) Select **Add** in the "Fields for cell line 0" section, then select the **Emp_id_attribKey** key. Click **OK**.

This maps cell line 0 with the id attribute for the Emp MBO results returned by the object query.
 - e) Repeat steps 3 and 4 again for the remaining two attributes.
- 5. Select the **Problems** view, and verify there are no errors.

You now have a deployable Hybrid App package that passes the DeptID value to the findByParameter object query which returns matching EIS results and displays them in the Results screen.

Server Notification

Configure matching rules for MBO-related data on SAP Mobile Server. Any data changes matching these rules trigger a notification from SAP Mobile Server to the client.



1. MBO data is updated from the EIS, by an external process or application that updates EIS data and triggers a data change notification (DCN), or a scheduled data refresh.
2. If matching rules that correspond to the notification message fields are configured for the MBO and Hybrid App package, SAP Mobile Server sends a notification to the client.

Implementing Server Notification for Hybrid Apps

Set up SAP Mobile Server to send notifications to Hybrid Apps when matching rules are encountered.

Defining the Mobile Business Object for Server Notification

The server notification pattern supports any number of MBO definitions. For this example, create an MBO with one load argument, assign the load argument a propagate-to attribute value, then assign the MBO to a cache group that uses an Online policy.

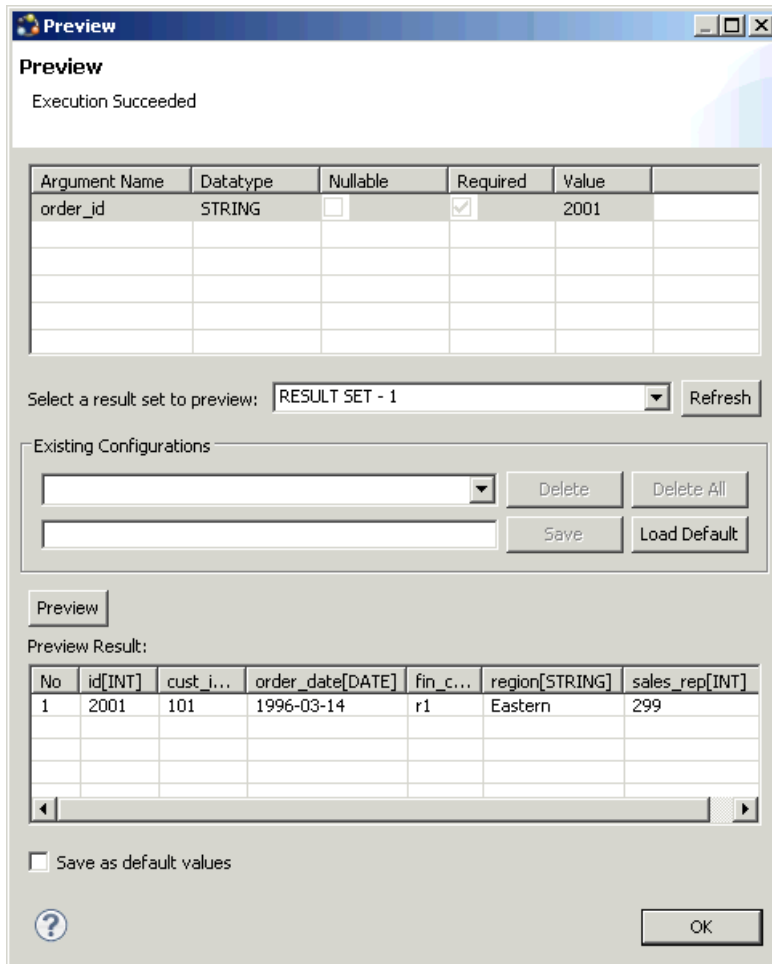
The MBO definition described here allows retrieval of online results by the Hybrid App to which the MBO belongs.

1. In SAP Mobile WorkSpace, create an MBO from the sampledadb database that has at least one load argument. For example, you could define a Sales_order MBO as:

```

SELECT  id,
        cust_id,
        order_date,
        fin_code_id,
        region FROM sampledadb.dba.sales_order
WHERE id = :order_id
  
```

2. Preview the MBO by selecting **Preview** from the Definition tab. Enter 2001 as the value. The preview returns one row from the sales_order table based on the id attribute (2001).



3. In the MBO Properties view, click the **Load Arguments** tab, select the **id** attribute as the Propagate to attribute that maps to the order_id load argument. Change the datatype to INT, and include an integer value for the data source default value.
4. Set the Online cache group policy for the MBO.
 - a) Add the MBO to a cache group that uses the Online cache group policy. For example, create a new cache group named CacheGroupOnline and set the policy to **Online**.
 - b) Drag and drop the MBO to CacheGroupOnline.

The findByParameter object query is automatically generated based on the order_id load argument:

```
SELECT x.* FROM Sales_order x WHERE x.id = :order_id
```

5. Deploy the project that contains the MBO to SAP Mobile Server.

Creating the Server-Driven Notification Starting Point

Create a new Hybrid App with a server-initiated starting point.

1. From SAP Mobile WorkSpace, select **File > New > Hybrid App Designer**.
2. Select the folder that contains the Sales_order MBO as the parent folder, name the file Sales_order.xbw, and click **Next**.
3. In the Starting Points screen, select **Responds to server-driven notifications**, and click **Next**.
4. Configure the starting point:
 - a) In the Select a Mobile Business Object and Object Query screen, select **Search**.
 - b) Select the project that contains the Sales_order MBO and select **Search**. Select the **Sales_order** MBO and select **OK**.
 - c) Select the **findByParameter** object query.

The order_id parameter appears in the Parameters field. Click **Next**.
 - d) Specify a sample notification. Enter Order (2001) created in the Subject line. Click **Next**.
 - e) Click and drag to select "Order (", while this phrase is highlighted, right-click and select **Select as Matching Rule**.
 - f) Click **Next**. Select **order_id**. In the Extraction Rule Properties:
 1. Select **Subject** as the field.
 2. Select "Order (" as the Start tag.
 3. Select ") created" as the End tag.

When the notification is sent to the client, the sample value (2001 in this example), is replaced with the order_id key, which identifies the id attribute of the object query. The Hybrid App the client receives is populated with values returned by the findByParameter object query.

Develop a Hybrid App Using the Hybrid App Designer

New

Identify Parameter Values

For each parameter of the MBO operation, specify where to find the data in the incoming message.

Extraction Rules

Key	Field	Start Tag	End Tag	Format	Sample Value	Type
order_id	subject	Order {()} created		2001	int

Extraction Rule Properties

Field:

Start tag: ...

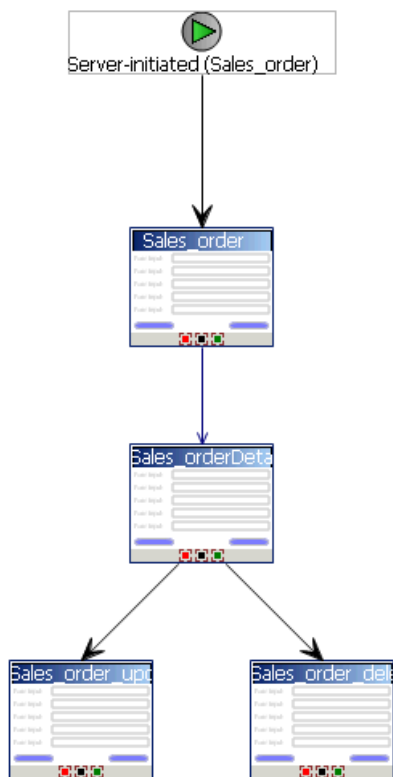
End tag: ...

Format:

Sample notification field contents:

Value extracted from sample notification field contents:

5. Click **Finish** to create default screens and starting points.
Screens are populated with menu items and controls based on the MBO definition.



6. Deploy the Hybrid App package to SAP Mobile Server.

Sending an Order Notification to the Device

Use the "Send a notification" option to send a message to the registered user, which tests the server notification process.

Prerequisites

Before sending notification to the client, you must:

1. Register the Hybrid App connection in SAP Control Center.
2. Download and configure the Hybrid Web Container on the device or emulator.

Task

Use this method only for testing purposes, during development. In a production system, notifications would come in as DCN, or e-mail-based notifications.

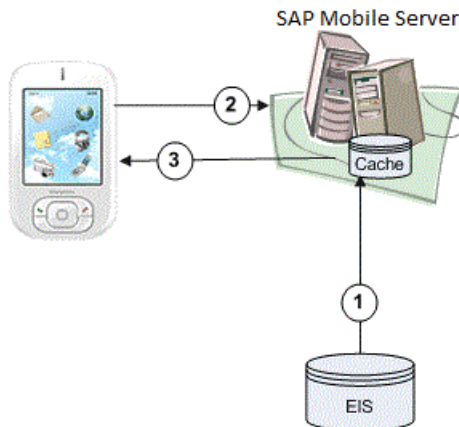
Develop a Hybrid App Using the Hybrid App Designer

1. In the Flow Design of the Hybrid App Designer, right-click and select **Send a notification**.
2. Select **Get Device Users**, and set the "To" field to **User1**, or whatever device user is registered in SAP Control Center and assigned to the Hybrid App package.
3. In the Subject field, enter a sales order that meets the matching rules criteria defined for the Sales_order Hybrid App. For example:
Order (2001) created
4. Click **Send**.

The message is sent to the device. The number 2001 in the notification identifies and returns row 2001 (the findByParameter object query parameter).

Cached Data

This pattern is efficient when access to cached data is sufficient to meet business needs. For example, it may be sufficient to refresh the cache once a day for noncritical MBO data that changes infrequently.



1. EIS data is cached based on the MBO cache policy (Scheduled or On demand). Either policy lets you define the length of time for which cached data is valid.
2. The Hybrid App requests data through an object query.
3. Cached data is returned to the client if it is within the cache policy's specified cache interval.

Implementing the Cached Data Pattern

Define an MBO that uses either a Scheduled or On demand cache group policy to allow the Hybrid App to which it belongs to retrieve cached data.

Defining the Mobile Business Object

Create an MBO with the required attributes, assign the MBO to a cache group that uses a scheduled policy, and define an object query that returns the results from the SAP Mobile Server cache (also called the CDB) to the client.

This example defines an MBO that retrieves employee benefit information for all employees of a given department based on the dept_id attribute using the findByDeptId object query.

1. From SAP Mobile WorkSpace, create an MBO. For example, you could define the employee MBO as:

```
SELECT emp_id,
       emp_fname,
       emp_lname,
       dept_id,
       bene_health_ins,
       bene_life_ins,
       bene_day_care
FROM sampledb.dba.employee
```

2. Set the cache group policy for the MBO:
 - a) Create a new cache group named CacheGroupScheduled and set the policy to **Scheduled**. Set the **Cache interval** to 24 hours, so the cache is refreshed once a day.
 - b) Drag and drop the MBO to CacheGroupScheduled.
3. Define an object query for the MBO that retrieves employee information based on the dept_id attribute. For example, define the findByDeptId object query as:

```
SELECT x.* FROM Employee x
WHERE x.dept_id = :deptIDLp
```

Object Query 'findByDeptId'

Edit Object Query
Edit the object query

Name:

Comment:

Parameters:

Name	Datatype	Nullable	Mapped to
deptIDLP	INT	<input type="checkbox"/>	dept_id

Buttons: Add, Delete, Delete All, Up, Down

Query Definition

```
SELECT x.* FROM Employee x
WHERE x.dept_id = :deptIDLP
```

☒ Create an index

Return Type:

☐ Return a single object ☒ Return multiple objects ☐ Return a result set

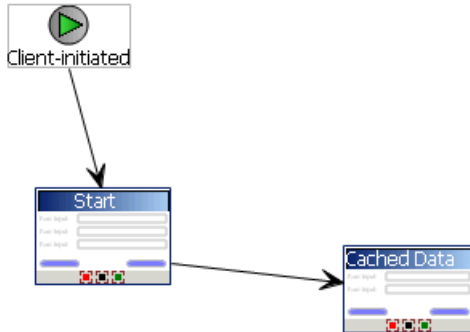
4. Deploy the project that contains the MBO to SAP Mobile Server.

Binding the findByDeptId Object Query to a Menu Action

For access to cached data, define a menu action and bind it to the findByDeptId object query.

1. From SAP Mobile WorkSpace, launch the Hybrid App Designer.
2. From the Flow Design screen, double-click the screen for which you are defining a mapping to open it in the Screen Design tab.

For example, you can have a client-initiated starting point with a Start screen that connects to the Cached Data screen.



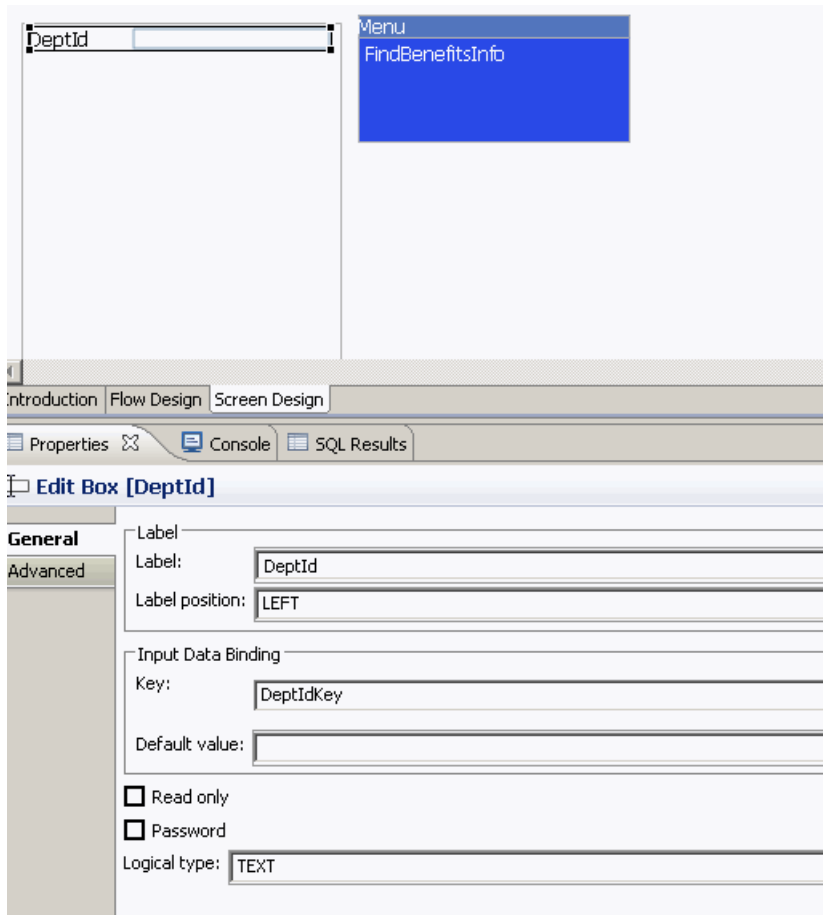
3. Highlight the menu item you want to map, or create a new menu item.
4. Define a Submit action named FindBenefitsInfo that invokes the findByDeptId object query:
 - a) In the Properties view, in the General properties for the selected menu item, select **Online Request** as the Type.
 - b) In the Details section, select **Search** to locate the MBO that contains the findByDeptId object query.
 - c) Click the **General** tab, select **Invoke object query** and select **findByDeptId**.

If you select the Parameter Mappings tab, you see the parameters associated with the object query (findByDeptId). Map this parameter to a key.

Defining the Control that Contains the findByDeptId Object Query Parameter

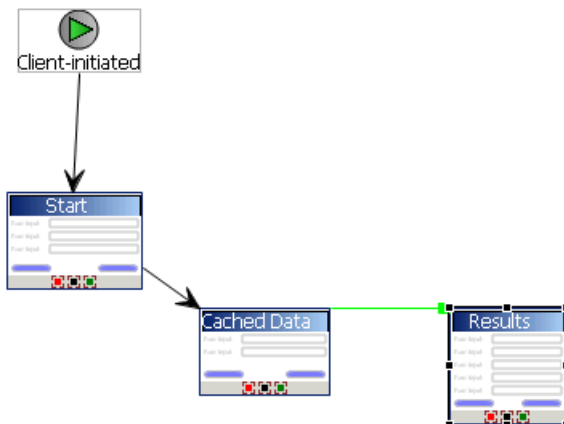
Add a control to pass the object query parameter to SAP Mobile Server. Define a screen that displays the results returned from the SAP Mobile Server cache.

1. Define a control that passes the object query parameter to SAP Mobile Server from the screen (named Cached Data) that contains the menu item (named FindBenefitsInfo) that invokes the findByDeptId object query:
 - a) Select an **EditBox** control and click in the control area.
 - b) Name the EditBox DeptId.
 - c) From the Properties view, select **New key** and name it DeptIdKey. Click **OK**.

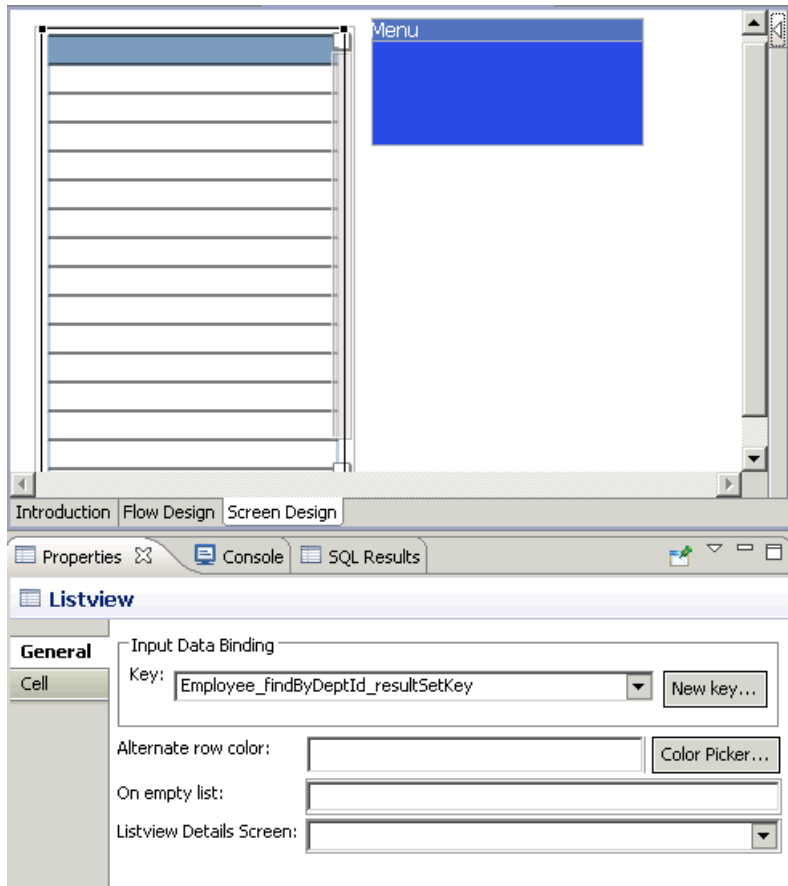


2. Select the FindBenefitsInfo menu item, and from the Parameter Mappings tab, map parameters to input keys defined for the controls. For example, map the deptIDLp parameter to the DeptIdKey key.
3. Define a screen that displays the results of the findByDeptId object query:
 - a) From the Flow Design window, add a new Screen and name it Results. Select the Screen Design tab.
 - b) Drag and drop a **Listview** control onto the control area.
 - c) Select the Flow Design tab and double-click the **Cached Data** screen to open it.
 - d) Select the **FindBenefitsInfo** menu item, and in the Properties view, in General properties, select **Online Request** as the Type and in the Details section, select **Results** as the Success screen.

The Cached Data screen now sends successful results returned by the SAP Mobile Server cache to the Results screen. The Flow Design window indicates the connection between the screens.



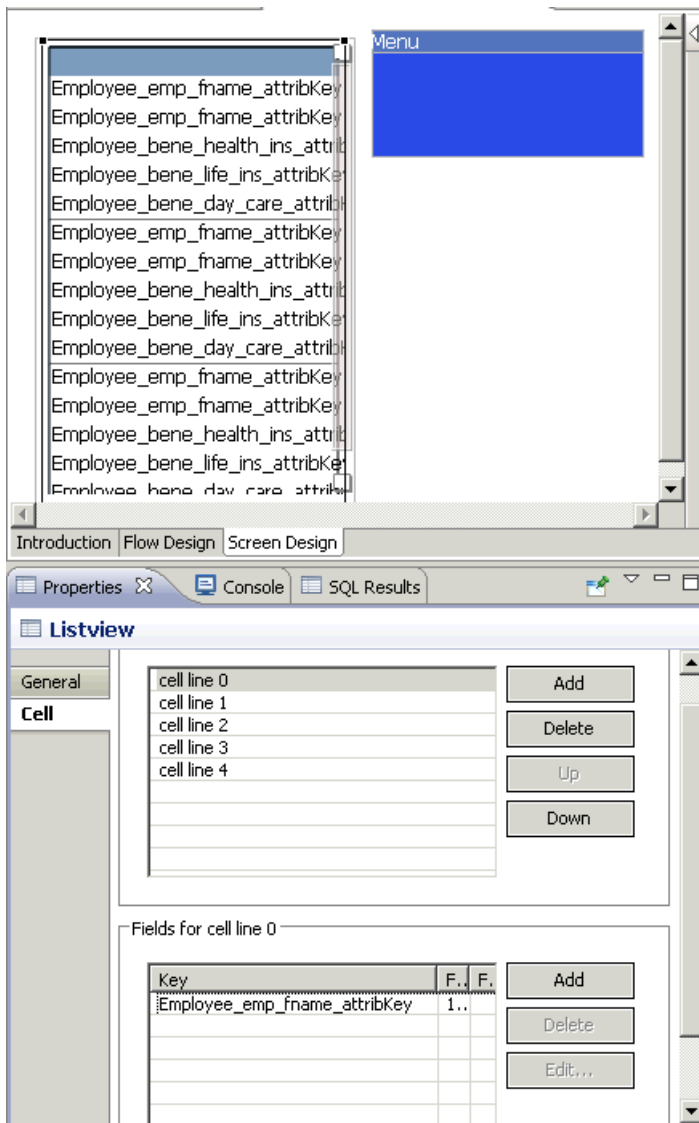
4. Configure the Results screen to display the results. In this example, the Employee MBO, contains seven attributes that identify the employee and their benefits. Create a Listview with a cell for each attribute to display the results returned from the cache as a list:
 - a) From the Flow Design window, double-click the **Results** screen to display it in the Screen Design window.
 - b) Select the control area, select the **General** tab in the Properties view, and for the Input Data Binding Key select **MBOName_findByDeptId_resultSetkey** (where MBOName is the name of the MBO).



- c) Select the **Cell** tab, then click **Add** to add cell line 0.
- d) Select **Add** in the "Fields for cell line 0" section, then select the **Employee_emp_fname_attribKey** key. Click **OK**.

This maps cell line 0 with the id attribute for the Emp MBO results returned by the object query.

- e) Repeat steps 3 and 4 again for the remaining employee's last name and benefits related attributes.



5. Select the **Problems** view, and verify there are no errors.

You now have a deployable Hybrid App package that passes the DeptID value to the findByDeptID object query which returns matching cached results and displays them in the Results screen.

Binding Transient Personalization Keys to Hybrid App Keys

Use transient personalization key values to determine the data to be cached.

Prerequisites

You must have transient personalization keys mapped to Mobile Business Object load arguments.

Task

1. Launch the Hybrid App Designer from SAP Mobile WorkSpace and create a new Hybrid App:
 - a) Select **File > New > Hybrid App Designer**.
 - b) Select the parent folder that contains the MBO with a load argument mapped to a transient personalization key. Name the file and click **Next**.
 - c) Select **Responds to server-driven email notifications** from the Starting Points screen and click **Next**.
 - d) Select the MBO that contains the load argument to transient key mapping in the Search for MBO screen and click **OK**, then click **Next**.
 - e) Specify sample e-mail contents and click **Next**.
 - f) Specify the matching rules used to trigger a screen flow by highlighting the text, right-clicking it, and selecting **Select as matching rule**.
 - g) Click **Finish**.
2. In the Hybrid App Designer, map the personalization keys to the Hybrid App keys for the menu item:
 - a) From the Flow Design screen select the operation for which you are defining a mapping.
 - b) Select the Screen Design tab, and highlight the menu item you want to map.
 - c) Select **Personalization Key Mappings**, click **Add**, and select a personalization key from the drop-down list and the key to which it maps.

You can also fill the personalization key values from values extracted from the e-mail, depending on from where you are invoking the object query.

When the application runs, the values are sent from the client which are used to fill the load argument values, and determine what data is cached in the SAP Mobile Server cache (CDB) and returned to the client.

Hybrid App Package Customization

The designer-based user interface is customizable using HTML, JavaScript and CSS Web technologies.

Customizing Generated Code

Modify generated JavaScript code to customize the Hybrid App.

1. Use the Hybrid App Package Generation wizard to generate the Hybrid App package and its files.

When the Hybrid App package is generated, the `Custom.js` file is generated if not already present in the project. The `Custom.js` file is located in `Generated Hybrid App\<hybridapp_project_name>\html\js`.

2. Right-click the `Custom.js` file and select the editor in which to open the file.
3. Modify the JavaScript code in the file or add your own code.
4. Save and close `Custom.js`.

Since `Custom.js` is generated only if it is not already present in the Hybrid App project, it is not created again if you subsequently generate the Hybrid App package. In this way, your customizations are preserved.

5. Deploy the Hybrid App package to SAP Mobile Server.

Any time you customize the code, you must redeploy the Hybrid App package to SAP Mobile Server.

You can also add your own separate JavaScript files to `Generated Hybrid Apps \hybridapp_project_name\html\js`, then add custom code to the `Custom.js` file that calls the functions in the JavaScript files you added. Modularizing your custom code can prevent the `Custom.js` file from becoming too long, and make it easier for multiple developers to collaborate on the same Hybrid App.

Adding Local Resources to a Hybrid App Project

When loading resources using custom JavaScript, be aware of the folder structure.

Depending on localization, the structure and path to the local resource may be different. Possible folder paths include:

- `.../html/default/hybridapp.html`
- `.../html/{locale}/hybridapp.html`
- `.../html/hybridapp.html`

Referencing custom resources in HTML elements requires the use of relative URLs. The parent directory may be the HTML directory, the root, or something else. There is no guarantee that the URL structure is always `http://hostname/html/hybridapp.html`. It is possible to copy the resources into each localization directory or reference the resources from one directory (paying attention to localization paths).

An example of a useful helper function to get the relative path to the HTML directory is:

```
/**
 * Returns relative URL to the html directory
 */
```

```
function getRelativeRoot()
{
    return ((resources != null) ? "../" : "")
}

// Helper function usage
var imageElement = document.getElementById("ImageElement");
imageElement.src = getRelativeRoot() + "images/myImage.gif";
```

Generated Hybrid App Files

When you use the Hybrid App Generation wizard to create a Hybrid App package, all the package files are generated the first time. Subsequent generations overwrite only a small subset of the files.

Generated package files are created in a top-level folder with the name of the Hybrid App. If you choose the option to generate into the current project, this file is visible in WorkSpace Navigator under the project `Generated Hybrid App` folder.

These files are always generated:

- *hybridapp-name.zip* – a single archive containing all of the Hybrid App files, including the Web application files, look and feel files, and JavaScript files.
- *manifest.xml* – describes the contents of the *hybridapp-name.zip* file.
- *datajs-version.js* – a JavaScript library of functions for ODATA and native device services that are not included in Hybrid Apps by default. By referencing these functions in your customization (in *Custom.js*, you can incorporate functionality from third-party JavaScript SDKs into your Hybrid Apps.

These files are regenerated only if you select the **Generate platform specific files** option in the Hybrid App Package Generation wizard:

- *hybridapp.html* – contains all the screens in the Hybrid App, each in its own div element. This is used with the **Optimize for performance** look and feel. On Windows Mobile, it is used for all looks-and-feels.
- *hybridapp_Custom.html* – contains all the screens in the Hybrid App.
- *hybridapp_jQM.html* – contains all the screens in the Hybrid App. This is used with the **Optimize for appearance** look and feel on iOS, BlackBerry, and Android.
- *WorkflowClient.xml* – contains metadata that specifies how to map the data in the Hybrid App message to and from calls to Mobile Business Object (MBO) operations and object queries.
- *hybridapp_name.xml* – look and feel file that uses the basic *hybridapp_name.html* file.
- *js* and *css* – subfolders containing the Javascript and CSS style sheet files for the application, including these files:
 - *Resources.js* – allows you to access localized string resources.

- `HybridApp.js` – contains functions for common menu, screen, and database operations.
- PhoneGap JavaScript file. Typically named `js\platform\cordova-x.x.x.js`, for any Hybrid App package that is built for an Android, iOS, or BlackBerry device using the PhoneGap library. The file is copied from `<SMP_HOME>\MobileSDK<version>\HybridApp\API\Container`.

These files are generated only if you select the **Generate** option and the files do not exist:

- `API.js` and `Utils.js` – provide Hybrid App functions used to communicate with the Hybrid Web Container.
- `Custom.js` – enables you to add JavaScript code to customize the Hybrid App. Your file is preserved each time you regenerate the package.

You can edit this file to customize your Hybrid App. It is generated the first time, but is not overwritten subsequently. In this way, your changes are preserved each time you regenerate the Hybrid App package. Examples of ways you can customize the Hybrid App include:

- Manipulating HTML elements.
- Writing code that is called before or after generated behavior is invoked for menu items.
- Implementing custom validation logic.
- `WorkflowMessage.js` – provides functions to access Hybrid App message resources.
- All `*.css` files – defines formatting rules to render the screens in HTML.

These files are overwritten when you regenerate a package:

- All the files in the top-level `Generated Hybrid App\hybridapp-name` folder, including the XML and ZIP files.
- The files in the `html` subfolder.

Generated HTML Files

The Hybrid App Designer generates these HTML files.

- `hybridapp.html` – contains all the screens in the Hybrid App, each in its own `div` element. This is used with the **Optimize for performance** look and feel. On Windows Mobile, it is used for all looks-and-feels.
- `hybridapp_Custom.html` – contains all the screens in the Hybrid App.
- `hybridapp_jQM.html` – contains all the screens in the Hybrid App. This is used with the **Optimize for appearance** look and feel on iOS, BlackBerry, and Android.

Note: In Preferences, **Optimize for appearance** is the default look and feel.

Look and Feel Files

By default, on BlackBerry 6.0, Android, and iOS platforms, the jQuery Mobile look and feel is used. On BlackBerry 5.0, a custom look and feel is used as the default.

Note: In Preferences, **Optimize for appearance** is the default look and feel.

CSS files include:

- `jquery.mobile-1.1.0.css` – located in `Generated Hybrid App\Hybrid App name\html\css\jquery` folder and used on BlackBerry 6.0, Android, and iOS platforms. By default, pages are generated using the B data theme. Modify the `ui-body-a` class selector in this file to modify the look and feel, for example, the background image or color.
- `master.css` – located in `Generated Hybrid App\Hybrid App name\html\css\bb` and used on the BlackBerry 5.0 platform. This is used on the BlackBerry 5.0 platform when the Optimize for appearance preference is selected. Modify the `body` selector to change the look and feel, for example, the background color.
- `stylesheet.css` – located in `Generated Hybrid App\Hybrid App name\html\css`. This look and feel is considerably simpler, using no JavaScript code to manipulate the controls, and only a single CSS file. This style sheet is used on all platforms for the Optimize for performance preference is selected. To modify the background color for this look and feel, modify the `body` selector.

Default Look and Feel

The default look and feel is provided by the jQuery Mobile framework.

In Preferences, **Optimize for appearance** is the default look and feel.

For the standard look and feel, the layout of the HTML at a high level is:

- Each screen has a block, contained in a `div` element, with attributes `data-role="page"` and `data-theme="a"`. Each `div` element has a `div` child element with a `data-role="header"` attribute and a child element for the menu. Use the contents of the header `div` to manipulate the menu.

```
<div data-role="page" data-theme='a'
id="Department_createScreenDiv">
  <div data-role="header" data-position="inline">
    <a data-icon="arrow-l"
id="Department_createScreenDivCancel" name="Cancel"
onclick="menuItemCallbackDepartment_createCancel();" > Cancel</a>
    <h1>Department_create</h1>
    <a id="Department_createScreenDivCreate" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();" >
Create</a>
  </div>
```

- The menu has one anchor (`a`) element for each menu item:


```
<a id="Department_createScreenDivCreate" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();">
Create</a>
```

- In addition to a menu, each screen div has a child div element with a data-role="content" attribute, where the controls are hosted. The content div element has a child div with a data-role="scroller" attribute. This div in turn has a form with a number of div elements. The content div is where you can do customizations, such as branding.

```
<div data-role="content" class="wrapper" >
  <div data-role="scroller">
    <form name="Department_createForm"
id="Department_createForm">
      <div class="customTopOfFormStyle" ><span
id="Department_createForm_help" class="help"></span></div>
      <div class="customTopOfFormStyle"
id="topOfDepartment_createForm"></div>
      <div class="editbox">
        <label class="left"
for="Department_create_dept_name_paramKey">Dept name:</label>
        <input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_name_paramKey_help"
class="help"></span>
      </div>
```

The first div element is a block used to display help in a span element.

The next div is a built-in element that can be used to find the top of the form. The last div is another built-in element that can be used to find the bottom of the form.

In the Custom.js file, it is recommended that you add customizations such as branding to the div element, "TopOf" ScreenKey "Form" and "bottomOf" screenKey "Form." For example:

```
/*
var screenKey = getCurrentScreen();
var form = document.forms[screenKey + "Form"];
if (form) {
var topOfFormElem = document.getElementById("topOf" screenKey +
"Form");
! topOfFormElem.innerHTML = "Use this screen to ...";
var bottomOfFormElem = document.getElementById("bottomOf"
screenKey + "Form");
bottomOfFormElem.innerHTML = "<a href=\"help.html\">Click here to
open help</a>";
}
*/
```

All the other divs in the form correspond to the controls put on that screen during design time in the Hybrid App Designer. You might see, for example, a div that holds a label and a textbox (input element). When the page is opened, the controls are enhanced by jQuery Mobile to supply additional functionality for controls like buttons, sliders, text inputs, and combo boxes.

A typical Hybrid App with this look and feel, without extraneous attributes, might look like this:

```
<html>
  <body onload="hwc.onHybridAppLoad();">
    <div data-role="page" data-theme='a'
id="Department_createScreenDiv">
      <div data-role="header" data-position="inline">
        <a data-icon="arrow-l" id="Department_createScreenDivCancel"
name="Cancel" onclick="menuItemCallbackDepartment_createCancel();">
Cancel</a>
        <h1>Department_create</h1>
        <a id="Department_createScreenDivCreate" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();">
Create</a>
      </div>
      <div data-role="content" class="wrapper" >
        <div data-role="scroller">
          <form name="Department_createForm"
id="Department_createForm">
            <div class="customTopOfFormStyle" ><span
id="Department_createForm_help" class="help"></span></div>
            <div class="customTopOfFormStyle"
id="topOfDepartment_createForm"></div>
            <div class="editbox">
              <label class="left"
for="Department_create_dept_name_paramKey">Dept name:</label>
              <input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_name_paramKey_help"
class="help"></span>
            </div>
            <div class="customBottomOfFormStyle"
id="bottomOfDepartment_createForm"></div>
          </form>
        </div>
      </div>
    </body>
</html>
```

Default Look and Feel CSS Files

CSS look and feel files include:

- `jquery.mobile-1.1.0.css` – located in Generated Hybrid App \hybridapp-name\html\css\jquery folder. By default, pages are generated using the B data theme. Modify the `ui-body-a` class selector in this file to modify the look and feel, for example, the background image or color.
- `master.css` – located in Generated Hybrid App \hybridapp-name\html\css\bb. Modify the body selector to change the look and feel, for example, the background color.
- `stylesheet.css` – located in Generated Hybrid App \hybridapp-name\html\css. This look and feel is simple: it uses no JavaScript code to manipulate the

controls, and only a single CSS file. This style sheet is used on all platforms for which the Optimize for performance preference is selected. To modify the background color for this look and feel, modify the `body` selector.

BlackBerry Custom Look and Feel File

`hybridapp_Custom.html` defines the HTML structure for the BlackBerry custom look and feel.

Each screen has a `div` element block with a form element, and each form has a number of `div` child elements. The first `div` in the form has a `span` used to display help. The next `div` is a built-in element that can be used to find the top of the form. The last `div` is another built-in element that can be used to find the bottom of the form. All the `divs` in the form correspond to the controls put on that screen in the Hybrid App Designer. You might get, for example, a `div` that holds a label and a textbox (input element).

This example shows a Hybrid App with this look and feel, without extraneous attributes:

```
<html>
  <body onload="hwc.onHybridAppLoad();" >
    <div id="Department_createScreenDiv">
      <form name="Department_createForm"
id="Department_createForm">
        <div class="customTopOfFormStyle" ><span
id="Department_createForm_help" class="help"></span></div>
        <div class="customTopOfFormStyle"
id="topOfDepartment_createForm"></div>
        <div class="editbox">
          <label class="left"
for="Department_create_dept_name_paramKey">Dept id:</label>
          <input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_id_paramKey_help"
class="help"></span>
        </div>
      </form>
    </div>
  </body>
</html>
```

Optimize for Performance Look and Feel

This is a simple look and feel option that you can use on all platforms.

Note: Windows Mobile 6.x Professional platforms always use the Optimize for performance look and feel, as this platform is not supported by jQuery Mobile.

Choose the **Optimize for performance** option when you configure Hybrid App Designer preferences. For this look and feel, the layout of the HTML at a high level is:

- Each screen has a block, a `<div>` element. Each of those `<div>` elements has an unordered list element, ``, a child element for the menu. The menu has one list item, ``, for each menu item.

Develop a Hybrid App Using the Hybrid App Designer

- In addition to a menu, each `<div>` has a form element, `<form>`, where the controls are hosted.
- Each form has a single table, `<table>`, with a number of table rows, `<tr>`. The first table row has a block to display help, a `` element. The next table row is a built-in element, a table data or `<td>`, that can be used to find the top of the form.
- The last table row is another built-in element, a `<td>`, that can be used to find the bottom of the form.
- All the other rows in the form correspond to the controls put on that screen in the Hybrid App Designer. You might get, for example, a row with two table datas, the first holding a `<label>` and the second holding a textbox (`<input>`).
- A column can have only one width, so if you have more than one line, one column may contain different widths, which means the last width prevails. The contents of a field are wrapped only where there is a space. If there is no space, the contents are not wrapped. As a result, depending on the length of the data, Listviews may not respect the field widths specified in the Hybrid App Designer with this look-and-feel.

A typical Hybrid App with this look and feel, without extraneous attributes, looks similar to this:

```
<html>
  <body onload="onHybridAppLoad();">
    <div id="Department_createScreenDiv">
      <ul id="Department_createScreenDivMenu" class="menu">
        <li><a class="nav" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();">Creat
e</a></li>
        <li><a class="nav" name="Cancel"
onclick="menuItemCallbackDepartment_createCancel();">Cancel</a></
li>
      </ul>
      <form name="Department_createForm"
id="Department_createForm">
        <table class="screen">
          <tr>
            <td colspan="2"><span id="Department_createForm_help"
class="help"></span></td>
          </tr>
          <tr>
            <td colspan="2" id="topOfDepartment_createForm"></td>
          </tr>
          <tr>
            <td class="left"><label
for="Department_create_dept_name_paramKey">Dept name:</label></td>
            <td class="right"><input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_name_paramKey_help"
class="help"></span></td>
          </tr>
          <tr><td colspan="2" id="bottomOfDepartment_createForm"></
td></tr></table>
        </form>
      </div>
```

```
</body>
</html>
```

Reference

This section describes the generated files and the Hybrid App client API.

Hybrid App Client API

SAP Mobile Platform Hybrid Apps include a JavaScript API that open Hybrid Apps to customization, from including client-side business logic to changing the presentation layer.

Use the client API to build custom applications to support SAP Mobile Platform Hybrid App features and functionality.

Public JavaScript Functions

The JavaScript files contain the functions that you can access for use with Hybrid App package customization.

The files where the Hybrid Web Container JavaScript APIs are defined are located in `<SMP_HOME>\UnwiredPlatform\MobileSDK<version>\HybridApp\API\Container`.

Note: The detail of the individual APIs is not available if you are viewing this document from DocCommentXchange (<http://dcx.sybase.com>) or in PDF format. You can access this information by going to Product Documentation: access <http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories>, then navigate to the current version of this topic.

These JavaScript files are also included:

- `Utils.js` – does not contain public functions to call
- `HybridApp.js` – does not contain public functions to call
- `json2.js` – third-party library. For information about the functions in this library, see the JSON documentation at <http://json.org>
- `cordova-2.0.0.javascript` – contains PhoneGap APIs. For information about PhoneGap APIs, see the documentation at www.phonegap.com.

API.js

The `API.js` file contains several different types of functions.

They include:

- General and Hybrid App utility functions
- Validation functions
- Credential functions

Hybrid App UI Functions

Functions that allow you to access the Hybrid App user interface (UI).

updateUIFromMessageValueCollection

To completely override the behavior provided by

updateUIFromMessageValueCollection for a given screen, provide a UIUpdateHandler object for that screen. That UIUpdateHandler object has a screenName property, which indicates which screen's behavior it is overriding, and a callback function that indicates the function to call for that screen. That function is passed in the relevant MessageValueCollection object and it is its responsibility to update the controls' values based on its contents. An example of this is:

```
function MyListViewUpdateHandler() {
    this.screenName = "Prev_Expenses";
    this.values;
}

MyListViewUpdateHandler.prototype.callback = function(valuesIn)
{
    // Rows returned from RMI Call
    this.values = valuesIn;

    // construct our table
    try {
        var mvc =
this.values.getData("PurchaseTrackingJC_findOtherRequests_resultSet
Key");
        var txt = "";
        var htmlOut = "<p>";

        // Do we have any rows to display?
        if (mvc.value.length > 0) {
            // Start the table and header
            htmlOut += "<table id='MyPrevExpensesTable'
class='altrowstable'>";
            htmlOut += "<tr><th>Item Name</th><th>Cost</th></tr>";

            // Draw the rows+H15
            for (var rows = 0; rows < mvc.value.length; rows++) {
                var mvName =
mvc.value[rows].getData("PurchaseTrackingJC_itemName_attribKey");
                var mvCost =
mvc.value[rows].getData("PurchaseTrackingJC_itemCost_attribKey");

                if (mvName && mvCost) {
                    // Alternate the row colors
                    htmlOut += "<tr
onclick='navigateForward(\"Prev_Expenses_Detail\", \" +
mvc.value[rows].getKey() + \");'>";
                    if (rows % 2 == 0) {
                        htmlOut += " class='evenrowcolor'>";
                    }
                }
            }
        }
    }
}
```

```

        else {
            htmlOut += " class='oddrowcolor'>";
        }

        htmlOut += "<td>" + mvName.getValue() + "</td><td>" + mvCost.getValue(); + "</td></tr>";
    }
}

// Finish the table
htmlOut += "</table>";
}
else {
    htmlOut += "No rows returned.";
}
htmlOut += "</p>";

//Now add the table to the document
var form = document.forms[curScreenKey + "Form"];
if (form) {
    //var topOfFormElem = document.getElementById("topOf" +
curScreenKey + "Form");

    var topOfFormElem =
document.getElementById("PurchaseTrackingJC_findOtherRequests_resultSetKey");
    topOfFormElem.innerHTML = htmlOut;
}

}
catch (e) {
    alert(e.message);
}
} // function callback

function customAfterWorkflowLoad() {
    //Setup UIHandler to draw our Listview Screen
    UIUpdateHandlers[0] = new MyListViewUpdateHandler();
}

```

Hybrid App Native Device Functions

Access the native features of the device using the native device functions.

showUrlInBrowser(url)

To have a hyperlink in the default value for the HtmlView control, or for doing customization in Javascript, follow the **showUrlInBrowser** method without using standard HTML. To add HTML in the default value for the HtmlView control, you can use something similar to:

```

<html>
<body>
<b>Welcome</b><br>
<br>Your activation was successful, the newly created Hybrid App
requests will automatically be pushed to you.<br>
<br>For more information contact your administrator or visit us

```

```
at:<br>
<br>
<a href="javascript:showUrlInBrowser('http://www.sap.com/
unwiredenterprise')">SAP Mobile Platform</a>
</body>
</html>
```

View an attachment such as an image, a Word document, a PDF file, and so on as part of the Hybrid App package. This example uses an image file.

1. Generate the Hybrid App package and its files.
2. In WorkSpace Navigator, go to the location where the generated Hybrid App files are located and add an images folder under the html folder, for example, Generated Hybrid App\<hybridapp_name>\html\images.
3. Copy an image to the images folder.
4. In the Hybrid App Designer, add a menu item to the Hybrid App.
5. Open the Custom.js file with a text editor and edit the method customBeforeMenuItemClick:

```
if (screen === "ScreenKeyName" && menuItem === "ShowAttachment") {
    showLocalAttachment("html/images/ipod.jpg");
    return false;
}
```

6. Save and close the Custom.js file.
7. Deploy the Hybrid App package to SAP Mobile Server.

Hybrid App Message Data Functions

Access the Hybrid App message data functions.

A Hybrid App has an in-memory data structure where it stores data. This data is used to update the controls on the screen through `updateUIFromMessageValueCollection()`. Values are extracted from those controls and used to update the data through `updateMessageValueCollectionFromUI()`.

You can program the data content and use it to make decisions on the client. To get the active instance of this data structure, you start by calling `getDataMessage()`. This returns a `WorkflowMessage` object. This object has a function, `getValues()`, that is used to return the top-level `MessageValueCollection` object. This object has a list of key-value pairs, represented by `MessageValue` objects and is retrieved by calling `getData(key)`. `getData()` returns either a single `MessageValue` object, or an array of `MessageValueCollection` objects.

A typical Hybrid App message might look similar to this.

```
WorkflowMessage
  .getHeader()           <undefined>
  .getWorkflowScreen()   "salesorderList_newSOCreate"
  .getRequestAction()    "Submit_Workflow"
  .getValues()           MessageValueCollection
  .getData("salesorderList_newSOCreate_WITHOUT_COMMIT_paramKey")
```



```

        .getKey()
"salesorderList_newSOCreate_WITHOUT_COMMIT_paramKey"
        .getType()      "TEXT"
        .getValue()     "1"
        .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_DOC_TYPE_attribKey")
        .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_DOC_TYPE_attribKey"
        .getType()      "TEXT"
        .getValue()     "1"
        .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_SALES_ORG_attribKey")
        .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_SALES_ORG_attribKey"
"
        .getType()      "TEXT"
        .getValue()     "1"
        .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_DISTR_CHAN_attribKey")
        .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_DISTR_CHAN_attribKe
y"
        .getType()      "TEXT"
        .getValue()     "1"
        .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_DIVISION_attribKey")
        .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_DIVISION_attribKey"
        .getType()      "TEXT"
        .getValue()     "1"
        .getData("salesorderList_newSOCreate_ORDER_PARTNERS_para
mKey")
        MessageValue
        .getKey()
"salesorderList_newSOCreate_ORDER_PARTNERS_paramKey"
        .getType()      "LIST"
        .getValue()     MessageValueCollection[]
        [0].getKey()    "6476c1a4-94e9-e5a4-b903-
caf2ca613c4a"
        [0].getState()  "add"
        [0].getData("PARTN_ROLE")
        MessageValue
        .getKey()      "PARTN_ROLE"
        .getType()     "TEXT"
        .getValue()    "1"
        [0].getData("PARTN_NUMB")
        MessageValue
        .getKey()      "PARTN_NUMB"
        .getType()     "TEXT"
        .getValue()    "1"

```

getCurrentMessageValueCollection

Handling individual items

Develop a Hybrid App Using the Hybrid App Designer

```
var message = getCurrentMessageValueCollection();

var cityObj = message.getData("Customer_city_attribKey");
var city = cityObj.getValue();

var stateObj = message.getData("Customer_state_attribKey");
var state = stateObj.getValue();

var zipObj = message.getData("Customer_zip_attribKey");
var zip = zipObj.getValue();
```

List

```
var message = getCurrentMessageValueCollection();
var itemList = message.getData("CustDocs");

var items = itemList.getValue();
var noOfItems = items.length;
var i = 0;

while (i < noOfItems) {
    var theItems = items[i];
    var
fileNameObj=theItems.getData("CustDocs_fileName_attribKey");
    var fileName = fileNameObj.getValue();
    i = i + 1;
}
```

Callbacks.js File

This file contains callback functions.

Callback functions are typically used for event handlers that are asynchronous.

Camera.js

These functions allow you to take a picture from the camera, or pick one from the photo library and use the picture in the Hybrid App.

getPicture Function

The `getPicture` function provides access to the device's default camera application or device's photo library for retrieving a picture asynchronously.

If the `SourceType` is `CAMERA` or `BOTH`, the `getPicture` function opens the device's default camera application (if the device has a camera) so the user can take a picture. Once the picture is taken, the device's camera application closes and the Hybrid App is restored. If the device does not have a camera application, the function reports that it is not supported.

Using the getPicture Function for Larger Image Sizes

For larger images, use the `IMAGE_URI` destination type.

For larger images, use the `IMAGE_URI` destination type. The MIME type for the image URI is determined using the extension of the file name parameter in the `onGetPictureSuccess`

callback. You must add this extension information to the Hybrid App message as a separate `MessageValue` to use it on the server. For the HTML image tags, the browser should be able to determine the type through the HTTP connection opened on the URI.

You must create a new option object similar to this:

```
var options = { destinationType:
PictureOption.DestinationType.IMAGE_URI,
                sourceType: PictureOption.SourceType.CAMERA
            };

getPicture(onPictureError, onPictureSuccess, options);
```

The `destinationType` can be `PictureOption.DestinationType.IMAGE_DATA` (Base64 string behavior), or the new `PictureOption.DestinationType.IMAGE_URI` type. Depending on the destination type specified, the picture success callback's second parameter may be a Base64 string or a URI. The source type can be `PictureOption.SourceType.CAMERA`, `PictureOption.SourceType.PHOTOLIBRARY`, or `PictureOption.SourceType.BOTH`.

The image URI passed back is expected to be valid and resolvable to the image by the browser. You can create an HTML image tag with a URI to display the image, for example, ``. This can also be used to create thumbnails.

Uploading the Image to the Server for a URI

To upload the image to the server for a URI, you must create a `MessageValue` in the JavaScript with a "FILE" type. When the JavaScript Hybrid App message is serialized it will identify if the message contains files. During a submit or online request, the query sent to the container will contain a new query parameter that identifies that this message must be parsed again. The query looks similar to: `?querytype=submit&parse=true`.

Note: When you upload a large image to the server using an online request, rather than a submit Hybrid App, the image contents come back from the online request, which can result in too large of a Hybrid App message for the container to handle. It is recommended that you use the submit action instead of online request action when it is likely that the message size will be very large, such as when it includes large images.

The custom code must call the function

```
getDataMessage().setHasFileMessageValue(true);
```

for the parse query to be sent to the container.

When uploading the image to the server for a URI, the JavaScript looks similar to this example:

```
var options = { destinationType:
PictureOption.DestinationType.IMAGE_URI, sourceType:
PictureOption.SourceType.PHOTOLIBRARY };

getPicture( onGetPictureError, onGetPictureSuccess, options );

function onGetPictureSuccess(fileName, imageUrl){
```

```
// Set file for upload
var fileDataKey = "Picture_create_fileData_paramKey";

var messageValue =
getDataMessage().getValues().getData(fileDataKey);

if (messageValue)
{
    // Update file for upload
    messageValue.setValue(imageUri);
}
else
{
    // Add file for upload
    messageValue = new MessageValue();
    messageValue.setKey(fileDataKey);
    messageValue.setValue(imageUri);
    messageValue.setType(MessageValueType.FILE);
    getDataMessage().getValues().add(fileDataKey, messageValue);
}

getDataMessage().setHasFileMessageValue(true);
}
```

Handling a larger image size example:

```
function reportError(errCode)
{
    if (errCode != PictureError.USER_REJECT) {
        // error occurred
    }
}

function reportImage(fileName, imageUri)
{
    // Image captured
    alert("Photo taken");

    // Optional - Display preview in image tag
    var imageTagId = "Thumbnail"; // The id of your image tag
    var imageElement = document.getElementById(imageTagId);
    imageElement.src = imageUri;

    // Optional - Create message value to upload image
    var fileKey = "Picture_create_fileData_paramKey"; // Key that
maps to submit or online request parameter
    var messageValue = new MessageValue();
    messageValue.setKey(fileKey);
    messageValue.setValue(imageUri);
    messageValue.setType(MessageValueType.FILE);

    // Add message value to Workflow message - NOTE: Code may differ
dependent on the context for adding image (Eg. ListView).
    getDataMessage().getValues().add(fileKey, messageValue);

    getDataMessage().setHasFileMessageValue(true); // Explicitly
}
```

```
tell Workflow about image
}
var options = { destinationType:
PictureOption.DestinationType.IMAGE_URI, sourceType:
PictureOption.SourceType.CAMERA};
getPicture( onGetPictureError, onGetPictureSuccess, options );
```

Limitations

The server has a limit of 75MB per parameter, which is what the Hybrid Web Container uses as the `XmlWorkflowMessage`. Therefore, the server imposes a maximum size limit of 50 MB (assuming one picture per `XmlWorkflowMessage`, and no other keys are present). Keep in mind that clients may impose a lower limit than 50MB.

Note: When accessing very large binary (image) data in the mobile business object associated with the Hybrid App, ensure that the attribute set in the mobile business object is a **BigBinary** datatype, rather than Binary.

Certificate.js

Provides functions for X.509 credential handling.

Use these functions to create a user interface in HTML and JavaScript, that uses X.509 certificates as the Hybrid App credentials.

This file contains the functions that allow parsing a certificate date, creating a certificate from a JSON string value, retrieving a certificate from a file (Android), retrieving a certificate from the server (iOS), and so on.

You can choose to set the results of a `getSignedCertificate` function as the password.

```
certificateLabels(filterSubject, filterIssuer)

// The following script gets all the labels for certificates
// with the provided subject and issuer
var certStore = CertificateStore.getDefault();
var labels = certStore.certificateLabels("MyUser", "mydomain.com");

- getPublicCertificate(label)

// The following script gets the certificate data for the first
// certificate to match the provided subject and issuer
var certStore = CertificateStore.getDefault();
var labels = certStore.certificateLabels("MyUser", "mydomain.com");
var cert = certStore.getPublicCertificate(labels[0]);

- getSignedCertificate(label)

// The following script gets the signed certificate data for the
// first
// certificate to match the provided subject and issuer
var certStore = CertificateStore.getDefault();
var labels = certStore.certificateLabels("MyUser", mydomain.com");
var cert = certStore.getSignedCertificate(labels[0]);
```

```
var username = cert.subjectCN;
var password = cert.signedCertificate;

- listAvailableCertificatesFromFileSystem(sFolder, sFileExtension)

// The following script gets an array of file paths for files on
// the sdcard with the extension p12
var certStore = CertificateStore.getDefault();
var certPaths = certStore.listAvailableCertificatesFromFileSystem("/sdcard/", "p12");

- getSignedCertificateFromFile(filePath, password)

// The following script gets the signed certificate data for the
first
// p12 file found on the sdcard
var certStore = CertificateStore.getDefault();
var certPaths = certStore.listAvailableCertificatesFromFileSystem("/sdcard/", "p12");
var cert = certStore.getSignedCertificateFromFile(certPaths[0],
"password");

- getSignedCertificateFromServer(username, serverPassword,
certPassword)

// The following script gets the signed certificate data for the
// user MYDOMAIN\MYUSERNAME from the server
var certStore = CertificateStore.getDefault();
cert = certStore.getSignedCertificateFromServer("MYDOMAIN\
MYUSERNAME", "myserverpassword", "mycertpassword");
```

Custom.js File

The first time you generate the Hybrid App package files, the `Custom.js` file is generated.

In subsequent file generations for the same Hybrid App package, this file will not be overwritten, so any customizations you make are preserved.

These touch points are available for customization: `WorkflowLoad`, `Submit`, `NavigateForward`, `NavigateBackward`, `ShowScreen`, `MenuItemClick`, and `Save`. At each touch point, a **customBefore** method is invoked and a **customAfter** method is invoked. The `customBefore` method returns a boolean. If it returns true, it continues to execute the default behavior, for example, navigating to a new screen or performing an online request. If it returns false, it does not execute the default behavior, so you can override the default behavior by customizing these methods.

The `Custom.js` file contains these methods:

Note: You can delegate the implementation of these functions to different functions supplied in other custom JavaScript files. It is not necessary to include all of your customization logic in the single `Custom.js` file.

```
//Use this method to add custom html to the top or bottom of a form
function customBeforeWorkflowLoad() {

    var form = document.forms[curScreenKey + "Form"];
    if (form) {
        // header
        var topOfFormElem = document.getElementById("topOf" +
curScreenKey + "Form");

        if (topOfFormElem) {
            topOfFormElem.innerHTML = "<img id='ImgSylogo' src='./
images/syLogo.gif'/><br/>";

            // footer
            var bottomOfFormElem = document.getElementById("bottomOf"
+ curScreenKey + "Form");
            bottomOfFormElem.innerHTML = "<p>Copyright 2010, Sybase
Inc.</p>";
        }
    }
    return true;
}
```

When using the **customBeforeNavigateForward(screenKey, destScreenKey) {}** function, if you want to create your own JQuery Mobile style listview, remember that JQueryMobile does not allow duplicate ID attributes. So if there is an existing listview with the same ID attribute, you must:

1. Delete the existing listview with the same ID attribute.
2. Re-create the listview.
3. Call **refresh** for your listview.

For example:

```
//Use this method to add custom code to a forward screen transition.
If you return false, the screen
//transition will not occur.
function customBeforeNavigateForward(screenKey, destScreenKey) {

..
try {
    if (destScreenKey == 'Personal_Work_Queue') {

        //grab the results from our object query
        var message = getCurrentMessageValueCollection();
        var itemList = message.getData("PersonalWorkQueue");
        var items = itemList.getValue();
        var numOfItems = items.length;
        var i = 0;

        //iterate through the results and build our list
        var htmlOutput = '<div id="CAMSCustomViewList"><ul data-
role="listview" data-filter="true">';
        var firstOrder = '';
```

```

        while ( i < numOfItems ){
            var currItem= items[i];
            var opFlags =
currItem.getData("PersonalWorkQueue_operationFlags_attribKey").getV
alue();
            var orderId =
currItem.getData("PersonalWorkQueue_orderId_attribKey").getValue();
            var operationNumber =
currItem.getData("PersonalWorkQueue_operationNumber_attribKey").get
Value();
            var description =
currItem.getData("PersonalWorkQueue_description_attribKey").getValu
e();
            try {
                var promDate =
currItem.getData("PersonalWorkQueue_datePromised_attribKey").getVal
ue();
            } catch (err) {
                var promDate = "";
            }

            try {
                var planDate =
currItem.getData("PersonalWorkQueue_dateStartPlan_attribKey").getVa
lue();
            } catch (err) {
                var planDate = "";
            }

            var onHold =
currItem.getData("PersonalWorkQueue_onHold_attribKey").getValue();

            htmlOutput += '<li><a id ="' + currItem.getFullKey() +
'" class="listClick">';
            htmlOutput += '<p><b>Flags: </b>' + opFlags + '</p>';
            htmlOutput += '<p><b>Order Id: </b>' + orderId + '</
p>';
            htmlOutput += '<p><b>Operation No: </b>' +
operationNumber + '</p>';
            htmlOutput += '<p><b>Title: </b>' + description + '</
p>';
            htmlOutput += '</a></li>';

            i++;

        }

        htmlOutput += '</ul></div>';

//append the html to the appropriate form depending on the
key
        if (destScreenKey == 'Personal_Work_Queue') {
            var listview = $('div[id="CAMSCustomViewList"]');
            //Try to remove it first if already added
            if (listview.length > 0) {

```



```

        var ul = $(listview[0]).find('ul[data-
role="listview"]');
        if (ul.length > 0) {
            htmlOutput = htmlOutput.replace('<div
id="CAMSCustomViewList"><ul data-role="listview" data-
filter="true">', '');
            ul.html(htmlOutput);
            ul.listview('refresh');
        }
    } else {
        $('#Personal_Work_QueueForm').children().eq(2).hide();
        $('#Personal_Work_QueueForm').children().eq(1).after(htmlOutput);
    }
}
//add the listener based on the class added in the code
above
$(".listClick").click(function(){
    currListDivID = $(this).parent().parent();
    $(this).parent().parent().addClass("ui-btn-active");

    //special case for bb
    navigateForward("Shop_Display", this.id );

    if (isBlackBerry()) {
        return;
    }
});
}

```

Overriding the showErrorFromNative Function

The generated JavaScript allows you to override the behavior of the showErrorFromNative function using the customBeforeReportErrorFromNative(errorString) and customAfterReportErrorFromNative(errorString) methods.

This shows an example of how to override or customize the error message based on the returned numeric error codes through customBeforeReportErrorFromNative.

```

function customBeforeReportErrorFromNative(errorString) {
    var errorCode = getURLParamFromNativeError("errCode",
errorString);
    // 500 and above are network errors
    if ( errorCode >= 500 )
    {
        // Could check lang global variable if so desired
        //if ( lang == ... )
        {
            // Show your own custom error message based on errorCode
            showAlertDialog("Do you have a network connection?", "My
custom error");
            // return false to by pass default behavior
            return false;
        }
    }
}

```

```
}  
    return true;  
}
```

Identified error scenarios include:

- Any network related errors during an online (synchronous) request contain an error code of 500 or greater (check for ≥ 500)
- `public static final int UNKNOWN_ERROR = 1; // "unknown error"`
- `public static final int ATTACHMENT_NOT_DOWNLOADED = 100; // "Attachment has not been downloaded"`
- `public static final int UNKNOWN_MIME_TYPE = 101; // "Unknown MIME type"`
- `public static final int FILENAME_NO_EXTENSION = 102; // "File name without extension"`
- `public static final int REQUIRED_PARAMETER_NOT_AVAILABLE = 103; // "Required parameter is not available"`
- `public static final int UNSUPPORTED_ATTACHMENT_TYPE = 105; // attachment type is not supported"`
- `public static final int SSOCERT_EXCEPTION = 106; // SSO Certificate manager exception`
- `public static final int FAIL_TO_SAVE_CREDENTIAL = 107; // Fail to save credential`
- `public static final int FAIL_TO_SAVE_CERTIFICATE = 108; // Fail to save certificate`
- `public static final int DEVICE_NOT_CONNECTED = 109; // Device is not connected`

Resources.js

The resource functions allow you to access localized string resources.

ExternalResource.js

These functions allow you to access resources on external HTTP servers.

This shows an example of the UPDATE function:

```
function update() {  
    // Using json to update a value  
    var url = // URL of your external resource;  
    var webResponse;  
    var options = {  
        method: "PUT",  
        data: "{\"Value\":\"Value A Updated\"}",  
        headers: {  
            "Content-type": "application/json"  
        },  
    },
```

```

        async: false,
        complete: function(response) { webResponse = response; }
    };

    getExternalResource(url, options);

    if (webResponse.status === 200)
        alert("Update successful");
    else
        alert("Update Failed");
}

```

This shows an example of the DELETE function:

```

function delete() {
    // Delete a value
    var url = // URL of your external resource;
    var webResponse;
    var options = {
        method: "DELETE",
        async: false,
        complete: function(response) { webResponse = response; }
    };

    getExternalResource(url, options);

    if (webResponse.status === 200)
        alert("Delete successful");
    else
        alert("Delete Failed");
}

```

SUPStorage.js

Functions to store results from online requests in a specified cache.

Storage functions enable you to:

- Name the cached result sets
- Enumerate the cached result sets
- Read, delete, and modify cached contents individually for each cached result set

Usage Notes: PhoneGap must be initialized before a storage function is called. The initialization happens automatically when you generate code using the Hybrid App Designer; if you do not use Designer, you must detect PhoneGap initialization in your own code. See *Implementing PhoneGap*.

Cached result sets must be stored as strings (before deserialization to an `xmlWorkflowMessage` structure).

Calls to these methods do not trigger events.

Example: Constructors

```

// These constructors create two local storage instances with their
own domain

```

Develop a Hybrid App Using the Hybrid App Designer

```
var store1 = new hwc.SUPStorage("mydomain");
var store2 = new hwc.SUPStorage("myotherdomain");

// This constructor creates a shared storage instance whose key is
// the one set in the
// packaging tool for generated JavaScript API, or in the Hybrid App
// Designer.
var storeS = new hwc.SharedStorage();
```

Example: length

```
// Displays the current number of elements in the storage
var store = new hwc.SUPStorage();
alert(store.length());
```

Example: key(index)

```
// Displays the value at the provided index in the storage
var store = new hwc.SUPStorage();
alert(store.key(2));
```

Example: getItem(key)

```
// Displays the value for the provided key
var store = new hwc.SUPStorage();
alert(store.getItem("mykey"));
```

Example: setItem(key, value)

```
// Sets a key/value pair
var store = new hwc.SUPStorage();
store.setItem("mykey", "myvalue");
```

Example: removeItem(key)

```
// Removes a key/value pair
var store = new hwc.SUPStorage();
store.removeItem("mykey");
```

Example: clear

```
// Clears the storage
var store = new hwc.SUPStorage();
store.clear();
```

SAP Mobile PlatformStorage

The SAP Mobile Platform Storage API allows you to store structured data on the client side.

You can also use these functions as an arbitrary key or value storage mechanism. Keys are strings, and any string (including the empty string) is a valid key. Keys cannot be duplicated in the same Hybrid App package. Values are also strings and values can be duplicated in the same Hybrid App package. Keys and values can contain multi-byte characters.

SUPStorage can span multiple screens in the Hybrid App, and lasts beyond the current session. This allows the storage of user data on the client, such as entire user-authored documents.

Using platform-specific mechanisms, the items stored using the `SUPStorage` API are encrypted according to the particular platform policies:

Platform	Encryption policy
BlackBerry	<code>PersistentStore</code> , which adheres to the Content Protection BES IT policy
Android	Encrypted before storing into the SQLite database
iOS	Stored in SQLite Encryption Extensions database
Windows Mobile	Unencrypted SQLite—security is deferred to Afaria Security Manager

The amount of data that can be stored on the client is limited only to the available storage space on the particular platform:

Platform	Data storage
BlackBerry	Amount of free <code>PersistentStore</code> .
iOS and Android	Amount of free file system for the SQLite database, and/or the SQLite database size limit
Windows Mobile	Amount of free file system, and the SQLite database size limit.

Limitations

- The amount of data that you can retrieve and return to the JavaScript space when using the `SUPStorage` API is limited to the JavaScript size limitation as established for each platform. See the topic *AttachmentViewer and Image Limitations* in *SAP Mobile WorkSpace - Hybrid App Package Development*.
- On Windows Mobile devices, there is a 500K limitation for the length of the shared storage item. If the length of a shared item is more than 500K, the JavaScript does not accept anything.
- Physical SAP Mobile Platform storage is tied to a Hybrid App package. When the Hybrid App package is uninstalled, the corresponding SAP Mobile Platform storage for the Hybrid App package is removed immediately.
- Items stored using the `SUPStorage` API are persisted, and therefore, survive soft device resets.
- `SUPStorage` persists through invocations of the Hybrid App.
- The `SUPStorage` API does not restrict reading or writing of the storage data from different domains. For example, if a Hybrid App loads some code from an external HTTP server that attempts to access the `SUPStorage` API, it is allowed.

Develop a Hybrid App Using the Hybrid App Designer

- The `SUPStorage` API does not take into account the current locale or language of the device. You can, however, access the global JavaScript variable called *lang* and implement this in your custom code.

Shared Storage

All Hybrid Apps with a shared storage key assigned share the storage with other Hybrid Apps that have the same storage key assigned.

- When the last Hybrid App with the shared storage key is removed from the device, the storage data is also removed.
- Since shared storage data is loaded into JavaScript, the same limitations apply to it as that which applies to the JavaScript size limitation as established for each platform. See the topic *Attachment Viewer and Image Limitations*. If a large amount of data is involved in the operation, the shared storage should be used only to store the reference or location of the data, not the data itself. This helps to ensure you stay within the JavaScript size limitations. For example, if data for an image needs to be saved in shared storage for later use, the image data should be stored in the device file system or the persistent store, and then store only the file path to the shared storage.
- Shared storage items are removed when the last Hybrid App using the same shared storage key is removed from the device (it happens on unassignment)
- On Windows Mobile devices, there is a 500K limitation for the length of the shared storage item. If the length of a shared item is more than 500K, the JavaScript does not accept anything.

Timezone.js

The date/time functions allow you to extract and format the date and time for the Hybrid App.

WorkflowMessage.js

Use these functions to access message resources.

Using Third-Party JavaScript Files

To include your own files in Hybrid Web Container, copy them into the appropriate place in the Generated Hybrid Apps folder.

To load external JavaScript and CSS files dynamically, copy the relevant third-party JavaScript and CSS files to the Generated Hybrid Apps\`<package_name>\html` and `js` or `css` folders. If the files are JavaScript files, and are in the `html\js` folder, they are automatically included in the HTML as script.

Note: On Android, individual HTML, JavaScript, and CSS files cannot exceed 1MB.

These files will be included in the Hybrid App `manifest.xml` and ZIP files automatically when the Hybrid App package is regenerated.

Repackaging Hybrid App Package Files

After modifying the `Custom.js` file, you must redeploy the Hybrid App package to SAP Mobile Server.

1. Save and close the modified files after adding your custom code.
2. In Workspace Navigator, right-click the `<hybrid_app_name>.xbw` file and select **Generate Hybrid App**.
3. In the Hybrid App generation wizard, select the connection profile.
4. In Generation Options, choose:
 - Generate Package
 - Update generated code
 - Deploy to an SAP Mobile Server as a replacement
5. Click **Finish**.

Common Customizations

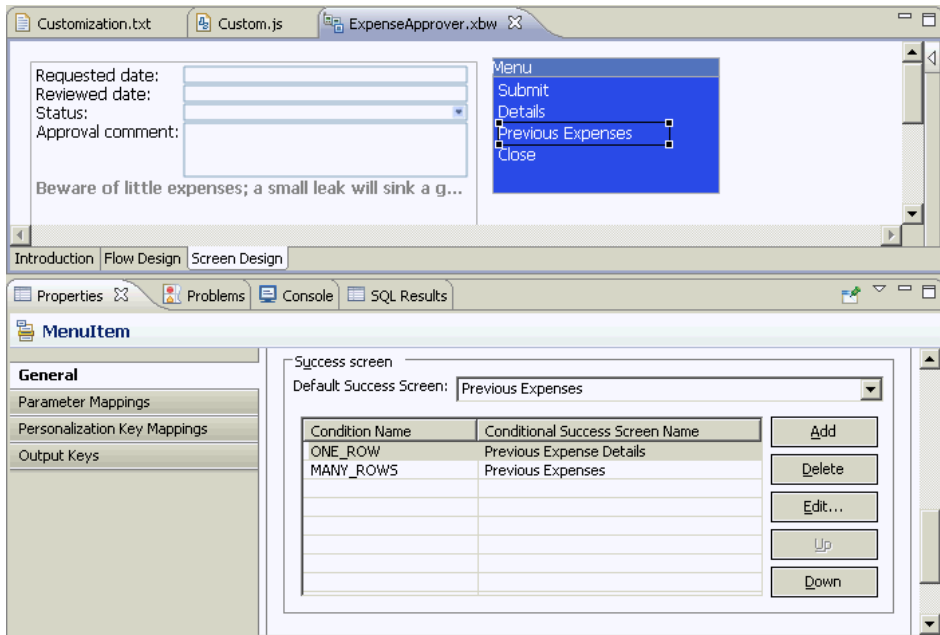
Implementing Conditional Navigation

Conditional navigation allows you to implement a custom function that allows you to override navigation behavior between screens.

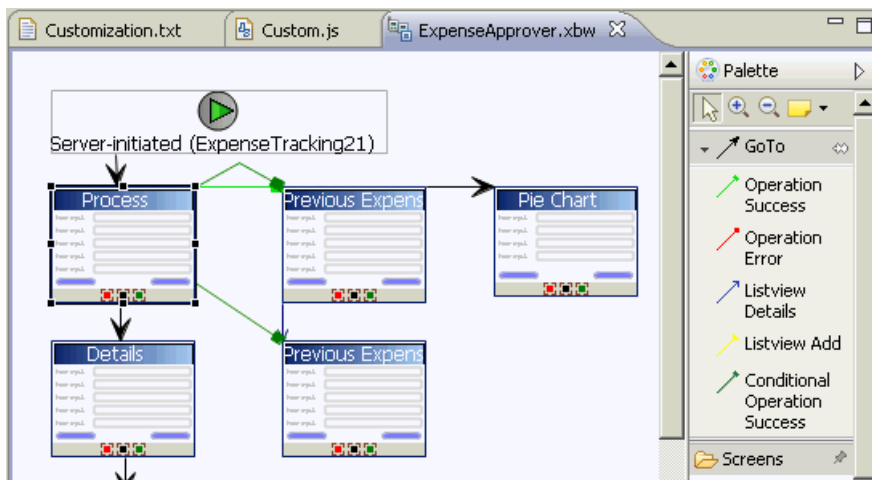
This procedure gives an example of how you can use conditional navigation to skip a screen.

1. In the Screen Design page, modify the menu item by adding conditions.
In this example, two conditions are added to the Previous Expenses menu item.

Develop a Hybrid App Using the Hybrid App Designer



- Go to the Flow Design page to see the conditional navigation paths in the flow.



- In the Custom.js file, add the custom code for conditional navigation.

```
//This example demonstrates the conditional navigation
functionality for an online request.
//In this example we skip the list view screen and go directly to
the details screen if there is only one item in the list
function customConditionalNavigation(currentScreenKey,
actionName, defaultNextScreen, conditionName, workflowMessage) {
    if ((currentScreenKey === 'Process') && (actionName ===
'Previous Expenses')) {
```



```

        if (conditionName === 'ONE_ROW') {
            var values = workflowMessage.getValues();
            var m = workflowMessage.serializeToString();
            var expenseTracking =
values.getData("ExpenseTracking21View");
            var etList = expenseTracking.getValue();
            var count = etList.length;
            if (count == 1) {
                var etRow1 = etList[0];
                workflowMessage.updateValues(etRow1);
                return true;
            }
        }
        else if (conditionName === 'MANY_ROWS') {
            return false; //ie do the normal navigation which is
to go to the listview screen
        }
    }
    // default case is to NOT change the flow
    return false;
}

```

4. Use the Hybrid App Generation wizard to re-generate the Hybrid App package with a new `hybridapp_jQueryMobileLookAndFeel.html` file that contains the newly added conditional navigations.
5. Use a browser to debug the code.

Implementing a Conditional Start Screen

Add conditions that determine which start screen the user sees based on the conditions.

Like the conditional success navigation feature, there is a table of condition names with the matching Start screen. If all of the conditions are evaluated as false (or if they are absent), the default navigation is executed.

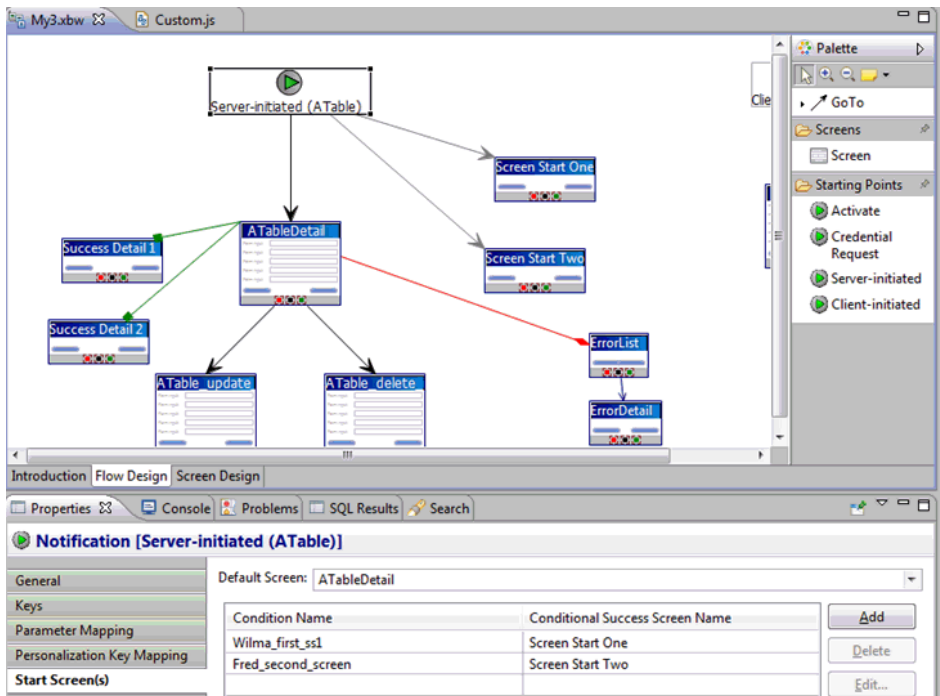
1. In the Flow Design page, select the server-initiated starting point to see the Properties.
2. In the Properties view, click **Start Screen(s)**.
3. Click **Add** to add a condition.
4. In the dialog, enter the condition name, select the target screen with which to associate the condition, and click **OK**.

This means that if the defined condition is found to be true, the screen you choose here will be the start screen. Condition names can include:

- Letters A-Z and a-z
- Numbers 0-9
- Embedded spaces (beginning and ending spaces are trimmed off)
- Special characters in the set `$. _ - +`

In the Flow Design page, you can see the flow line for the conditional start is a shade of gray to differentiate it from the default GoTo line.

Develop a Hybrid App Using the Hybrid App Designer



5. Add you custom code to the Custom.js file. For example:

```
function customConditionalNavigation( currentScreenKey,
actionName,
defaultNextScreen, conditionName,
workflowMessage ) {
    if((currentScreenKey === SERVERINITIATEDFLAG) && (actionName
=== '')) {
        // conditional start screen uses this magic screen key and
the empty action name.
        if( conditionName === 'Wilma_first_ss1') {
            // custom logic
            return true;
        }
        else if(conditionName === 'Fred_second_screen'){
            // custom logic
            // return true or false
            return false;
        }
    }
    // default case is to NOT change the flow
    return false;
}
```

6. Regenerate the Hybrid App package.

When you regenerate the Hybrid App package, the `hybridapp.js` file is regenerated. The conditional start screen method is shown in the `hybridapp.js` file similar to this:

```
function customNavigationEntry() {
    this.condition;
    this.screen;
}
function customNavigationEntry( a_condition, a_screen ) {
    this.condition = a_condition;
    this.screen = a_screen;
}

/**
 * For the specific pair - screen named 'currentScreenKey' and the
 * action 'actionName', return
 * the list of custom navigation condition-names and their
 * destination screens.
 */
function getCustomNavigations( currentScreenKey, actionName ) {
    var customNavigations = new Array();
    if((currentScreenKey === SERVERINITIATEDFLAG) && (actionName
=== '')) {
        customNavigations[0] = new
customNavigationEntry( 'Wilma_first_ssl',
'Screen_Start_One' );
        customNavigations[1] = new
customNavigationEntry( 'Fred_second_screen',
'Screen_Start_Two' );
        return customNavigations;
    }
    return customNavigations;
}
```

Clearing the Contents of the Signature Control

Add JavaScript to clear the contents of a signature control.

1. Use the Hybrid App Generation wizard to generate the Hybrid App package and its files.

When the Hybrid App package is generated, the Custom.js file is generated if not already present in the project. The Custom.js file is located in Generated Hybrid App\<project_name\html\js.

2. Open the Custom.js file and add your JavaScript code to the click event of a menu or button.

For example:

```
function customAfterMenuItemClick(screen, menuItem) {
    if (menuItem === "Clear_Signature") {
        $.data(document.getElementById('sigKey'),
'signature').clearSignature();
    }
}
```

3. Save and close the Custom.js file.
4. Re-generate the Hybrid App package and deploy it to SAP Mobile Server.

Security

Set up static or dynamic authentication, and configure the Hybrid App to use credentials.

Credentials

You can use either dynamic or static credentials in a Hybrid App screen flow.

See *Security* and *System Administration* for more detailed information about implementing security and certificates.

The user name and password values are required when the Hybrid App invokes a mobile business object operation. These authentication values can be provided statically (at design time), or dynamically (by the user at runtime). For requests sent by the client with a credential screen specified, requests are always invoked on the server using the credentials specified by the user, regardless of whether static or dynamic authentication is specified.

The choice of static versus dynamic authentication applies only to requests that must be executed on the server that do not have any credentials, or that do not have valid credentials. This happens when an object query needs to be run by a server-initiated notification, for example, or if the client provides incorrect credentials. In that scenario, the decision between static and dynamic becomes important. If static was chosen, it silently uses those hard-coded credentials. If dynamic was chosen, it sends a notification to the client and asks the user to supply the credentials.

For example, you might define a server-initiated Hybrid App with a credential screen and static authentication. When the notification first comes in, it runs an object query using the hard-coded credentials. This is then sent to the user, who opens the notification and then makes an online request. This online request, be it an operation or an object query, will be made using the credentials supplied by the user.

Dynamic credentials require the user to enter the user name and password on a screen that the credential request starting point references. Select **Credential Cache User Name** and **Password** to indicate the user name and password to be required on the client. When the user logs in, the credentials are authenticated using the stored credentials.

Note: If an e-mail triggered Hybrid App has dynamic cached credentials, the cached credentials are not cached between invocations of the Hybrid App form through an e-mail trigger.

Static credentials mean that everyone who has access to the resource uses the same user name and password. By default, static credentials are used. The static credential user name and password for the Hybrid App can be extracted from the selected SAP Mobile Platform profile user name and password when the Hybrid App is generated, or they can be hard-coded using the Properties view. After deployment, you can change static credentials in SAP Control Center.

The application can also have a credential screen (Credential Request) that appears if the Hybrid App detects that the cached credentials are empty or incorrect.

Setting Up Static Authentication

With static authentication, everyone who has access to the resource uses the same user name and password.

Set up static credentials in the Authentication section of the Properties tab. To see the Properties page, verify there are no objects selected on the Flow Design page.

1. In the Properties view, click **Authentication**.
2. Select **Use static credentials**.
3. Select from these options:
 - **Use SAP Mobile Server connection profile authentication** – specifies that the user name and password associated with the connection profile are used when code is generated for the Hybrid App. Selected by default.
 - **Use hard-coded credentials** – sets the user name and password. When you select this option, the User name and Password fields are activated.
 - **Use certificate-based credentials** – enables you to use a certificate to generate authentication credentials.
4. (Optional) If you select **Use hard-coded credentials** in the previous step, enter the **User name** and **Password** that are to be used for authentication.
5. Select **File > Save**.

Setting Up Static Authentication Using a Certificate

Set up static authentication credentials generated from a certificate.

1. In the Properties view, click **Authentication**.
2. Select **Use static credentials** and **Use certificate-based credentials**.
3. Click **Generate from Certificate** to select a certificate file from which to generate authentication.
4. In the Certificate Picker, click **Browse** to locate the certificate to use.
5. Enter a password and select an alias, then click **OK**.

The information from the certificate is shown in the Properties view.

- **Issuer** – the issuer of the certificate
 - **Subject** – the value of the subject field in the metadata of the certificate as defined in the X.509 standard
 - **Valid from** – the date the certificate is valid from
 - **Valid until** – the date after which the certificate expires
6. Select **File > Save**.

Setting Up Dynamic Authentication

Use dynamic authentication to enable the user to set the name and password on the client.

You can create the Credential Request starting point with a Credential screen automatically when you initially create a new Hybrid App, or you can create the Credential Request starting point and associated screen manually. This procedure shows how to create the Credential Request starting point automatically when you create a new Hybrid App.

1. In the Mobile Development perspective, select **File > New > Hybrid App Designer**.
2. Follow the instructions in the Hybrid App Designer wizard:
 - **Enter or select the parent folder** – select the Hybrid App project in which to create the Hybrid App screen flow.
 - **File name** – enter a name for the Hybrid App screen flow. The extension for Hybrid App screen flows is `.xbw`.
 - **Advanced** – link the Hybrid App screen flow to an existing file in the file system.
 - **Link to file in the file system** – click **Browse** to locate the file to which to link the Hybrid App screen flow. Linked resources are files or folders that are stored in the file system outside of the project's location. If you link a resource to an editor, when you select the editor, the resource is selected in the WorkSpace Navigator. Conversely, when you select the resource in the WorkSpace Navigator, the editor is selected.

Click **Variables** to define a new path variable. Path variables specify locations on the file system.

3. In the Starting Points page, select **Credentials (authentication) may be requested dynamically from the client application**.
4. Follow the steps to create the type of Hybrid App you want. Click **Finish**.
5. In the Hybrid App Designer, open the **Flow Design** to see the Credential Request starting point and its associated Credential Request screen.

To see the two pre-defined keys, `cc_username` and `cc_password` in the Properties view, click the Credential Request starting point.

6. Double-click the **Credential Request** screen to open the Screen Design page.

The two editbox controls on the screen are bound to the pre-defined keys, `cc_username` and `cc_password`.

7. Select **Username**. In the Properties view, open the **Advanced** page.

On the Username editbox, **Credential cache username** is selected by default. Click the **Password** editbox; the associated **Credential cache password** checkbox is selected.

Note: If you create a Credential Request starting point and screen manually, you must add the editbox controls, create the keys for the username and password, and check the corresponding Credential cache username or password box.

8. (Optional) To use certificate-based authentication instead of the user name and password:

- a) Add a **MenuItem** to the Menu box.
- b) Select the MenuItem to see its Properties.
- c) In the Properties view, from Type, choose **Select Certificate**.

When the user selects the menu item on the device, a dialog opens to select a certificate for credentials.

9. Select **File > Save**.

The first time the Hybrid App is started following deployment, the credential screen opens. The username and password values are cached in the credential cache.

Note: If an e-mail-triggered screen flow has dynamic cached credentials, the cached credentials are not cached between invocations of the screen flow through an e-mail trigger.

Basic Authentication

On iOS, Android, and BlackBerry platforms, each Hybrid Web Container has a default basic authentication screen to enter credentials if challenged for basic authentication when Hybrid Web Container connects with the server.

The entered credentials are persisted, so any time the application restarts, the previously accepted credentials are used.

If the basic authentication screen is canceled, it is shown again only under these circumstances:

- New connection information is entered and saved on the settings screen
- The restart engine menu item is pressed on the settings screen
- The application is restarted (device restart or force stop)

See *HTTP Authentication Security Provider* in *Security* for more information.

Single Sign-on

Android, BlackBerry, and iOS Hybrid Apps can provide a single sign-on (SSO) token.

Cookie-based Network Edge Authentication

Unlike standard credential cache authentication, network edge authentication is global to the Hybrid Web Container, not specific to each Hybrid App. Each Hybrid Web Container has a dialog to prompt for HTTP basic authentication credentials when challenged, and a session header or cookie is returned if the system is so configured for SSO. See *HTTP Authentication Security Provider* in *Security* for more information.

The sequence of authentication is as follows:

1. Client Network Edge authentication – The client begins a session by sending an HTTP(S) request to the Reverse Proxy. The Reverse Proxy detects the un-authenticated request and challenges for Basic authentication. After the 401 challenge, the client may already have network credentials configured, or perhaps there is a callback to prompt for credentials.

2. The client sends another HTTP request with the credentials, which the Reverse Proxy validates, and if valid issues a Cookie with an SSO token value. The HTTP headers will be added to the request that is created and sent to SAP Mobile Platform.
3. SAP Mobile Platform receives the request and uses an enhanced CSI LoginModule to authenticate. This login module is configured to extract HTTP Headers from the request (Cookie values are a subset).
4. SAP Mobile Platform processes the request and a response is sent back to the client. The client is still waiting on the original HTTP request from the Reverse Proxy. When the response comes back, the Reverse Proxy typically adds the setCookie response header at this time to pass the SSO data back to the client to use in subsequent HTTP requests.
 - If the SSO token is valid, everything proceeds.
 - If the SSO token is invalid, a server to device method instructs the Hybrid Web Container to prompt for credentials again.

Configuring the Hybrid App to Use Credentials

Configure a Hybrid App to pass user credentials, which are authenticated by SAP Mobile Server and the EIS.

For information about configuring and implementing X.509 and SSO2 on the server, see *Security*.

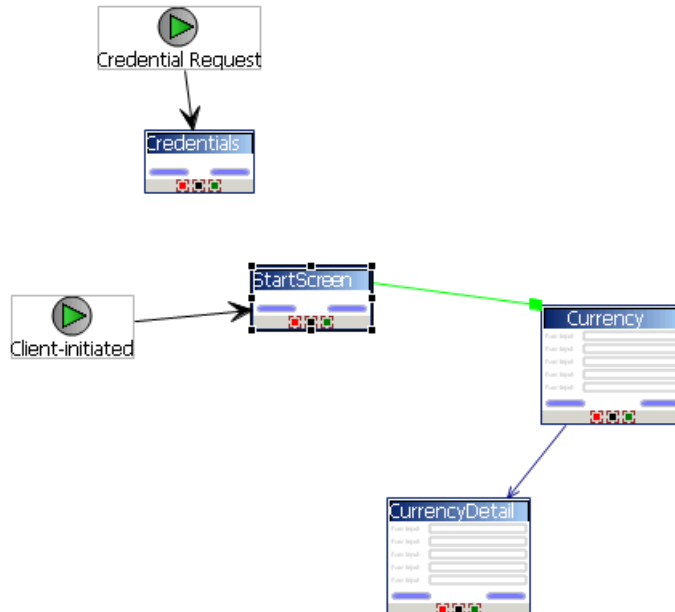
Configuring the Hybrid App to Use X.509 Credentials

Add a screen that contains a Specify Certificate Credentials menu item to the Credential Request starting point from which a Hybrid App user selects a certificate to gain access to the MBO and related resources.

1. In the Hybrid App Designer, add a **Credential Request** starting point to the Hybrid App.
2. Add a screen named **Credentials** and connect it to the Credential Request starting point.
3. Double-click **Credentials** to open it in the Screen Design. Add a **Select Certificate** menu item of the Submit type.

On the device, the Specify Certificate Credentials action prompts the user for a *.p12 certificate and passes it to SAP Mobile Server for validation.

4. Add a **Client-initiated** starting point to which you add screens that contain the Submit menu items used to run MBO operations and object queries, return and display results, and so on. These actions all use the same credentials created in the previous steps.



Configuring the Hybrid App to Use Static X.509 Credentials

When using static credentials, the Hybrid App does not prompt the user for credentials, instead it passes the credentials to SAP Mobile Server automatically and displays the Hybrid App's start screen.

1. Remove the Credential Request starting point and screen from the Hybrid App (so the client is no longer prompted for credentials).
2. From Flow Design, select **Authentication**, **Use static credentials**, and **Use certificate-based credentials**.
3. Click **Generate from Certificate**.
4. Browse to the location of the *.p12 certificate file.
5. Enter the certificate's password, select the alias, and click **OK**.
6. Save and regenerate the Hybrid App, and reassign it to a device.

Propagating a Client's Credentials to the Back-end Data Source

Use client credentials (including certificates and SSO tokens on EIS types that support them) to establish enterprise information system (EIS) connections on the client's behalf for all data source types.

To use client credentials, map an EIS connection's user name and password properties to system-defined "user name" and "password" personalization keys respectively. This creates a new connection for each client and the connection is established for each request (no connection pooling.)

Develop a Hybrid App Using the Hybrid App Designer

1. During development of the mobile business object MBO/operation, from the data source definition page (available either in the Creation wizard or from the Properties view), in the **Runtime Data Source Credential** section (or **HTTP Basic Authentication** section for a Web Service, RESTful Web Service, or SOAP MBO), enter the client credentials in the User name and Password fields. The runtime data source credential values (user name and password) that SAP Mobile WorkSpace uses for refresh or preview operations is taken in this order:
 - a) Any literal value entered in the User name and Password fields.
 - b) User-defined personalization keys that have non-empty default values.
 - c) System personalization keys 'user name' and 'password'.
 - d) User name and password property values contained in the connection profile.
2. During deployment of the package that contains such MBOs, map the design-time connection profiles to the existing or new server connections, but be aware that the user name and password portions for the selected server connection is replaced by the user name and password propagated from the device application.

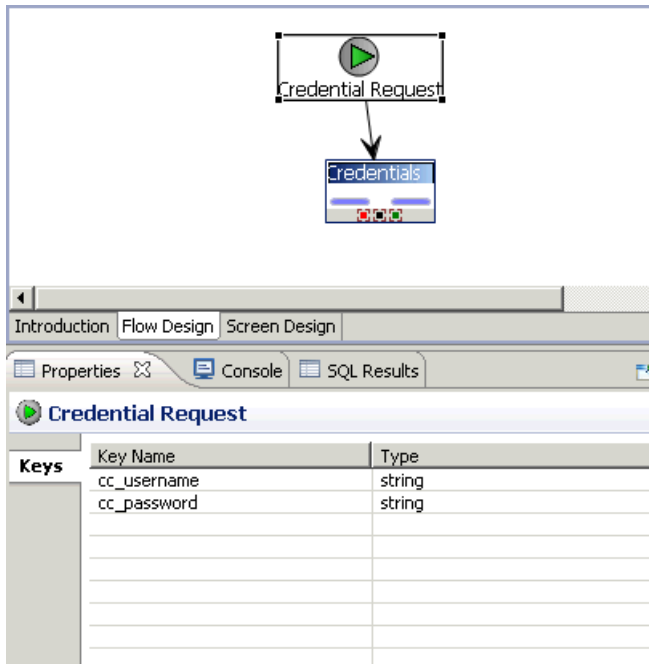
Note:

- Do not set client credentials using the Runtime Data Source Credential option for MBO's that belong to a cache group that uses a Scheduled policy, since this is unsupported.
 - In general, a MBO operation that uses data source credential settings as connection properties cannot have these settings mapped to an enterprise information system (EIS) during deployment. Instead, they maintain their original settings, which you can map after deployment using SAP Control Center.
 - When you create a new security configuration that includes the SAPSSOTokenLoginModule, and deploy it to a new domain, if the Hybrid App uses the MBOs associated with the new security configuration, you must specify an SAP Mobile Server domain that corresponds to the domain using the security configuration. See *Security* for more information about security configurations
-

Configuring a Hybrid App to Use SSO2 Tokens

Configure a Credential Request starting point from which a Hybrid App user can pass a user name and password to gain access to the MBO and related resources.

1. In the Hybrid App Designer, add a **Credential Request** starting point to the Hybrid App.
2. Add two keys to the Credential Request named `cc_username` and `cc_password`.
3. Add a screen named `Credentials` and connect it to the Credential Request starting point.

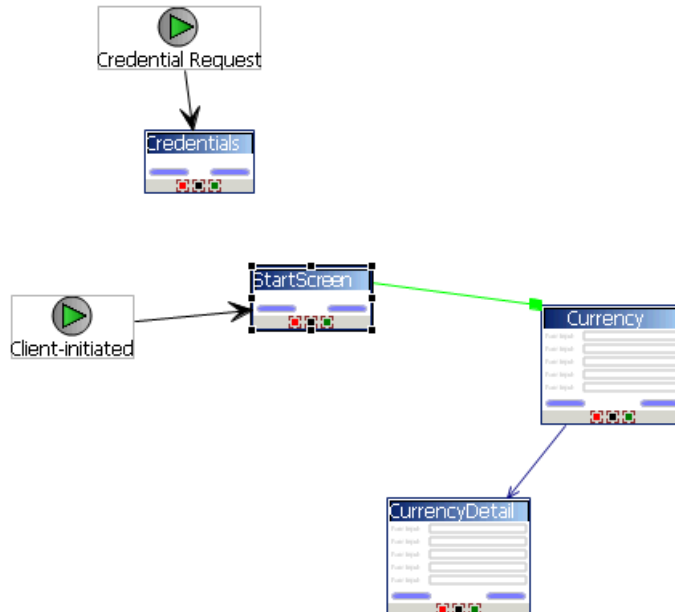


4. Double-click **Credentials** to open it in the Screen Design.
5. Add a **Save screen** menu item to the Menu, and two edit boxes (Username and Password).

The Save screen saves the Username and Password entered by the Hybrid App. You could also add a **Submit** menu item instead of **Save screen**.

6. Add a Client-initiated starting point to which you add screens that contain the Submit menu items used to run MBO operations and object queries, return and display results, and so on. These actions all use the same credentials created in the previous steps.

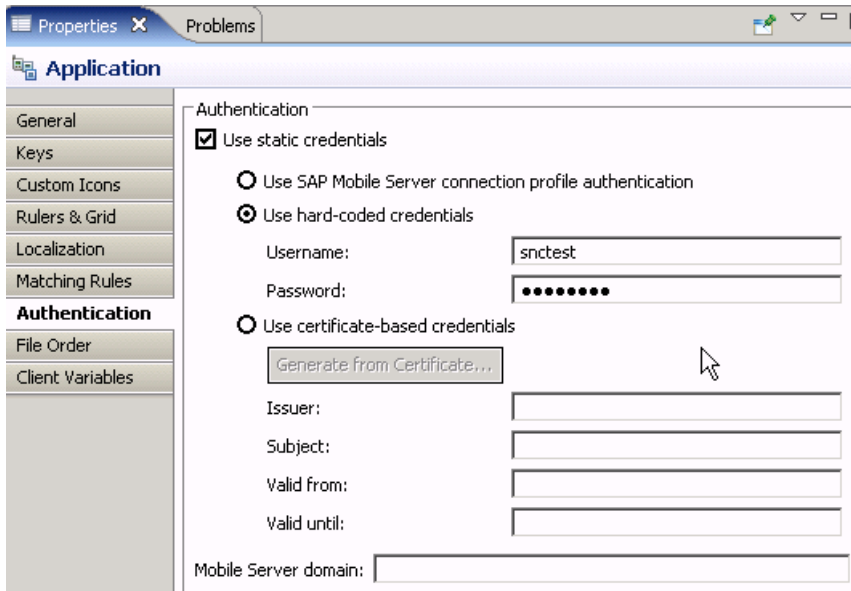
Develop a Hybrid App Using the Hybrid App Designer



Configuring the Hybrid App to Use a Static SSO2 Token

When using static credentials, the Hybrid App does not prompt the user for credentials, instead it passes the credentials to SAP Mobile Server automatically and displays the Hybrid App's start screen.

1. Remove the Credential Request starting point and screen from the Hybrid App (so the client is no longer prompted for credentials).
2. From Flow Design, select **Authentication, Use static credentials, and Use hard-coded credentials**. Enter a `username` and `password` that corresponds to those defined in SAP Control Center for the server connection (for example: `snctest/snctest`).



3. Save and regenerate the Hybrid App package, and reassign it to a device.

Modify Certificate Information for Hybrid App Packages

If using static credentials, either SSO token or static x.509 certification, you can replace the Hybrid App package certificate using either SAP Control Center or the `SUPMobileHybridApp.replaceMobileHybridAppCertificate()` API. To replace a certificate, you must have access to the certificate file and password.

Replacing the Hybrid App Certificate Through SAP Control Center

If using static credentials, you can set or modify the context variable certificate settings for a Hybrid App package from SAP Control Center.

The Hybrid App certificate password context variable is read-only. You can modify this only by using the Admin Java API method

`SUPMobileHybridApp.replaceMobileHybridAppCertificate()`.

1. From SAP Control Center, navigate to **Hybrid Apps > <Hybrid_App_Name>**, where *Hybrid_App_Name* is the name of the Hybrid App package.
2. On the Context Variables tab, verify that `SupUser` and `SupPassword` contain valid credentials for the specified security configuration, for Hybrid App packages that do not use certificate-based authentication.
3. For Hybrid App packages that use certificate based authentication, you can view these context variables:
 - `SupCertificateIssuer`

Develop a Hybrid App Using the Hybrid App Designer

- SupCertificateSubject
- SupCertificateNotAfter
- SupCertificateNotBefore

Replacing the Hybrid App Certificate Using the Admin API

Use the `SUPMobileHybridApp.replaceMobileHybridAppCertificate()` method to set or modify the certificate password context variable for the Hybrid App package.

```
InputStream is = getClass().getResourceAsStream("sybase101.pl2");
ByteArrayOutputStream baos = new ByteArrayOutputStream();
byte[] buf = new byte[512];
int count;
while ((count = is.read(buf)) != -1) {
    baos.write(buf, 0, count);
}
is.close();
baos.flush();
baos.close();
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setWID(4);
hybridAppID.setVersion(1);

mobileHybridApp.replaceMobileHybridAppCertificate(hybridAppID,
    baos.toByteArray(), "password");
```

Content Security on Devices

This explains how the files that make up the Hybrid Web Container are protected when stored on the device, and under what circumstances the files are stored in plain text.

Content Security on Android Devices

On Android operating systems, all Hybrid Web Container files, and extra data entered by the user or retrieved from the server, are encrypted before being stored in the application's sandbox and SQLite database. You can turn off the encryption of Hybrid Web Container files to decrease the load times for Hybrid Apps by using the `disableFileEncryption` customization point.

The cryptographic libraries provided by Google/Android are used. Specifically, the encryption algorithm used is AES-256 symmetric encryption.

Hybrid Web Container Files

Hybrid Web Container files include all the files contained in the `<Hybrid_App_package_name>.zip` that is deployed to the device, including all HTML, JavaScript, CSS, and any other files that may be included as part of the ZIP package.

- When the platform's browser control requests these Web files, they are read from the device's sandbox, stored unencrypted on the file system temporarily, and then passed to the browser control through a content provider.

- These temporary files are removed from the content provider immediately after the last of them are requested by the browser control.

Note: Prepackaged files are not secured on Android. They are stored in the `assets` directory unencrypted.

Attachments

If attachments, such as *.docx, *.pdf, and so on, are part of the `<Hybrid_App_package_name>.zip` deployed to the device, they are stored in the application's sandbox after they have been encrypted through the Google/Android crypto libraries.

- When the JavaScript requests these attachments for viewing, they are read from the application's sandbox, and temporarily written unencrypted to the device's flash memory for the external viewers to display them.
- Once the application closes, these temporary attachment files are immediately removed.

Note: The Android operating system enforces the sandboxing of these temporary files.

Attachments that are downloaded through an online request using an object query are stored unencrypted in the device's flash memory for the file viewers to display them. Once the application closes, these temporary attachment files are immediately removed.

Images

The image is saved, unencrypted on the file system, into the Gallery application, (`ImageOptions.CAMERA`, `ImageOptions.BOTH`).

Note: The Android operating system enforces the sandboxing of these image files.

Cached Online Requests

The results of online requests that are specified to be cached are stored on the device's SQLite database (after they are encrypted through the Google/Android cryptographic libraries). Cached results are removed when the Hybrid Web Container is unassigned from the device, or uninstalled from the server.

Notifications From the Server

Notifications from the server are stored in the same SQLite database after they have been encrypted through the Google/Android cryptographic libraries, including the payload that makes up the notification. When the notification is acted upon, the JavaScript makes a request for the notification contents. This is read from the SQLite database, unencrypted, and passed to the browser in memory.

User Input Sent to the Server

When the device has no network connectivity, and the user submits a Hybrid App for the server to process, the data destined for the server is queued up on the device. The contents of this queue are again encrypted through the Google/Android cryptographic libraries before it is stored into the SQLite database.

Encryption Keys

- How the encryption key is generated:
 - A generated GUID is used as the key for encrypting the data ("data password")
 - A user-provided password (PIN) is used to secure/encrypt the "data password," which is persisted in its encrypted form. In order to have access to the "data password", one must know the user password.
 - The salt is a different persisted, generated GUID.
 - Encryption of data is done with the "data password."
- Where is the encryption key stored?
 - The "data password" is persisted in its encrypted form in a separate table in the SQLite database.

Content Security on BlackBerry Devices

In general, all Hybrid Web Container files and extra data entered by the user, or retrieved from the server, are stored on the BlackBerry device's PersistentStore.

This is the same storage area used by e-mail, calendar entries, and applications. See your BlackBerry documentation for information about persistent store APIs.

The BlackBerry Hybrid Web Container uses the RIM PersistentContent APIs when reading and writing of data from PersistentStore is required. This ensures that the content being written is stored at the device's current encryption level. See your BlackBerry documentation for information about content protection strength settings.

When content protection is turned on, content on the BlackBerry device is protected using the 256-bit Advanced Encryption Standard (AES) encryption algorithm.

- Use 256-bit AES encryption to encrypt stored data when the BlackBerry device is locked
- Use an Elliptic Curve Cryptography (ECC) public key to encrypt data that the BlackBerry device receives when it is locked

These settings apply to the encryption of data that the BlackBerry device receives while locked:

Content protection strength setting	ECC encryption key length (in bits)
Strong	160
Stronger	283
Strongest	571

The BlackBerry Hybrid Web Container also registers a PersistentContentListener, which allows it to be notified when the device's encryption level changes. This also enables previously stored content to be re-encoded to the new encryption level setting. The device's

encryption level setting can be changed by a BlackBerry Enterprise Server Administrator remotely, or by the user, from the device.

Hybrid Web Container Files

Hybrid Web Container files include all the files contained in the `<hybrid_app_name>.zip` that is deployed to the device, including all HTML, JavaScript, CSS, and any other files that may be included as part of the Hybrid App ZIP package. When the platform's browser control requests these Web files, they are read from the device's PersistentStore and passed to the browser control in memory, which means there are no temp files.

Attachments

If attachments, such as *.docx, *.pdf, and so on, are part of the `<hybrid_app_name>.zip` deployed to the device, they are stored on the device's PersistentStore:

- When the JavaScript requests to display these attachments, they are read from the PersistentStore, and temporarily written unencrypted to the device's flash memory for the external viewers to display them.
- Once the mobile Hybrid App closes, these temporary attachment files are immediately removed.

Attachments that are downloaded using an online request that use an object query are stored unencrypted in the device's flash memory for the file viewers to display them. Once the Hybrid App closes, these temporary attachment files are immediately removed.

Images

Images are stored unencrypted on the file system and saved into the `Pictures` folder (`ImageOptions.BOTH`).

Cached Online Requests

The results of online requests that are specified to be cached are stored on the device's PersistentStore. Cached results are removed when the Hybrid Web Container is unassigned from the device, or uninstalled from the server.

Notifications From the Server

Notifications from the server are stored in the same PersistentStore area, including the payload that makes up the notification. When the notification is acted upon, the JavaScript makes a request for the notification contents. This is read from the PersistentStore and passed to the browser in memory.

User Input Sent to the Server

When the device has no network connectivity, and the user submits a Hybrid App for the server to process, the data destined for the server is queued up on the device. This queue is part of the device's PersistentStore.

Content Security on iOS Devices

On iOS devices, all Hybrid Web Container files and extra data entered by the user or retrieved from the server, are stored in a SQLite database that uses the SQLite Encryption Extensions (AES-128).

Hybrid Web Container Files

Hybrid Web Container files include all the files contained in the <Hybrid_App_package_name>.zip that is deployed to the device, including all HTML, JavaScript, CSS, and any other files that may be included as part of the ZIP package. When the iOS device's browser control requests these Web files, they are read from the encrypted SQLite database. The data is temporarily written to the file system under the application sandbox, after which the browser control reads the file contents into memory. The temp files are removed when the Hybrid App closes.

Note: When using a prepackaged Hybrid App, all of the files associated with the prepackaged Hybrid App (HTML, JavaScript, CSS, and so on) exist within the sandbox in clear text.

Attachments

If attachments, such as *.docx, *.pdf, and so on, are part of the <Hybrid_App_package_name>.zip deployed to the device, they are stored in the encrypted SQLite database.

- When the JavaScript requests the attachments for viewing, they are read from the database, and temporarily written, unencrypted, to the Hybrid Web Container's sandbox for the viewer to display them.
- Once the application closes, these temporary attachment files are immediately removed.

Attachments that are downloaded using an online request that uses an object query are stored unencrypted in the Hybrid Web Container's sandbox for the file viewers to display them. Once the application closes, these temporary attachment files are removed immediately.

Images

Images are stored unencrypted in the Hybrid Web Container's sandbox, then removed when the application closes.

Cached Online Requests

The results of online requests that are specified to be cached are stored in the encrypted SQLite Database. Cached results are removed when the Hybrid Web Container is unassigned from the device, or uninstalled from the server.

Notifications From the Server

Notifications from the server are stored in the same encrypted SQLite database, including the payload that makes up the notification. When the notification is acted upon, the JavaScript makes a request for the notification contents. This is read from the SQLite database and passed to the browser in memory.

User Input Sent to the Server

When the device has no network connectivity, and the user submits an application for the server to process, the data destined for the server is queued up on the device. This queue is again part of the encrypted SQLite database.

Encryption Keys

- The Hybrid Web Container generates a hash from the password entered by the user, and a salt, combined
- The Hybrid Web Container generates a random key
- The Hybrid Web Container encrypts the key with the hash and stores it in the app area of the keychain

Content Security on Windows Mobile Devices

On Windows Mobile Professional, Hybrid Web Container files are stored unencrypted on the device's file system, and Hybrid Web Container settings are stored unencrypted in the device's registry.

Note: The Windows Mobile Hybrid Web Container defers all security and encryption responsibilities to the Afaria® Security Manager; therefore, SAP strongly recommends that you use Afaria Security Manager.

If you do not use Afaria Security Manager, you must:

- Protect these files through alternative means. The \Program Files\SAP\Messaging\AMP folder (and all of its sub folders) must be secured on the device.
- To protect the Hybrid Web Container settings, the [HKEY_LOCAL_MACHINE\Software\SAP\MessagingClientLib] registry key (and all of its sub keys) must be secured on the device.

Hybrid Web Container Files

Hybrid Web Container files include all the files contained in the <hybrid_app_name>.zip that is deployed to the device, including all HTML, JavaScript, CSS, and any other files that may be included as part of the Hybrid App zip package. These are all stored unencrypted on the file system of the device.

Attachments

If attachments, such as *.docx, *.pdf, and so on, are part of the <hybrid_app_name>.zip deployed to the device, they are stored unencrypted on the file system of the device.

- When the JavaScript requests these attachments for viewing, a file URI is constructed for a suitable external viewer to display these files.
- Once the Hybrid App closes, these temporary attachment files are immediately removed.

Images

Images are stored unencrypted on the file system, then removed when the Hybrid App closes.

Cached Online Requests

The results of online requests that are specified to be cached are stored unencrypted on the device's file system. Cached results are removed when the Hybrid Web Container is unassigned from the device, or uninstalled from the server.

Notifications From the Server

Server notifications are stored unencrypted in the Inbox database of the device (the same database that houses the device's regular e-mail messages). When the notification is acted upon, the JavaScript makes a request for the notification contents. This is read from the Inbox database and passed to the browser in memory. If you are not using Afaria Security Manager, the Windows Mobile Inbox database must be secured.

User Input Sent to the Server

When the device has no network connectivity, and the user submits a Hybrid App for the server to process, the data destined for the server is queued up on the device. The contents of this queue are stored in an unencrypted SQLite database.

Localization and Internationalization

You can localize different objects in the Hybrid App Designer, such as the names of screen controls, screens, and mobile business objects.

You can localize the Hybrid App by creating locale properties files. You can then load, update, and generate localized Hybrid Apps.

All the localizable strings in the Hybrid App Designer XML model work as resource keys in the localization properties file. All the localization properties files are in the same directory as the Hybrid App packages (.xbw files).

Resource keys are divided into these categories, which include all the elements of the Hybrid App Designer XML model:

- Menus
- Controls
- Screens

Localization consists of two levels of localization—the Hybrid App Designer XML model localization and the Hybrid App client localization.

All locale properties files are saved in the same directory as the Hybrid App package.

To ensure that the correct locale is picked up for the Hybrid Web Container, the following mechanism is used:

1. If a precise match is found for language and country, for example, English - United States (en-us) is the locale and the file exists in `html\en-us\hybridapp*.html`, that file is used and the HTTP lang parameter is set to "en-us."
2. If a precise match for country is not found, the language is used. For example, English (en). If the file exists in `html\en\hybridapp*.html`, that file is used and the HTTP lang parameter is set to "en."
3. If a language match is not found, the default locale is used. If the file exists in `html\default\hybridapp*.html`, that file is used and the HTTP lang parameter is set to "default";
4. If a default match is not found, no locale is used. If the file exists in `html\hybridapp*.html`, that file is used and the HTTP lang parameter is set to "".

Localization Limitations

Locale properties files have some restrictions.

These restrictions apply:

- Traditional Chinese characters are not supported on iOS.
- Hybrid Apps that have names that begin with numbers or special characters cannot be localized; you will receive an error when you generate the code. Make sure that any Hybrid App you want to localize does not have a file name that begins with a number or special character.
- When you specify a country for the language, the basic language locale must also be available. For example, if you create a locale and specify English as the language and the United States as the country, then a locale for English (the basic language) must also be available.
- If you create a locale that specifies language, country, and variant, the locale for the basic language and the locale for the basic language and the country must be available. For example, if you create a locale and specify English as the language, United States as the country, and WIN as the variant, then English (United States) and English locales must also be available.
- The language code must be a 2-letter code, and the country code can be either a 2-letter or 3-letter code.

Note: BlackBerry 9800 Asia simulators do not have a place to specify a country name, so you can specify only a language.

- If you specify a variant, the country code must be a 2-letter code.

Localizing a Hybrid App Package

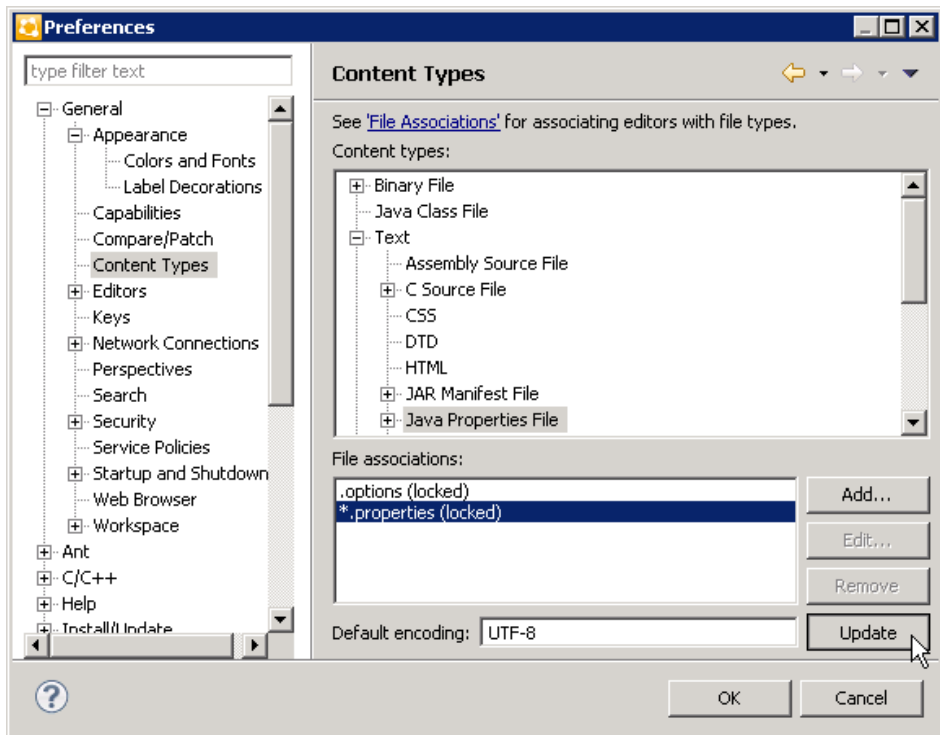
Use the Hybrid App Designer to complete these tasks to localize Hybrid App packages (.xbw files).

Changing the Encoding Type

Change the encoding type in Preferences.

If you manually localize the locale properties file using an external editor, you must make sure the file is encoded in ASCII, so that the content can be correctly read and converted to Unicode. The localization file is encoded in standard ISO-8859-1. All non-ASCII character values are converted to escaped Unicode hexadecimal values before they are written to the properties files. Before translating the localization file, select the correct file encoding option, for example UTF-8.

1. In SAP Mobile Platform, select **Window > Preferences**.
2. Expand **General > Content Types**.
3. In the right pane, select, **Text > Java Properties File**.
4. In the **File Associations** list, select ***.properties**.
5. In the Default encoding field, change ISO-8859-1 to **UTF-8**, and click **Update**.



Creating and Validating a New Locale Properties File

Create a locale properties file as the default locale.

Prerequisites

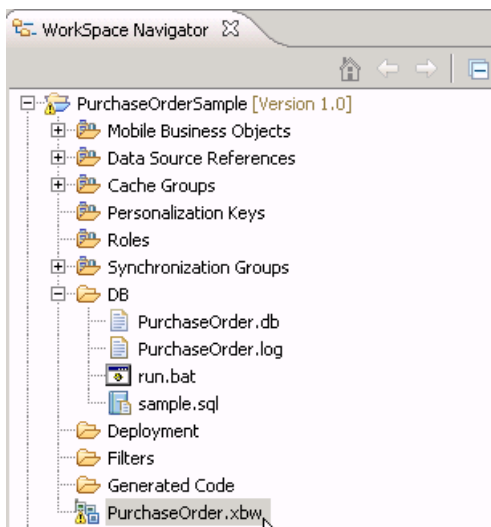
You must have an existing Hybrid App package before you create the locale properties file.

Task

When you create a new locale, keep in mind:

- When you specify a country for the language, the basic language locale must also be available. For example, if you create a locale and specify English as the language, then there must also be a locale for English (the basic language).
- If you create a locale that specifies language, country, and variant, the locale for the basic language and the locale for the basic language and the country must be available. For example, if you create a locale and specify English as the language, United States as the country, and WIN as the variant, then English (United States) and English locales must also be available.

1. In WorkSpace Navigator, double-click the *Hybrid App.xbw* file to open the Hybrid App Designer.

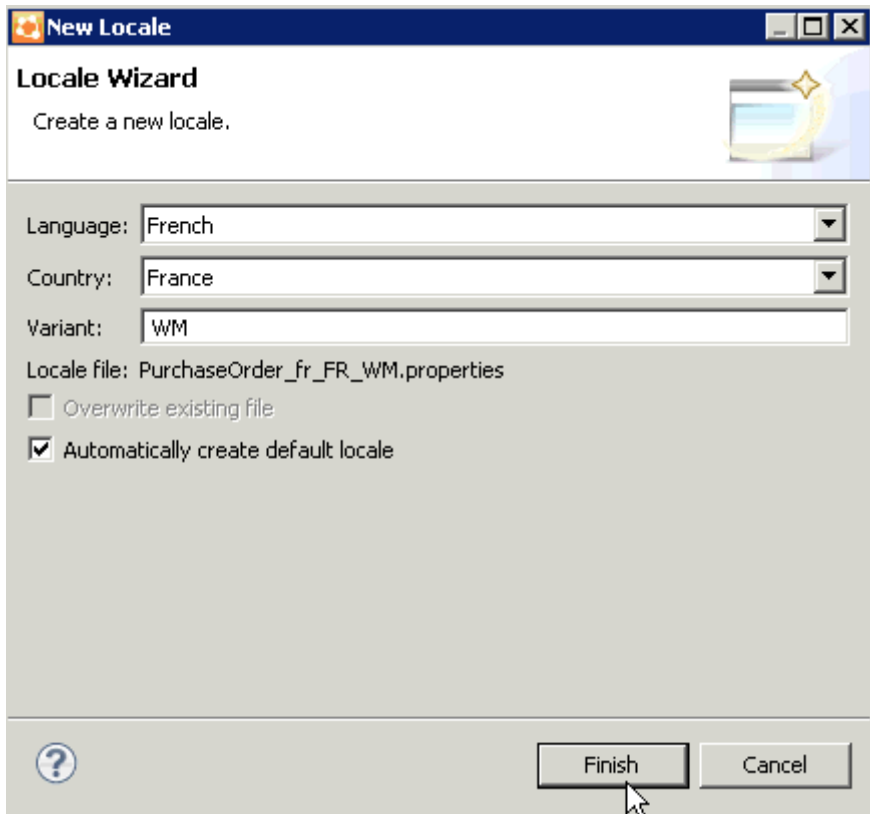


2. Click the **Flow Design** tab.
3. Right-click in a blank area on the Flow Design page, and select **Show Properties View**.
4. In the Properties view, on the left, click the **Localization** tab.
5. In the right pane, click **New**.
6. Select or enter the information for the new locale, select **Automatically create default locale**, and click **Finish**.

Option	Description
Language	Select the language.
Country	Select the country.
Variant	Enter the variant, which is the vendor or browser-specific code. For example, enter WIN for Windows, MAC for Macintosh and POSIX for POSIX. If there are two variants, separate them with an underscore, and put the most important one first. For example, a Traditional Spanish collation might construct a locale with parameters for language, country, and variant as: es, ES, Traditional_WIN.
Overwrite existing file	Overwrite an existing localization file.
Automatically create default locale	Automatically create the default locale properties file. For example, if you specify the language as English and the country as United States for a device application called test, then both test_en_us.properties and test.properties files are created.

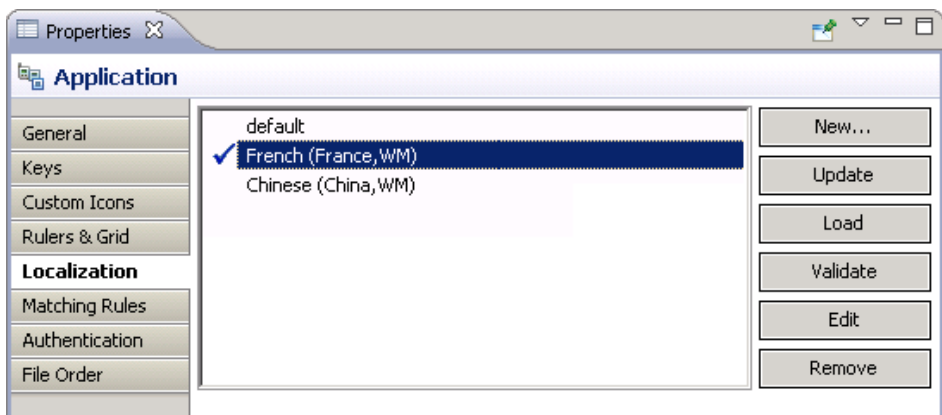
For example:

- Language – select **French**.
- Country – select **France**.
- Variant – enter a value to make this locale file unique from others, for example, WM for Windows Mobile.



This locale file is now the default locale file, and will be used when the regional setting of the device does not match that of any supplied locale file.

7. In the Properties view, in the Localization page, select the file to validate and click **Validate**.



Develop a Hybrid App Using the Hybrid App Designer

The properties file is scanned and if there are any errors, a dialog appears. Click **Yes** to correct the errors automatically; click **No** to see the errors in the Problems view.

Editing the Locale Properties File

Edit the locale properties file.

1. In WorkSpace Navigator, under the Generated Code folder, right-click the locale properties file you created, and select **Open With > Properties File Editor**.
2. You can make and save changes to the file in the Properties File editor, for example, you can replace all the values of the resource keys with Chinese characters.
3. Select **File > Save**.

The next time you open the locale properties file, notice that all of the ASCII characters have been changed.

4. In the Localization pane, select the localization file you edited, and click **Load**.

The elements of the application in the editor are translated into the language you specified if the localization file passes the loading validation.

Removing a Locale

Remove locale properties files.

1. In the Screen Design page Properties view, click **Localization**.
2. Select the locale to remove and click **Remove**.
3. Click **Yes** to confirm the deletion.

Updating the Current Locale

Update the currently loaded locale properties file with the resource keys from the current Hybrid App Designer.

If the locale properties file does not already exist, it is created. If the current locale is not defined in the Hybrid App file, the updated locale is used as the default, and the file name is *{device_application}.properties*. Otherwise, the locale defined in the Hybrid App file is updated.

Note: When you update the localization bundle, it removes all resources that are not explicitly bound to existing UI elements (screens, menuitems, controls, and so on). If you want to manually supply resources, you must do so after updating, and be careful not to update the resource bundles afterwards, or you will have to re-add those manually-supplied resources after updating.

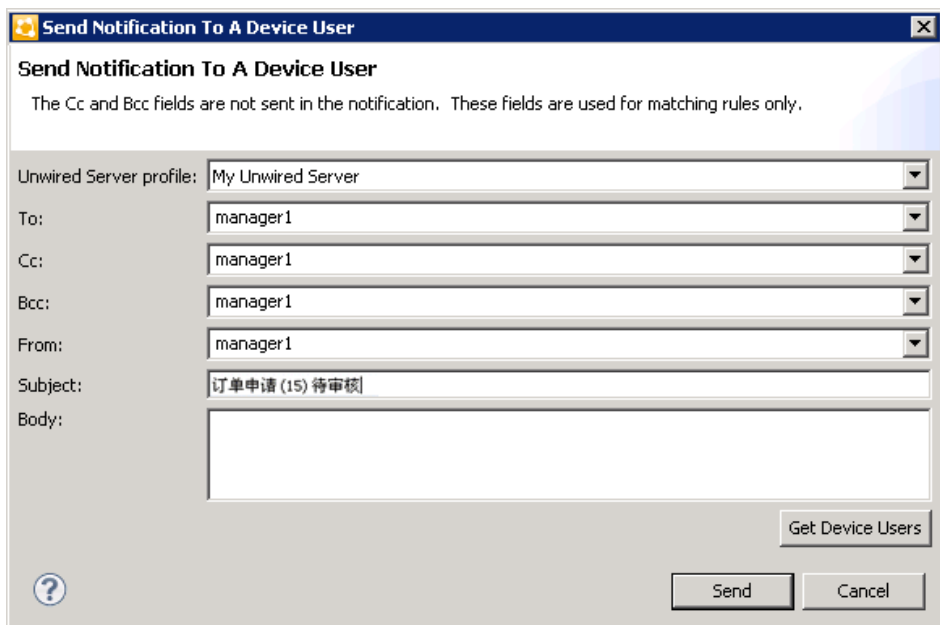
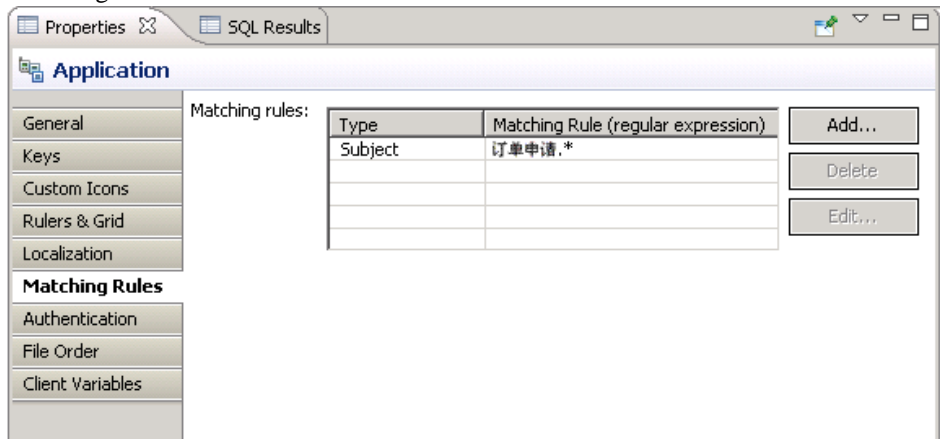
1. In the Screen Design page Properties view, click **Localization**.
2. Click **Update**.

Hybrid App Package Internationalization

The internationalization feature depends on the internationalization setting on the operating system where SAP Mobile Platform Hybrid App is running.

In the Hybrid App Designer, you can use international data in:

- Matching rules for notifications.



- Key names – you can create keys with names in other languages and map them to mobile business object parameters.

Key

Specify the name of the key and, optionally, the input data binding for the key as well.

Name:

Type:

☐ Sent by server

Input Data Binding

Mobile business object:

☐ Mobile business object attribute

Name:

☒ Convert to UTC

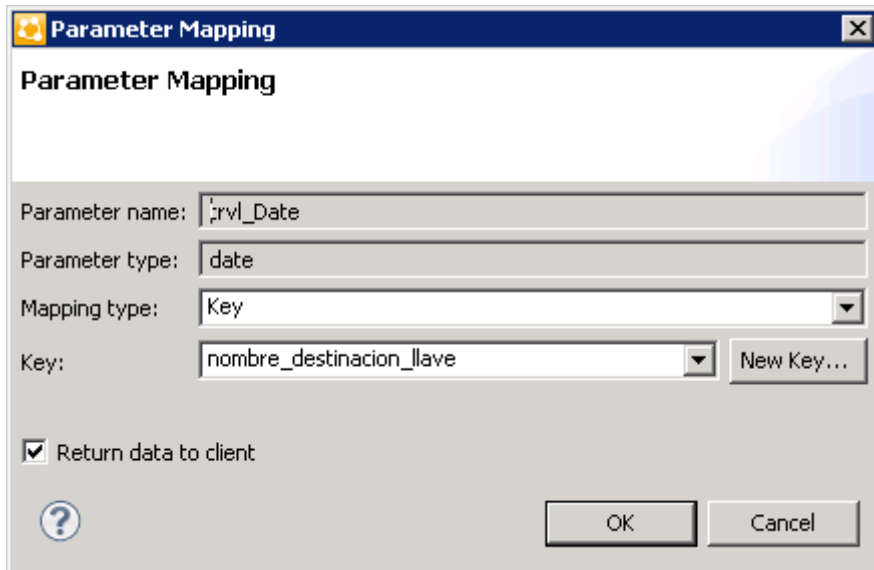
☐ Mobile business object relationship

☐ MBO object query results

☐ Hard-coded value

☒ User-defined

☐ Extraction rule



- Generated Code folder – you can include languages other than English in the code generation path based on the name of the selected language.

Internationalization on the Device

On the device, e-mail messages and data can include languages other than English.

The internationalization feature depends on the internationalization setting on the device where the Hybrid App client running.

E-mail messages can be sent and received using Chinese, for example, which can then be used to extract the parameter. You can also create and update records in using international data, such as Chinese. For example:

Develop a Hybrid App Using the Hybrid App Designer



Test Hybrid App Packages

Test a Hybrid App on a device or simulator.

1. Launch and/or connect to the mobile device or emulator.
2. Deploy the Hybrid App package to the device.
3. Establish the connection to the server on the device.
4. For user-initiated Hybrid App packages, go to the Hybrid Apps menu and click on the appropriate Hybrid App.

5. For e-mail subscription Hybrid App packages, send the e-mail to the device, either automatically, for example, database trigger, or manually, through the Send E-mail dialog; then open that e-mail on the device.
6. Enter data and execute menu items appropriately.
7. Verify that the backend is updated correctly.
8. Check the logs.

Testing Server-Initiated Hybrid App Packages

Test a server-initiated Hybrid App package.

1. In the Hybrid App Designer, open the Hybrid App <hybridapp>.xbw.
2. Click **Flow Design**.
3. Right-click in the editor, and select **Send a notification**.
4. In the Send a Notification window:
 - a) Select the SAP Mobile Server profile and click **Get Device Users**.
 - b) Choose the desired user and fill in the fields according to the matching rules specified when creating the Hybrid App.
5. Click **Send**.
6. On the client, from the applications screen, open the Hybrid Web Container.
7. In the client application, click **Hybrid Apps**. This contains the server-initiated Hybrid App.

Viewing Hybrid App Messages on the Device

Where Hybrid App messages that are sent to the device appear varies by platform.

Note: Registration must be successfully completed either through providing an activation code or a password for automatic registration in the connection settings before any Hybrid App packages appear on the device.

Android and BlackBerry

To see Hybrid App messages on BlackBerry devices and simulators:

1. In the applications screen, open **Hybrid Web Container**.
2. Messages appear in the Messages screen.

iOS

To see Hybrid App messages on iOS devices and simulators:

1. Open the **Hybrid Web Container**.
2. Click **Messages** to view messages.

Windows Mobile

To see Hybrid App messages on Windows Mobile devices and emulators:

Develop a Hybrid App Using the Hybrid App Designer

1. In the Programs screen, open the **Hybrid Web Container**.
2. Messages appear in the Messages screen.

Launching a Server-initiated Hybrid App on the Device

Server-initiated Hybrid App messages are sent to the Hybrid Web Container that is installed on the device.

When you click the **Hybrid Apps** menu item in the Hybrid Web Container, only the latest version of the Hybrid Apps appear. When you click the icon for a particular Hybrid App, the Hybrid App version that is associated with the notification is launched, whether it is the latest version or not.

Example

You develop a Hybrid App that has both client-initiated and server-initiated starting points. You deploy the initial version, which is called version 1, and a notification is sent.

Next, make some changes and deploy a second version, version 2. Again, a notification is sent.

There are now three ways that this Hybrid App can be launched, and the way that it is launched determines which version of the Hybrid App is launched:

- If you launch the application from the **Hybrid Apps** menu item, the last deployed version of the Hybrid App, in this case, version 2, is launched. Although version 1 of the Hybrid App still exists somewhere on the device, it is never used as long as you launch the Hybrid App from the Hybrid Apps menu.
- If you launch the Hybrid App by opening the initial notification, the version that corresponds with the latest version that existed at the time the notification was sent, is used. In this case, that is version 1; it does not matter that a later version (version 2) exists.
- If you launch the Hybrid App by opening the second notification, the version that corresponds with the latest version that existed at the time the notification was sent is used. In this case, that is version 2.

Debugging Custom Code

Debug the Hybrid App package HTML and JavaScript files using a Windows desktop browser.

This procedure uses Google Chrome as an example, but you can use any browser that supports JavaScript debugging.

1. Change the tracing level of Hybrid App to Debug.
2. Open the browser to use for debugging and open the Java Console.

If you are using Chrome:

- a) Add this command line option to the shortcut used to start Chrome:

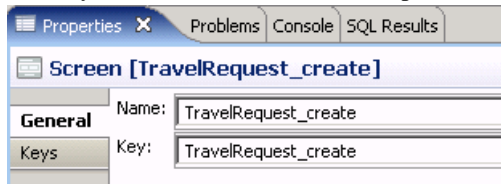
```
..\chrome.exe" --allow-file-access-from-files
```


3. You can debug a client-initiated Hybrid App up until the point where a menu item of the Submit type is performed. If the menu item action is an Online Request, place the XMLWidgetMessage (available in the WorkflowClient trace log located in SMP_HOME\Servers\UnwiredServer\logs\WorkflowClient) that is the expected response message into an rmi.xml file and place it at the same level as the generated hybridapp.html file.

Note: Control characters are not parsed correctly when using rmi.xml and Chrome to debug the Hybrid App. Do not format the content of the rmi.xml when debugging the Hybrid App using Chrome. If you want a formatted look at the rmi.xml file, make a copy of the file for that purpose.

4. From WorkSpace Navigator, drag and drop the hybridapp.html file for the Hybrid App to debug onto the browser window.
5. Find the name of the key to debug:
 - a) In Flow Design, click the screen to debug.
 - b) In the Properties view, click **General** in the left pane.

The key name is shown, in this example, that is TravelRequest_create.



6. In the URL, add the **?screenToShow=<Screen_name>** parameter to the end of the URL, for example:

```
file:///C:/Documents%20and%20Settings/<user_name>/
workspace/HybridApp101/Generated%20HybridApp/
travelrequest/html/hybridapp.html?
screenToShow=TravelRequest_create
```
7. To simulate an e-mail message triggered Hybrid App:
 - a) Create a file called transform.xml and place the contents of the XMLWidgetMessage into it.
 The contents of the XMLWidgetMessage are in the WorkflowClient trace log in <UnwiredPlatform_InstallDir>\UnwiredPlatform\Servers\UnwiredServer\logs\WorkflowClient).
 - b) To provide data to the Hybrid App you are debugging, place the transform.xml file at the same level as the generated hybridapp.html file (Generated Hybrid App\<Hybrid_App_name>\html).
 - c) Add a **?loadtransformdata=true** parameter to load the data into the Hybrid App.

Configuring Messaging Server Log Settings

Messaging Server logs create trace configurations for messaging modules, and retrieve trace data for all or specific messages. Configure trace configuration properties for modules to

specify the amount of detail that is written to the log. You can configure trace settings for the primary server cluster in SAP Control Center for each module. The settings are available to cluster servers through the shared data folder.

Note: The default settings may only need to change in case of technical support situations where, for diagnostic reasons, a request is made to configure the specific module(s) settings, and provide the request log. In all other cases, the administrator or developer should not need to change the settings.

Additionally, you should always use SAP Control Center to configure server logs. If you manually edit the configuration file, especially on secondary servers in a cluster, the servers may not restart correctly once shut down.

1. In the SAP Control Center left navigation pane, click **Configuration**.
2. In the right administration pane, click the **Log Setting** tab and select **Messaging Server**.
3. Select Default, or one or more of the messaging service modules. Click **Show All** to show all modules.

Module	Description
Default	Represents the default configuration. The default values are used if optional fields are left blank in a module trace configuration. Required.
Device Management	Miscellaneous functions related to device registration, event notification, and device administration. Enable tracing for problems in these areas.
JMSBridge	This module handles communications from the SAP Mobile Server to the messaging server. Enable tracing to view the detailed messaging exchange.
MO	This module handles the delivery of messages between the client and server, including synchronous function calls from client to server. Enable tracing for MO errors and message delivery issues.
SUPBridge	This module handles communications from the messaging server to the SAP Mobile Server. Enable tracing to view the detailed messaging exchange.

Module	Description
TM	This module handles the wire protocol, including encryption, compression, and authentication, between the messaging server and clients. All communication between the client and the messaging server passes through TM. Enable tracing for authentication issues, TM errors, and general connectivity issues.
WorkflowClient	The WorkflowClient module.

4. Click **Properties**.

- a) Enter trace configuration properties. If you selected multiple modules, a string of asterisks is used to indicate settings differ for the selected modules. You can select the option to view or change the property value for any module.

Property	Description
Module	Display only. Default, module name, or list of module names selected.
Description	(Optional) Custom description of the server module.
Level	Trace level for the module - DISABLED, ERROR, WARN, INFO, DEBUG, DEFAULT. If the default trace level is specified for the module, the module uses the trace level defined for Default. Required.
Max trace file size	(Optional) Maximum trace file size in MB. If the trace file size grows larger than the specified value, the trace file data is backed up automatically.
User name	(Optional) Only data for the specified user name is traced.
Application Connection ID	(Optional) Only data for the specified Application ID is traced.

- b) Click **OK**.

Log files for each module are stored in folders of the same name located in:
SMP_HOME\Servers\UnwiredServer\logs.

Manage a Hybrid App Package

The Hybrid Apps node in SAP Control Center allows administrators to view and manage deployed Hybrid App packages, including display name, module name, and module version.

Administrators deploy Hybrid App packages into the SAP Mobile Platform cluster through this node, as well as manage notification settings configuration.

Registering or Reregistering Application Connections

Registering an application connection groups the user, device, and application to create a unique connection in SAP Control Center, so the registered connection activity can be monitored. Use SAP Control Center to manually register an application connection. You can also reregister an application connection when the association between the user, device and application breaks or requires a different pairing.

For more information on registering and reregistering application connections, see *How Connections Are Registered in Mobile Application Life Cycle*.

1. In the left navigation pane, click the **Applications** node.
2. In the right administration pane, click the **Application Connections** tab.
3. Choose an action:
 - Click **Register** to register a new application connection. Using the Activation Code, this application is then paired with a user and a device.
 - Click **Reregister** to associate the application with a new device and user pairing. For example, reregister the application connection if someone loses their device. By reregistering the application connection, the user then receives the same applications and workflows as the previous device.

Note: If the client application does not support reregistration, you cannot reregister the application connection. To determine if the client application supports reregistration, review the **Capabilities** properties for the application connection. If the **Application Supports Client Callable Components** property has a value of `False`, reregistration is not supported.

4. In the Register Application Connection or the Reregister Application Connection dialog.
 - a) For new device registration only, type the name of the user that will activate and register the device. For reregistrations or clones, the same name is used and cannot be changed.
 - b) (Not applicable to reregistration.) Select the name of the template for initial application connection registration. The template you use supplies initial values in the subsequent fields.

- **Default** – a default template that you can use as is, or customize.
- **HWC** – a default template for Hybrid Web Container. Use as is, or customize. If you use the HWC template, Application ID must be set to HWC.
- **Custom** - customized templates are listed.

Note: You cannot change the application connection template for an application connection after registration.

5. Change the default field values for the template you have chosen.

If you are using Relay Server, ensure the correct values are used.

- **Application ID**- the application ID registered for the application. The value differs according to application client type - native application, Hybrid App, or Online Data Proxy client. See *Application ID Overview* for guidelines.

Note: If the template you have chosen supplies the Application ID, then this field is read-only.

- **Security Configuration**- select the security configuration relevant for the application connection.
- **Logical Role**- (not applicable to reregistration) select the logical role that users must belong to in order to access the application.
- **Domain**- select the domain to which the application connection is assigned. A domain is not required for registering application connections for Hybrid Web Container applications.

Note: This value is sent to and used by the device application, and is automatically derived from the application ID you select. Therefore, you must set this value correctly when using a domain with an application ID. If you set a domain, ensure it matches the domain of the packages needed by the application; otherwise, the application generates a `Package not found` error.

- **Activation code length** - the number of characters in the activation code.
 - **Activation expiration**- the number of hours the activation code is valid.
6. (Optional) Select the check box adjacent to **Specify activation code** to enter the code sent to the user in the activation e-mail. This value can contain letter A - Z (uppercase or lowercase), numbers 0 - 9, or a combination of both. Acceptable range: 1 to 10 characters.
7. Click **OK**

The application is registered or reregistered. SAP applications that have connections registered with SAP Mobile Server, can also have licenses counted by SAP License Audit service. For a list of SAP applications for which licenses are counted, see *SAP Applications Tracked with SAP License Audit* in *System Administration*.

Setting General Application Properties

Provide general application properties such as the application ID, description, security configuration and domain details while registering the application.

1. In the Application Creation Wizard, enter a unique **Application ID**.

Note:

- SAP recommends that application IDs contain a minimum of two dots ("."). For example, the following ID is valid: `com.sybase.mobile.app1`.
 - Application IDs cannot start with a dot ("."). For example, the following ID is invalid: `.com.sybase.mobile.app1`.
 - Application IDs cannot have two consecutive dots ("."). For example, the following ID is invalid: `com..sybase.mobile.app1`.
-

2. Enter a **Display name** and **Description** for the application.
3. Select the appropriate security configuration from the **Security Configuration** drop-down list.

For applications that do not require authentication, select the **anonymous** security configuration or the **Anonymous access** checkbox.

4. Select the appropriate domain from the **Domain** drop-down list.
5. (Optional) Assign one or more packages as desired.

Note: When an application ID is intended for use by Online Data Proxy, packages do not need to be assigned. .

6. (Optional) Modify application connection template settings.
 - a) Select **Configure additional settings**, and click **Next**.
 - b) To reuse the configuration of an existing template, select a **Base template** from the drop-down list.
 - c) Configure the application connection template properties as required.

Note: ODP applications require a proxy type connection endpoint. When modifying application connection template settings for an ODP application, you can automatically create the proxy connection endpoint by entering an OData URL as the Application Endpoint value in the connection template Proxy properties. This creates a proxy connection endpoint with the same name as the Application ID. If the ODP application uses an anonymous security configuration, the newly created connection endpoint will have the Allow Anonymous Access property set to True and the Address (URL) property set to the Application ID. If you want to create the proxy connection endpoint manually, leave the Application Endpoint property empty. You manually create the proxy connection endpoint through the SAP Control Center Domains node.

- Click **Finish** to register the application with the configured settings.

Application ID and Template Guidelines

Choose an appropriate application ID while registering application connection for use by native MBO, Hybrid App, or Online Data Proxy clients. Using an incorrect application ID results in failure when the client tries to activate itself.

Application Type	Guidelines
Hybrid App	<ul style="list-style-type: none"> 2.0.1 or earlier – leave the application ID empty. 2.1 or later – use preexisting HWC template, or, if you are using your own template, make sure that HWC is set as the application ID in the template. iOS sample container 2.1 or later – use the template you have created. The application ID used by the iOS sample container should match the application ID specified in registration.
Native MBO application	<ul style="list-style-type: none"> Previous to 2.1.2 – leave the application ID empty. This applies to native messaging-based application clients. 2.1.2 or later – (recommended) use the application connection template that is automatically created for the application. Otherwise, ensure you register the application connection with the correct template by verifying that application ID matches, and that the correct security configuration and domain are selected. Also, if using replication, set other template properties (such as synchronization-related properties in Connection category) as required. For Android native MBO applications, this recommendation applies starting with version 2.1.1.
Online Data Proxy	Register the application connection using the template created for the application. Existing templates can be found in the Applications > Application Connection Template tab.

Enabling and Configuring the Notification Mailbox

Configure the notification mailbox settings that allow SAP Mobile Server to transform e-mail messages into Hybrid App.

The notification mailbox configuration uses a listener to scan all incoming e-mail messages delivered to the particular inbox specified during configuration. When the listener identifies an e-mail message that matches the rules specified by the administrator, it sends the message as a Hybrid App to the device that matches the rule.

Note: Saving changes to the notification mailbox configuration deletes all e-mail messages from the account. Before proceeding with configuration changes, consult your e-mail administrator if you want to back up the existing messages in the configured account.

1. Log in to SAP Control Center.
2. In the left navigation pane, click **Hybrid Apps**.
3. In the right administration pane, click **Notification Mailbox**.
4. Select **Enable**.
5. Configure these properties:
 - **Protocol** – choose between POP3 or IMAP, depending on the e-mail server used.
 - **Use SSL** – encrypt the connection between SAP Mobile Server and the e-mail server in your environment.
 - **Server** and **Port** – configure these connection properties so SAP Mobile Server can connect to the e-mail server in your environment. The defaults are localhost and port 110 (unencrypted) or 995 (encrypted).
 - **User name** and **Password** – configure these login properties so SAP Mobile Server can log in with a valid e-mail user identity.
 - **Truncation limit** – specify the maximum number of characters taken from the body text of the original e-mail message, and downloaded to the client during synchronization. If the body exceeds this number of characters, the listener truncates the body text to the number of specified characters before distributing it. The default is 5000 characters.
 - **Poll seconds** – the number of seconds the listener sleeps between polls. During each poll, the listener checks the master inbox for new e-mail messages to process. The default is 60 seconds.
6. If you have added at least one distribution rule, you can click **Test** to test your configuration. If the test is successful, click **Save**.

Assigning and Unassigning a Hybrid App to an Application Connection

Assign a Hybrid App package to an application connection to make it available to a device user. Unassign the Hybrid App package when it is no longer required.

You can also assign Hybrid App packages indirectly through the application connection template. The set of packages assigned to an application connection will be a combination of packages assigned indirectly through the application connection template and directly through the application connection.

1. In the left navigation pane of SAP Control Center, click **Hybrid Apps** and select the Hybrid App to assign.
2. In the right administration pane, click the **Application Connections** tab.
3. Click **Assign**.
4. List the activation users to assign the Hybrid App package to.
Search for users by selecting the user property you want to search on, then selecting the string to match against. Click **Go** to display the users.
5. Select the user or users from the list that you want to assign the Hybrid App package to.
6. Click **OK**.
7. To set the Hybrid App package as the default application for an application connection, select the connection and click **default**.
Set a Hybrid App package as the default to run that application on the device as a single-purpose application. Single-purpose applications launch automatically when the user opens the Hybrid Web Container. This will be the only Hybrid App available on the device. You can select only one default per application connection.
8. To unassign a Hybrid App package, select the application connection and click **Unassign**.

Note: If you unassign the Hybrid App package that is set as the default, you may want to select a new default package.

9. Click **OK**.

Activating the Hybrid App

Hybrid App screen menus contain two menu item types: **Submit Hybrid App** (asynchronous) and **Online Request** (synchronous).

To complete the Hybrid App activation process, the last screen in the Hybrid App must have a **Submit Hybrid App** menu item. This is necessary for the device and server-side to activate the Hybrid App for the device.

A Hybrid App is considered to have been processed or activated only if it is closed with a **Submit Hybrid App** menu item, which may or may not be tied to a mobile business object (MBO).

Configuring Context Variables for Hybrid App Packages

The administrator can change some of the values of a selected variable, should the design-time value need to change for a production environment.

Which values are configurable depends on whether the developer hard-coded a set of user credentials or used a certificate.

1. In the left navigation pane, expand the **Hybrid Apps** folder and select the Hybrid App package to configure context variables for.
2. In the right administration pane, click the **Context Variables** tab.
3. Select the context variable to configure, then click **Modify**.

Context Variable	Description
SupUser	The valid Hybrid App application user ID that SAP Mobile Server uses to authenticate the user. Depending on the security configuration, SAP Mobile Server may pass that authentication to an EIS.
SupDomain	The name of the domain that the Hybrid App package is deployed to.
SupUnrecoverableErrorRetryTimeout	After sending a JSON request to SAP Mobile Server, if you receive an EIS code that indicates an unrecoverable error in the response log, the Hybrid App client throws an exception. A retry attempt is made after a retry time interval, which is set to three days by default. Select this property to change the retry time interval.
SupThrowCredentialRequestOn401Error	The default is true , which means that an error code 401 throws a <code>CredentialRequestException</code> , which sends a credential request notification to the user's inbox. If this property is set to false , error code 401 is treated as a normal recoverable exception.

Context Variable	Description
SupThrowBadHttpHeadersOn412Error	The default is true , which means that an error code 412 throws a <code>BadHttpRequestException</code> . If this property is set to false , error code 412 is treated as a normal recoverable exception.
SupRecoverableErrorRetryTimeout	After sending a JSON request to SAP Mobile Server, if you receive an EIS code that indicates a recoverable error in the response log, the Hybrid App client throws an exception. A retry attempt is made after a retry time interval, which is set to 15 minutes by default. Select this property to change the retry time interval.
SupPassword	The valid Hybrid App application user password that SAP Mobile Server uses to authenticate the user. Depending on the security configuration, SAP Mobile Server may pass that authentication to an EIS. An administrator must change development/test values to those required for a production environment.
SupPackages	The name and version of the MBO packages that are used in the Hybrid App. This cannot be changed.
SupMaximumMessageLength	Use this property to increase the allowed maximum Hybrid App message size. Limitations vary depending on device platform: <ul style="list-style-type: none"> • For BlackBerry 5, the limit is 512KB. • For Windows Mobile the limit is 500KB. • For BlackBerry 6 and Android, the limit depends on the memory condition of the device. Large message may result in an out of memory error.
SupWorkflowVersion	The version number of the Hybrid App package.

4. In the Context Variable dialog, change the value of the named variable and click **OK**.

Changing Hard Coded User Credentials

The administrator can change hard coded user credentials assigned at design time if the design time value needs to change for a production environment.

1. In the left navigation pane, expand the **Hybrid Apps** folder and select the Hybrid App package to configure context variables for.
2. In the right administration pane, click the **Context Variables** tab.
3. Select one or both of the variables: SupUser or SupPassword, and click **Modify**.
4. Type the new value and click **OK**.

Adding a Certificate File to the Hybrid App Package

The administrator can change the credential certificate assigned at design time.

Note: SAP recommends that you use Internet Explorer to perform this procedure.

1. In the left navigation pane, expand the **Hybrid Apps** folder and select the Hybrid App package to configure context variables for.
2. In the right administration pane, click the **Context Variables** tab.
3. Select **SupPassword** and click **Modify**.
4. Select **Use certificate-base credentials** and click **Browse** to find and upload a certificate file.
5. Set the value for **Certificate password** and click **OK**.

On the Context Variables page, the read-only values of SupUser, SupCertificateSubject, SupCertificateNotBefore, SupCertificateNotAfter, and SupCertificateIssuer change to reflect values of the new certificate and password you set.

End to End Trace and Performance

The SAP passport handling functionality allows for an end to end trace of data communication from the client to the back-end.

The `hwc.e2eTrace` JavaScript APIs enable or disable end-to-end trace and the ability to upload and view the trace file. SAP Mobile Server must be configured with SAP Solution Manager to upload and view this trace. See *Configuring SAP Mobile Server Performance Properties* in *SAP Control Center for SAP Mobile Platform*.

Note: End to end trace is supported on Android and iOS only.

The performance library provides the ability to capture performance metrics of the device while the Hybrid Web Container is running. Administrators can use this information to troubleshoot performance related issues.

These metrics are collected when the performance agent is enabled:

- totalTime [ms]
- networkTime [ms]
- totalCpuTime [ms]

- roundTrips
- totalBytes
- sentBytes
- receivedBytes
- memMax

Enabling the Performance Agent on the Device

The performance setting on the device gives administrators a mechanism to collect performance counters when running Hybrid Apps.

Note: The performance agent is not supported on Windows Mobile devices.

Note: To enable the performance setting on BlackBerry and Android, an SD card must be installed on the device.

1. Go to the Hybrid App settings screen.
2. Click the menu key and select **Advanced**.
3. Select **Performance** to start the performance agent.
4. Unselect **Performance** to create the performance log.

The performance numbers are stored in memory and saved to a file when you stop the performance library, either on the device or through the `stopInteraction` JavaScript API. View the performance logs in SAP Control Center. See *Tracing Application Connections*.

Tracing Application Connections

Send a request to SAP Mobile Server to retrieve log files for an application connection.

1. In the left navigation pane, select the **Applications** node.
2. In the right administration pane, click **Application Connections** tab.
3. Select an application connection, and click **Get Trace**.

Note: If the client application does not support tracing, you cannot trace the application connection. To determine if the client application supports tracing, review the **Capabilities** properties for the application connection. If the **Application Supports Client Callable Components** property has a value of `False`, tracing is not supported.

The application connection status must be "online" to retrieve the logs.

4. Click **OK**.
5. When the application connection is online, you can view the log contents in SAP Control Center by retrieving the server log for the domain that the application connection belongs to. The trace logs will be identified by one of the following values in the Category column of the Server log tab: `PerformanceAgent`, `MOCA`, or `SQLTrace`. Trace logs can also be

viewed in the file system. The default location for single node and cluster installations is *SMP_HOME\Servers\UnwiredServer\logs\ClientTrace*.

Build a Customized Hybrid Web Container Using the Provided Source Code

Use the provided source code to build your own customized user interface and configure other resources in the development environment of your choice.

Building the Android Hybrid Web Container Using the Provided Source Code

The Hybrid Web Container in this procedure is a sample container provided with the SAP Mobile Platform Mobile SDK installation.

Prerequisites

- Install the Android SDK version 2.2, API Level 8. You can get the Android SDK at <http://developer.android.com/sdk/index.html>.
- If you are developing in Eclipse, install the ADT Plug-in for Eclipse.

Task

This example uses Eclipse as the development environment, but you can use any development environment.

1. Open Eclipse and select **File > Import**.
2. Expand the **General** folder, choose **Existing Projects into Workspace**, and click **Next**.
3. Choose **Select archive file**, browse to `SMP_HOME\MobileSDK<version>\HybridApp\Containers\Android\`, and select `Android_HWC_<version>.zip`.

4. Click **Finish**.

A Hybrid Web Container project folder is added to Workspace Navigator. You may receive an error indicating that the source folder `gen` is missing. If so, add an empty folder named `gen` to the `src` folder in the project.

5. Open the `local.properties` file in the main directory of the project. This file contains a non-commented line, `sdk.dir = <filepath>`. Verify the `<filepath>` matches the filepath to your installation of the Android SDK.
6. If you receive an Android requires compiler compliance level 5.0 or 6.0. Found '1.4' instead. Please use Android Tools > Fix Project Properties error, follow the instructions and then clean the project.

Build a Customized Hybrid Web Container Using the Provided Source Code

7. If you receive errors of the type ... must override a superclass method, make sure the Java compiler has its compliance set to 1.6.
 - a) Right-click the **HybridWebContainer** project and select **Properties**.
 - b) Go to the Java Compiler section and set the Compiler compliance level to 1.6.
 - c) Rebuild the project.

Building the Android Hybrid Web Container Outside of Eclipse

You can build the Android Hybrid Web Container independent from SAP Mobile Platform.

1. Open a command prompt and navigate to the base directory of the Hybrid Web Container project.
2. Run either the **ant debug** or **ant release** command, depending on whether you want to debug or release the Hybrid Web Container.

You can download Apache Ant from <http://ant.apache.org/bindownload.cgi>, if necessary.

A file named either `HybridWebContainer-debug.apk` or `HybridWebContainer-release-unsigned.apk` (depending on the command you used) is added to the `bin` folder. If a file already exists with that name, it is overwritten.

3. Use Android Debug Bridge (ADB), which is included in the Android SDK installation, to install the `.apk` to the emulator.
 - a) Launch an Android Virtual Device (AVD) that does not have the Hybrid Web Container installed (or uninstall it if it is installed).
 - b) In the Command Prompt window, navigate to the folder that contains the `adb.exe` file, which should be in the `.../android-sdk/platform-tools/` folder.
 - c) Execute: **adb install <path>**, where *<path>* is the full filepath to the `HybridWebContainer.apk` file.

Building the BlackBerry Hybrid Web Container Using the Provided Source Code

You can use the provided BlackBerry Hybrid Web Container template to build a custom user interface and configure other resources.

Prerequisites

- Install the BlackBerry Plug-in for Eclipse. See <https://developer.blackberry.com/java/download/eclipse?IID=DEVJVA1223>.
- Register the device in SAP Control Center.

Task

This example uses Eclipse as the development environment. If you use another development environment, the steps might vary.

1. Extract the files from `SMP_HOME\MobileSDK<version>\HybridApp\Containers\BB\BB_HWC_<version>.zip`
2. In Eclipse, import the BlackBerry Hybrid Web Container template as a legacy BlackBerry project:
 - a) Select **File > Import**.
 - b) Expand the **BlackBerry** folder.
 - c) Select **Import Legacy BlackBerry Projects**.
 - d) Click **Next**.
 - e) Specify the JRE and, in the BlackBerry Workspace field, browse to the `HWCtemplate.jdw` file and select the project to import.
 - f) Select **Copy BlackBerry projects into workspace** to create a copy of the imported project in the Eclipse workspace.
 - g) Click **Finish**.
3. Supply a signing key.

Supplying a Signing Key

You must supply a BlackBerry code signing key from BlackBerry to access the persistent store.

1. Go to <https://www.blackberry.com/SignedKeys/codesigning.html> to obtain a signing key and import into Eclipse following BlackBerry's instructions.

Once you import your signing key, you must change some code to let the Hybrid Web Container know which keys you are using.
2. Open the `CustomizationHelper.java` file for editing.
3. Find the method named `getCodeSignerId()` and update it to return the name of your key.
4. Add the key file to your project so it is included in the `.cod` file.

Building the iOS Hybrid Web Container Using the Provided Source Code

Build a sample Hybrid Web Container.

Prerequisites

- Register the device in SAP Control Center.
- Have access to a Mac with a supported version of Xcode and the iOS SDK.

See *Supported Hardware and Software* for the most current version information for mobile device platforms and third-party development environments.

Task

1. On your Mac, connect to the Microsoft Windows machine where SAP Mobile Platform is installed:
 - a) In the Apple menu, click **Go > Connect to Server**.
 - b) Enter the name or IP address of the machine.
For example, `smb://machine DNS name` or `smb://IP Address`.
2. Copy the `iOS_HWC_version.tar.gz` archive from `SMP_HOME\MobileSDKversion\HybridApp\Containers\iOS\` to a location on your Mac.
In the archive file name, *version* is the current SAP Mobile Server version number. For example, `iOS_HWC_2.3.0.tar.gz`.
3. Unpack `iOS_HWC_version.tar.gz`.
The extraction creates a `HybridWebContainer` directory.
4. In the `HybridWebContainer` directory, double-click **HWC.xcodeproj** to open it in the Xcode IDE.
5. If you are building for a device, you must add provisioning profiles to the project to be able to sign the executable.
 - a) In Xcode, click the **HWC** project and select the HWC target.
 - b) Select the **Build Settings** tab.
 - c) Under the Code Signing section, add code-signing identities for each configuration (Debug, Release, or Distribution) you want to build, depending on how you will deploy the app.

When you build the Hybrid Web Container using your provisioning profile, you are creating your own version of the application. You can reuse the bundle ID that is distributed with the HWC template project, but you cannot upgrade your custom-built application through the normal means.

Build a Customized Hybrid Web Container Using the Provided Source Code

The reason for this is because on iOS the Keychain is used to store information and Keychain rights depend on the provisioning profile used to sign your application. Therefore, you should consistently use the same provisioning profile across different versions of your application. Follow the instructions in *Using Multiple Hybrid Web Containers on the Same iOS Device* when you build the HWC template source.

6. In Xcode, click **Product** > **Build** to build the project.

Building the Windows Mobile Hybrid Web Container Using the Provided Source Code

Use the provided Windows Mobile Hybrid Web Container template to build your own customized user interface and configure other resources.

1. Unpack SMP_HOME\MobileSDK<version>\HybridApp\Containers\WM\WM_HWC_<version>.zip into a local folder.
2. Include custom code files in your template project:
 - a) In Visual Studio, open Solution Explorer and select the template project.
 - b) Click the **Show All Files** button and select all files in the CustomCode folder.
 - c) With all files selected, right-click and choose **Include In Project**.
3. Specify the signing for the template project:
 - a) Right-click the project in the Solution Explorer and choose **Properties**.
 - b) Open the Signing tab, and select an existing key file or create a new one.
4. Right-click the project and choose **Add Reference**.
5. Click **Browse**, select **HybridAppLib.dll**, and click **OK**.

Install and Configure the Hybrid Web Container On the Device

To enable deploying Hybrid App packages to a device, you must download, install, and configure the Hybrid Web Container on the device.

Deploy the Hybrid Web Container to devices and register the devices with SAP Mobile Server. You can use Afaria® to install the container on devices for enterprise deployment. For information on setting up an Afaria environment, see *Provisioning With Afaria* in *Mobile Application Life Cycle*.

See the configuration procedure for your device type.

Preparing Android Devices for the Hybrid Web Container

Install the Hybrid Web Container on the Android device using the Android SDK. In the Settings for your Android device, disable all keyboards except the Android keyboard.

Installing the Hybrid Web Container on Android Devices

Use the Android SDK Manager to install Hybrid Web Container application files.

To install the Android Hybrid Web Container on your Android device:

1. Connect the device.
2. Install the Android SDK.
3. Run `platform-tools\adb` and install `SMP_HOME\MobileSDK<version>\HybridApp\Containers\Android\HybridWebContainer.apk`.

For example:

```
C:\Android\android-sdk\platform-tools\adb install ^
SMP_HOME\MobileSDK<version>\HybridApp\Containers\Android
\HybridWebContainer.apk
```

Configuring the Android Emulator

Configure an Android emulator for testing a Hybrid App package.

Note: The steps or interface may be different depending on the Android SDK version you are using.

1. Install the Android SDK.
 - a) Go to <http://developer.android.com/sdk/>.
 - b) Download the Android SDK (for example, `installer_r21-windows.exe`).

Note: Do not download the larger SDK starter package (ADT Bundle for Windows). The starter package includes not only the SDK but also the ADT plug-in for Eclipse and a more recent platform than the one shown in this tutorial.

- c) In Windows Explorer, double-click the downloaded installer to run it.

Note where the SDK is installed on your system, for example,

C:\Program Files\Android\android-sdk.

2. Install the SDK platform tools:

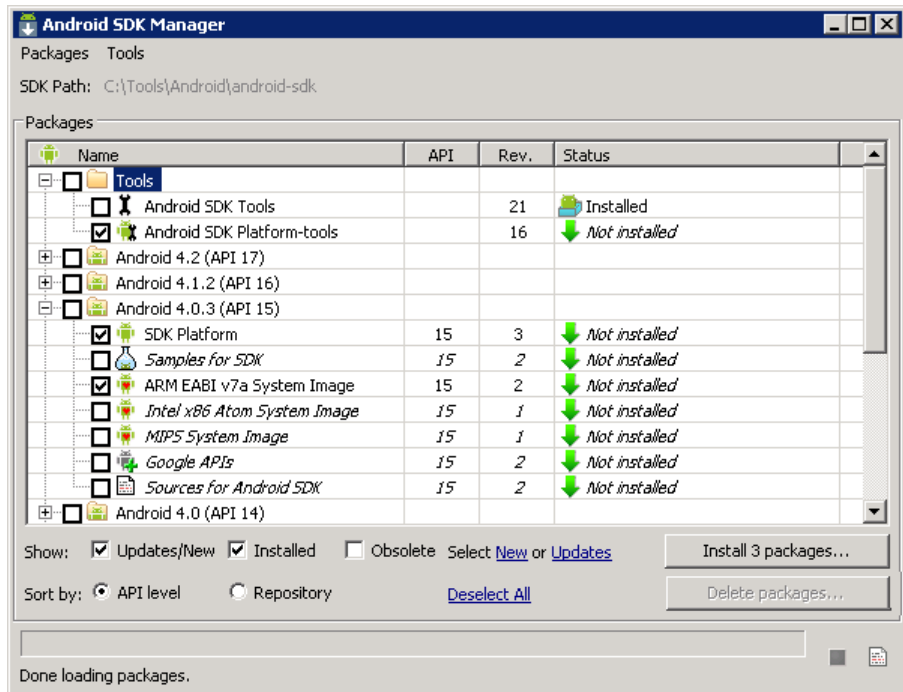
- a) Run the Android SDK Manager, *android-sdk\SDK Manager.exe*.
 b) In the Android SDK Manager, expand Tools and select **Android SDK Platform-tools**.

Android SDK Tools should already be installed.

- c) Expand **Android 4.0.3 (API 15)** and select these packages:

- **SDK Platform.**
- **ARM EABI v7a System Image.**

- d) Click the **Install *n* packages** button.



- e) In Choose Packages to Install, select **Accept All**, then click **Install**. Close the log window when done.

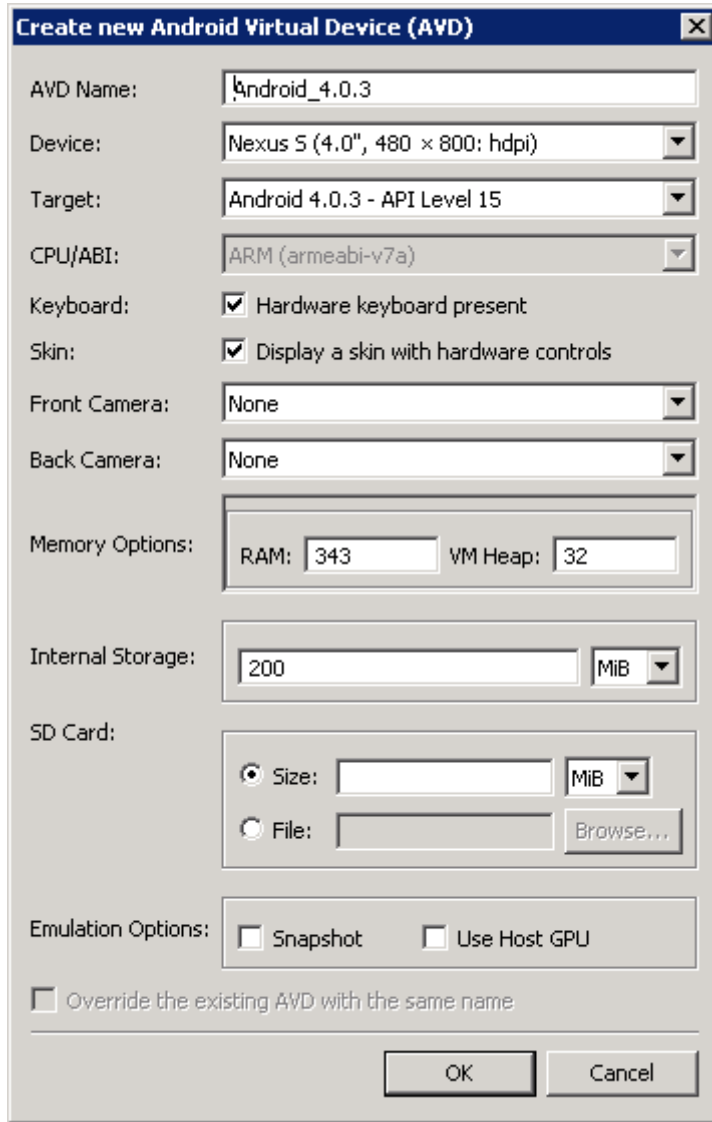
- f) Close the Android SDK Manager.

3. Run the Android Virtual Device Manager, *android-sdk\AVD Manager.exe*.

4. Configure and start an Android emulator instance.

- a) In the AVD Manager, click **New**.
- b) In the Create new Android Virtual Device window, enter an AVD name and select a supported Android device for this instance.

For example:



Create new Android Virtual Device (AVD)

AVD Name:

Device:

Target:

CPU/ABI:

Keyboard: ☒ Hardware keyboard present

Skin: ☒ Display a skin with hardware controls

Front Camera:

Back Camera:

Memory Options: RAM: VM Heap:

Internal Storage:

SD Card:

☒ Size:

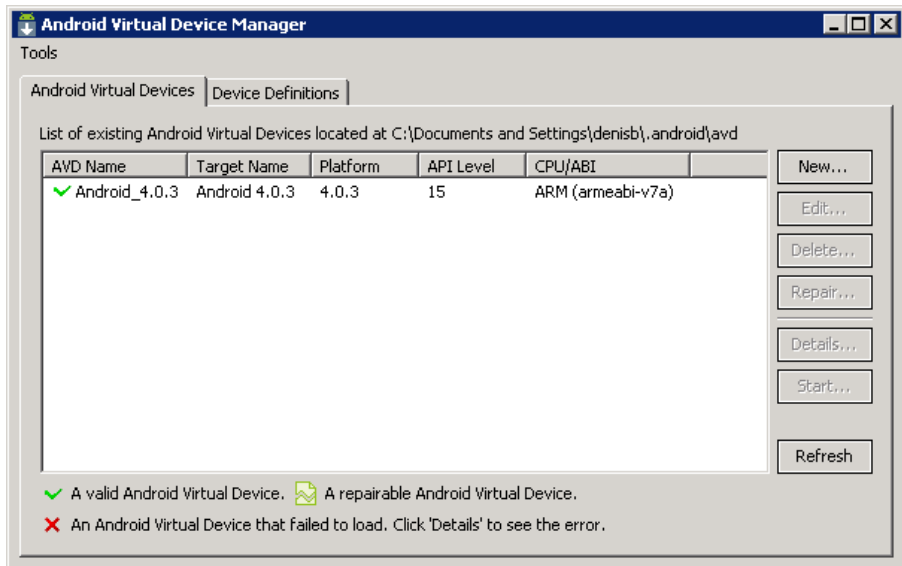
☐ File:

Emulation Options: ☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

- c) Click **OK** to add the instance to the AVD Manager.

Install and Configure the Hybrid Web Container On the Device



- d) Select the new virtual device and click **Start**.
- e) In Launch Options, click **Launch** to open the Android emulator screen.
5. Install the Hybrid Web Container on the emulator instance:
 - a) With the Android emulator running, open a command prompt window.
 - b) Run `android-sdk\platform-tools install SMP_HOME\MobileSDKversion\HybridApp\Containers\Android\HybridWebContainer.apk`.

For example:

```
C:\Android\android-sdk\platform-tools>adb install ^
C:\SAP\MobilePlatform\MobileSDKversion\HybridApp\Containers
\Android\HybridWebContainer.apk
```

Preparing BlackBerry Devices for the Hybrid Web Container

Install the Hybrid Web Container on the BlackBerry device using BlackBerry Desktop Manager.

Prerequisites

For prerequisites and complete information about provisioning BlackBerry devices see *Setting Up BES Environments for SAP Mobile Platform Applications* in *Mobile Application Life Cycle*.

Task

1. Connect the BlackBerry device to the computer that contains the Hybrid Web Container for BlackBerry.
2. Run the BlackBerry Desktop Manager following the instructions in the RIM documentation.
3. In the BlackBerry Desktop Software, select **Application Loader**.
4. Under Add/Remove Applications, select **Start**.
5. Browse to the location on your local machine or network that contains the Hybrid Web Container `HybridWebContainer.cod` and `HybridWebContainer.alx` container files, `<SMP_HOME>\MobileSDK<version>\HybridApp\Containers\BB`.
6. Select the files and click **Open**.
The application is listed on the Application Loader wizard.
7. Click **Next**.
8. Click **Finish**.
9. Restart your BlackBerry device.

Installing the Hybrid Web Container on BlackBerry Devices Over the Air

Your system administrator must provide the appropriate information for installing the Hybrid Web Container over the air, and the BlackBerry Exchange Server (BES) must be available.

Note: For information about provisioning BlackBerry devices see *Setting Up BES Environments for SAP Mobile Platform Applications* in *Mobile Application Life Cycle*.

The administrator stages the OTA files in a Web-accessible location and notifies BlackBerry device users through an e-mail message with a link, or a URL to the Hybrid Web Container installation file. This can be accomplished by pointing the BlackBerry browser to the `HybridWebContainer.jad` file. This single JAD and associated files for this type of deployment are located in `<SMP_HOME>\MobileSDK<version>\HybridApp\Containers\BB\OTA`.

Enabling Hybrid Web Container Message Notification

On each BlackBerry device, customize the alert profile to notify users when a new Hybrid Web Container message is received.

By default, Hybrid Web Container messages do not trigger BlackBerry sounds and alerts. The only indication of a new message is a change to the home screen icon. To add notifications, each BlackBerry user can create a new alert profile.

This topic describes how to configure alert profiles for Hybrid Web Container messages on a BlackBerry 9800 running BlackBerry 6 or a BlackBerry 9900 running BlackBerry 7.1: The

Install and Configure the Hybrid Web Container On the Device

steps are similar for other supported BlackBerry devices. For information about all devices, see the BlackBerry Manuals page at <http://docs.blackberry.com>.

1. On the home screen, select the **Sound and Alert** application.
2. Select **Change Sounds and Alerts**.
3. Select **Profile Management**.
4. Select **Add Custom Profile**.
5. In New Custom Profile, enter a name for the new profile in **Name**.
6. Expand **Other Applications - Notifiers** and choose **Hybrid Web Container**.
7. Configure the sound, visual, and other alert options that you want.
8. Save your changes and close the profile.
For example, open the menu and choose **Close**. When prompted, choose the **Save** option.
9. Activate the customized profile: return to the home screen, select the **Sound and Alert** application again, and choose the new profile.

Configuring the BlackBerry Simulator for Hybrid Web Containers

Copy the `HybridWebContainer.cod` file to the BlackBerry Simulator directory.

Prerequisites

MDS must be running.

Task

1. Start the BlackBerry simulator.
2. From **File > Load BlackBerry Application or Theme**.
3. Navigate to `SMP_HOME\MobileSDK<version>\HybridApp\Containers\BB`.
4. Select the `HybridWebContainer.cod` file, then click **OK**.

Preparing iOS Devices for the Hybrid Web Container

Install the Hybrid Web Container on the device using the App Store, or use the source code provided for the Hybrid Web Container to deploy to the iOS simulator from the Xcode project.

Complete these prerequisites before provisioning the Hybrid Web Container:

- Determine your security policy – SAP Mobile Platform provides a single administration console, SAP Control Center, which allows you to centrally manage, secure, and deploy applications and devices. Device user involvement is not required and you can maintain the authorization methods you already have in place. See *Security > Device Security*.

- Register each application connection using SAP Control Center – application connections pair an application with a device. See *SAP Control Center for SAP Mobile Platform* documentation.

Installing the Hybrid Web Container on the iOS Device

How you install the Hybrid Web Container on your iOS device depends on how your company provisions the application.

Your company will choose a method for provisioning the application. Your system administrator determines how you obtain and install the Hybrid Web Container. The possible methods include:

- Downloading and installing the free version of the Hybrid Web Container from the Apple App Store. The free version should not be used for enterprise deployment.
- Obtaining a copy of the application on your corporate network or through a link in an e-mail message, then using iTunes to install and synchronize it to your device. This mechanism should be used for enterprise deployment and is based on the application built using the XCode project, which is included as part of SAP Mobile Platform installation.

Installing the Hybrid Web Container from the Apple App Store

Install the Hybrid Web Container from the Apple App Store.

This is a free version of the Hybrid Web Container and should not be used for enterprise deployment.

1. On your device, on the iOS home page, tap **App Store**.
2. Search for **SAP Hybrid Web Container**.
3. In the search results, find the version of the Hybrid Web Container to install and click **Free**.
4. Tap **Install** to download the application.
5. In **Settings > HWC<version>**, for Connection Info, enter:
 - Server Name – the machine that hosts the server where the mobile application project is deployed.
 - Server Port – SAP Mobile Server port number. The default is 5001.
 - Farm ID – the farm ID you entered when you registered the application connection in SAP Control Center.
 - Protocol – HTTP or HTTPS. The protocol with which to connect to the Relay Server or the reverse proxy server. The default is HTTP.
 - (Optional) URL Suffix – the URL suffix used to connect to a Relay Server or the reverse proxy server. Get this information from your administrator. See *Device Advanced Properties* in *System Administration*.
6. Scroll to the page that contains the **HWC** icon, then tap to launch.
7. Enter your personal identification number (PIN).

This PIN is a security measure to safeguard your company's data.

Install and Configure the Hybrid Web Container On the Device

- The PIN must be at least six digits and cannot be consecutive digits (for example, 123456), or same digit (for example, 111111).
- (First time/reinstallation) Create a PIN in the **Password** field, then verify it in the second field.
- (Second or subsequent logins) Enter the PIN in the **Password** field. Select **Change Password** to change the PIN. You can change the PIN once you enter the current PIN.

The **HWC** page appears.

8. Tap **Messages** to view messages/notifications.
9. (Optional) If instructed by your system administrator, enable notifications on your device.

Installing the Hybrid Web Container Using iTunes

Install the Hybrid Web Container using iTunes.

1. Launch iTunes.
2. Download the application from your corporate network to your Applications library.
3. Sync the application to your Apple mobile device.
4. Specify the connection settings in **Settings > Hybrid App**.

Preparing Windows Mobile Devices for the Hybrid Web Container

Install the Hybrid Web Container on the Windows Mobile device.

Installing the Hybrid Web Container on Windows Mobile Devices

Install and configure the Hybrid Web Container required to prepare a Windows Mobile device to run Hybrid Apps.

1. Navigate to `SMP_HOME\MobileSDK\HybridApp\Containers\WM`.
2. Copy the Windows Mobile Professional device file, `HybridWebContainer.cab`, to the device's **My Documents** folder.
3. Cradle the Windows Mobile device.
4. Connect a USB cable between the PC and device, and transfer the `.cab` file.
5. Open the `HybridWebContainer.cab` file from the Windows Mobile device. This installs the container.
6. In Programs, click the Hybrid Web Container icon and click **Settings**.
7. In the Connection screen, enter the connection settings. These settings should match the values you used when you registered the device in SAP Control Center.

Note: Click **Menu** and select **Show Log** to view the container log. This is useful for checking the connection, or retrieving other debugging information.

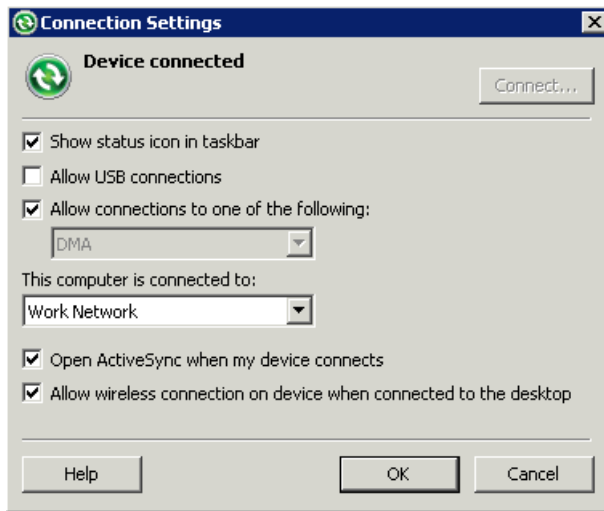
Installing Microsoft Synchronization Software

Install and configure Microsoft synchronization software so you can deploy and run an application on a Windows Mobile emulator.

Note: These instructions describe how to install Microsoft ActiveSync for Windows XP. If you are using Windows Vista, Windows 7, or Windows 2008, install Virtual PC 2007 SP1 and Windows Mobile Device Center to manage synchronization settings. Download the Windows Mobile Device Center from <http://www.microsoft.com/en-us/download/details.aspx?id=15> and follow Microsoft instructions for installing and using that software instead of this procedure.

1. Download Microsoft ActiveSync:
 - a) In a Web browser, open the Windows Phone page at <http://www.microsoft.com/windowsmobile/en-us/help/synchronize/device-synch.msp>.
 - b) Follow the instructions to select and download the sync software for the system's operating system. Windows XP requires ActiveSync version 4.5.
 - c) In the Windows Phone downloads page, click the **ActiveSync** button.
 - d) Download the ActiveSync installation file and save it to your local system.
2. Run the downloaded installation file.
For example, double-click **setup.msi** in Windows Explorer.
3. When the installation is complete, restart the system.
4. Start ActiveSync if it does not start automatically.
For example, click **Start > Programs > Microsoft ActiveSync**.
5. Click **File > Connection Settings**.
6. Select **Allow connections to one of the following**, then select **DMA**.
7. Select **Work Network** for "This computer is connected to".

Install and Configure the Hybrid Web Container On the Device

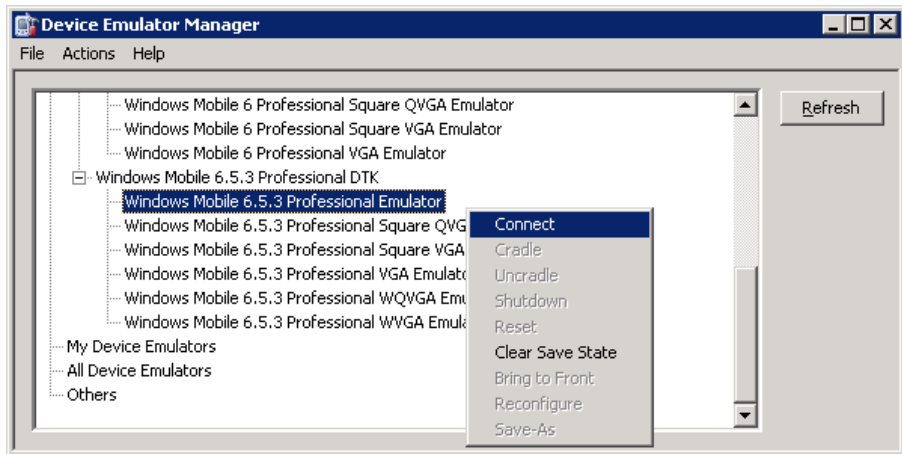


8. Click **OK**.

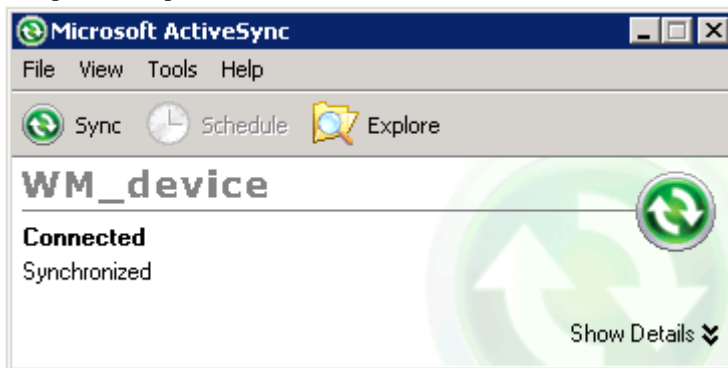
Installing the Hybrid Web Container on the Windows Mobile Emulator

Install the Hybrid Web Container software on your emulator.

1. Start the synchronization software.
For example, on Windows XP, start Microsoft ActiveSync. On Windows Vista, Windows 7, or Windows 2008, start the Windows Mobile Device Center.
2. Start the Device Emulator Manager and select an emulator to run.
For example:
 - a. Double-click `C:\Program Files\Microsoft Device Emulator\1.0\dvcemumanager.exe`.
 - b. In the Device Emulator Manager, right-click the device you want to run and choose **Connect** to open the emulator.



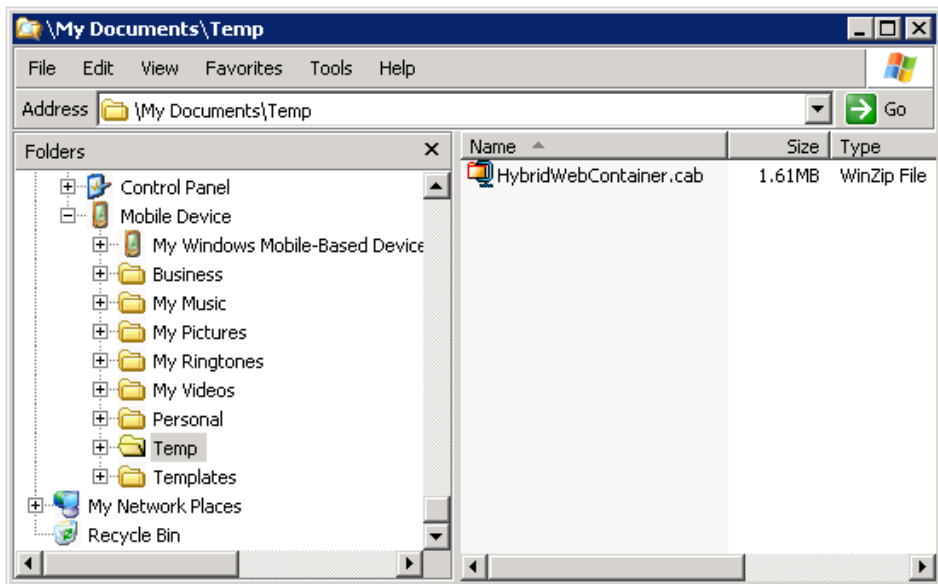
- c. In the Device Emulator Manager, right-click the device again and click **Cradle**.
3. The synchronization software runs and connects to your device. If the Synchronization Setup wizard opens, follow the instructions and click **Finish**.



4. Run the downloaded Microsoft .NET Compact Framework Redistributable file to install the .NET Compact Framework on your running emulator. Follow the setup wizard instructions, and click **Finish** to close the wizard when you are done.

Note: Be sure to run the installer while your emulator is running; otherwise the .NET Compact Framework Redistributable is not installed correctly.

5. Go to `SMP_HOME\MobileSDK<version>\HybridApp\Containers\WM` and copy the `HybridWebContainer.cab` file to a folder on mobile device folder on your system.
For example:



6. On the device emulator, open File Explorer and browse to the folder to which you copied the CAB file. Click the file once to install the Hybrid Web Container on your emulator.

Configure Connection Settings on the Device

Configure the connection settings for the Hybrid Web Container on the device.

See the topic for your platform.

Configuring Android Connection Settings

Configure the connection settings for the Hybrid Web Container.

1. Click the **HWC** icon on the applications screen, then select **Settings**.
2. In the basic authentication screen, enter the user name and password if you are prompted.
3. Click **Registration** to choose from the registration options:
 - Manual – enter connection settings and register manually.
 - Automatic (Password) – enter the password for automatic registration.
 - Automatic (Afaría Certificate) – register using an Afaria certificate.
 - Automatic (Local Certificate) – register using a local certificate.
4. Enter the settings for the Hybrid Web Container:
 - Server Name – the machine that hosts the server where the mobile application project is deployed.

Install and Configure the Hybrid Web Container On the Device

- Server Port – SAP Mobile Server port number. The default is 5001.
- Farm ID – the farm ID you entered when you registered the application connection in SAP Control Center.
- Protocol – HTTP or HTTPS. The protocol with which to connect to the Relay Server or the reverse proxy server. The default is HTTP.
- (Optional) URL Suffix – the URL suffix used to connect to a Relay Server or the reverse proxy server. Get this information from your administrator. See *Device Advanced Properties* in *System Administration*.

Select **Save** to save the settings.

5. (Optional) Configure trace and performance settings:

Note: To enable the performance agent, an SD card must be installed.

- a) In the Settings screen, click the menu key and select **Advanced**.
 - b) Select **Trace** to enable SAP Passport end to end trace.
 - c) Click **Level** to choose the log level.
 - Low – focuses on response-time-distribution analysis, in other words, how much time is spent on each server component, or the specific location of a bottleneck.
 - Medium – (default) gives performance analysis. Performance traces are triggered on the server-side.
 - High – gives functional analysis and has detailed functional logging and tracing.
 - d) Select **Performance** to enable the performance agent.
6. Start the application, then view the settings log to verify that the connection is active.
- From the application, tap **Settings > Show Log**.

Configuring BlackBerry Connection Settings

Configure the connection settings for the Hybrid Web Container.

1. Click the **Hybrid Web Container** icon on the applications screen, then press the **Menu** key and select **Settings**.
2. Enter the settings for the Hybrid Web Container:
 - Registration – choose from:
 - Manual – enter connection settings and register manually.
 - Auto (Password) – when you select this option, the Password field is enabled. Enter your password.

Note: The Activation Code and Enable Automatic Registration options are mutually exclusive. If you use a password for automatic registration, you cannot enter an activation code, and vice versa.

- Auto (Afaria Cert) – register using an Afaria certificate. When you choose this option, these fields are enabled:
 - Common name

Install and Configure the Hybrid Web Container On the Device

- Challenge code
- Auto (Local Cert) – register using a local certificate.
- Server Name – the machine that hosts the server where the mobile application project is deployed.
- Server Port – SAP Mobile Server port number. The default is 5001.
- Farm ID – the farm ID you entered when you registered the device in SAP Control Center.
- User Name – the user you registered in SAP Control Center.

Note: When there are multiple application connection templates for the same APP ID, and you need to establish a connection using the anonymous security configuration, you must include the security configuration in the user name, in this format:
`anonymous@anonymous.`

- Activation Code – the activation code for the user, for example, 123.
 - Protocol – the protocol with which to connect to the Relay Server or the reverse proxy server. Choose from:
 - HTTP
 - HTTPS
 - (Optional) URL Suffix – the URL suffix used to connect to a Relay Server or the reverse proxy server. Get this information from your administrator. See *Device Advanced Properties in System Administration*.
3. Select **Menu > Save** to save the settings.
 4. (Optional) In the settings screen, click the menu key and select **Advanced** to turn on the performance agent.

Note: To enable the performance agent, an SD card must be installed.

5. Start the application, then view the settings log to verify that the connection is active.
In the Hybrid Web Container, select **Settings**. On the connection settings screen, select **Show Log**.

Configuring iOS Connection Settings

Configure the settings for the Hybrid Web Container.

1. Go to the device Settings screen and click **HWC**.
2. In the basic authentication screen, enter the user name and password if you are prompted.
3. Enter the settings for the Hybrid Web Container:
 - Server Name – the machine that hosts the server where the mobile application project is deployed.
 - Server Port – SAP Mobile Server port number. The default is 5001.

- Farm ID – the farm ID you entered when you registered the application connection in SAP Control Center.
 - Protocol – HTTP or HTTPS. The protocol with which to connect to the Relay Server or the reverse proxy server. The default is HTTP.
 - (Optional) URL Suffix – the URL suffix used to connect to a Relay Server or the reverse proxy server. Get this information from your administrator. See *Device Advanced Properties* in *System Administration*.
4. Click in the **Registration Method** field to choose a registration method:
 - Manual – enter connection settings and register manually.
 - Automatic (Password) – when you select this option, the Password field is enabled.
 - Automatic (Afaria Certificate) – allows you to register using an Afaria certificate.
 5. Click the **HWC** icon to go back to the settings screen.
 6. If you chose manual registration, enter your user name and activation code.

Note: When there are multiple application connection templates for the same APP ID, and you need to establish a connection using the anonymous security configuration, you must include the security configuration in the user name, in this format:

anonymous@anonymous.

The activation code and password for automatic registration are mutually exclusive. If you use a password for automatic registration, you cannot enter an activation code, and vice versa.

-
7. If you chose automatic registration, enter your user name and password.
 8. If you chose automatic registration with an Afaria certificate, enter the common name and challenge code for the Afaria certificate.

Configuring Windows Mobile Connection Settings

Configure the connection settings.

Prerequisites

Install the Hybrid Web Container CAB file.

Task

1. Select **Start > Programs**.
2. Click the Hybrid Web Container icon.
3. Click **Settings**.
4. In the Connection screen, enter the connection settings:
 - Server Name – the machine that hosts the server where the mobile application project is deployed.
 - Server Port – SAP Mobile Server port number. The default is 5001.

Install and Configure the Hybrid Web Container On the Device

- Farm ID – the farm ID you entered when you registered the device in SAP Control Center.
- User Name – the user you registered in SAP Control Center.

Note: When there are multiple application connection templates for the same APP ID, and you need to establish a connection using the anonymous security configuration, you must include the security configuration in the user name, in this format:
`anonymous@anonymous.`

- Registration – choose from:
 - Manual – enter connection settings and register manually.
 - Automatic – when you select this option, the Password field is enabled.

Note: The Activation Code and Enable Automatic Registration options are mutually exclusive. If you use a password for automatic registration, you cannot enter an activation code, and vice versa.

- Certificate – allows you to register using a certificate.
- Activation Code – the activation code for the user, for example, 123.
- Password – this field is enabled if you chose Automatic registration. Enter your password.
- Certificate – this field is enabled if you chose Certificate as the registration type. Choose your certificate. The User Name field is populated with the certificate name.
- Protocol – the protocol with which to connect to the Relay Server or the reverse proxy server. Choose from:
 - HTTP
 - HTTPS

5. Click **Advanced** for these options:

- Allow roaming – the device is allowed to connect to server while roaming. By default, this is set to true.
- (Optional) URL Suffix – used to connect to a Relay Server or the reverse proxy server. Get this information from your administrator. See *System Administration > System Reference > Application Connection Properties > Device Advanced Properties*.
- Keep alive – the frequency used to maintain the wireless connection, in seconds. Acceptable values: 30 to 1800. The default is 240.

6. Click **Save**.

- ### 7. Start the Hybrid App, then view the settings log to verify that the connection is active.
- In the Settings screen, click **Menu > Show Log**.

Install and Test Certificates on Simulators and Devices

Install and test certificates on various types of simulators and devices.

Note: The supported algorithm for the public-key cryptography used in the X.509 certificates is RSA.

Copy the generated .p12 certificate to the device on which you are installing.

See the User Guide for your device or simulator for instructions.

Installing X.509 Certificates on Windows Mobile Devices and Emulators

Install the *.p12 certificate on a Windows Mobile device or simulator and select it during authentication.

1. Launch the simulator or device.
2. Start the Windows synchronization software and cradle the device.
3. Use File Explorer to copy the *.p12 certificate to the simulator or device.
4. Navigate to and double-click the certificate.
5. Enter the password at the prompt and click **Done**.

An informational window indicates the certificate installed successfully.

Testing X.509 Certificates on Windows Mobile Devices and Emulators

Select an X.509 certificate to use for user authentication.

Prerequisites

1. Create a Hybrid App that prompts the user to specify a certificate as credentials.
2. Package and assign the Hybrid App to a Windows Mobile device user.

Task

1. In the Programs screen, open the Hybrid Web Container and select the Hybrid App to test.
2. Select the **Specify Certificate Credentials** menu item from the Certificate Picker.
3. Select the certificate and continue with the Hybrid App.

Installing X.509 Certificates on Android Devices and Emulators

Install the *.p12 certificate on an Android device or emulator.

Prerequisites

- Java SE Development Kit (JDK) must be installed.
- The Android SDK must be installed.

Task

1. Connect the Android device to your computer with the USB cable.
2. To install using Eclipse with the ADT plugin:

Note: USB debugging must be enabled.

- a) Open the Windows File Explorer view. From the menu bar, navigate to **Window > Show View > Other**.
 - b) In the Show View dialog, expand the Android folder and select **File Explorer**.
 - c) Expand **mnt > sdcard** and select the **sdcard** folder.
 - d) In the top right of the File Explorer view, click **Push a file onto the device**.
 - e) In the Put File on Device dialog, select the certificate and click **Open**.
3. To install using Windows Explorer:

Note: USB debugging must be disabled.

- a) Open **Windows Explorer**
 - b) Under your computer, click the Android device to expand the folder.
 - c) Click **Device Storage**, navigate to and select the certificate.
 - d) Import the certificate to the Device Storage folder.
4. To install using the Android Debug Bridge (adb):

Note: USB debugging must be enabled.

- a) Open the command line directory to the adb.exe file, for example, C:\Program Files\android-sdk-windows\tools, or C:\Program Files\android-sdk-windows\platform-tools
- b) Run the command: `adb push %PathToCert%\MyCert.p12 /sdcard/MyCert.p12`

Testing X.509 Certificates on Android Devices and Emulators

Select an X.509 certificate for user authentication.

Prerequisites

1. Create a Hybrid App that prompts the user to specify a certificate as credentials.
2. Package and assign the Hybrid App to an Android device user.

Task

1. On the Android device or emulator, in applications, click **Hybrid Web Container**.
2. Select the Hybrid App on which to test the installed certificate.
3. From the Certificate Picker, select the **Specify Certificate Credentials** menu item.
4. Select the certificate and click **OK**.
5. Enter the password and click **OK**.

Installing X.509 Certificates on BlackBerry Simulators and Devices

Install the .p12 certificate on the BlackBerry device or simulator and select it during authentication.

1. Install the certificate on a device:
 - a) Connect to the device with a USB cable.
 - b) Browse to the SD Card folder on the computer to which the device is connected.
 - c) Navigate to and select the certificate. Enter the password.
 - d) Import the certificate.

You can also use the BlackBerry Desktop Manager to install the certificate on the device, but you may need to perform a custom installation to access the Synchronize Certificates option.
2. Install the certificate on a simulator:
 - a) From the simulator, select **Simulate > Change SD Card**.
 - b) Add/or select the directory that contains the certificate.
 - c) Open the media application on the device, and select **Menu > Application > Files > MyFile > MediaCard**.
 - d) Navigate to and select the certificate. Enter the password.
 - e) Check the certificate and select **Menu > Import Certificate**. Click **Import Certificate** then enter the data vault password.

Testing X.509 Certificates on BlackBerry Devices and Simulators

Select an X.509 certificate to use for user authentication.

Prerequisites

1. Create a Hybrid App that prompts the user to specify a certificate as credentials.
2. Package and assign the Hybrid App to a BlackBerry device user.

Task

1. From the applications screen, open the Hybrid Web Container.
2. Select the Hybrid App for which to test the certificate.
3. From the Certificate Picker, select the **Specify Certificate Credentials** menu item.
4. Select the certificate and continue with the Hybrid App.

Installing X.509 Certificates on iOS Devices

Add an authentication screen to the Hybrid App from which you can authenticate with a generated X.509 certificate instead of a user name and password combination.

1. Copy the X.509 certificate used for authentication into a directory on the same host as SAP Mobile Server. For example, `c:\certs`.
2. Create a registry string value on SAP Mobile Server at `HKLM\Software\SAP\SAP Messaging Server\CertificateLocation` and populate it with the path. For example, `c:\certs`.
3. Name the X.509 certificate file as `domain_user.p12`, where *domain* is the SAP Mobile Server domain and *user* is the certificate user. The user must have read permission for .p12 file.
4. The system administrator must ensure the specified domain\user has “log on as batch job” permission on the Windows machine on which SAP Mobile Server runs:
 - a) Double-click **Control Panel > Administrative Tools > Local Security Policies**.
 - b) Expand **Local Policies** and select **User Rights Assignment**.
 - c) Right-click **Log on as a batch job** and select **Properties**.
 - d) Select **Add User or Group** and add the domain\user.
5. The account under which SAP Mobile Server runs must have adequate permissions to impersonate the domain\user, for example, the Administrator account for the domain.

Testing X.509 Certificates on iOS Devices and Simulators

Select an X.509 certificate for user authentication to test.

Prerequisites

1. Create a Hybrid App that prompts the user to specify a certificate as credentials.
2. Package and assign the Hybrid App to an iOS device user.

Task

1. During device application development, define and add a screen that has a Certificate Picker menu item.
2. Generate and deploy the application to the iPhone client.
3. Select **Certificate Picker** from the iPhone client.
4. Enter Windows credentials and certificate password in the dialog and click **Done**. Make sure the format is *domain|user*.
5. Submit the credentials to SAP Mobile Server.

Apple Push Notification Service

SAP Mobile Platform provides support for Apple Push Notification Service by pushing notifications to Hybrid Apps when the Hybrid App is offline.

With APNS, each device establishes encrypted IP connections to the service and receives notifications about availability of new items awaiting retrieval on SAP Mobile Server. This feature overcomes network issues with always-on connectivity and battery life consumption on 3G networks.

For more information on end-to-end iPhone application development and provisioning, see *Mobile Application Life Cycle*.

Note: APNS cannot be used on a simulator.

Examples of cases when notifications are sent include:

- The server identifies that a new message needs to be sent to the device. This could include:
 - A new Hybrid App is assigned to the device.
 - A DCN message is sent to SAP Mobile Server, targeting a particular user and the Hybrid App is not running.

If you want to use APNs for the Hybrid App, use the Apple Provisioning Portal to create your own .p12 certificate if you build your own Hybrid App using the source code included in `<SMP_HOME>\MobileSDK<version>\HybridApp\Containers\iOS`.

After creating the .p12 certificate, you must configure the APNs settings in SAP Control Center.

Provisioning iOS Devices

Use this procedure to provision your iOS device for APNs if you build your own application using the source code provided in `<SMP_HOME>\MobileSDK<version>\HybridApp\Containers\iOS\iOS_HWC_<version>.tar.gz`.

See the Apple developer documentation for Provisioning and Development. These procedures are documented in detail there. Applications developed for distribution must be digitally signed with a certificate issued by Apple. You must also provide a distribution provisioning profile that allows user devices to execute the application.

1. Register with Apple to download and use the iOS SDK. A free account allows you to download the SDK and develop with the simulator. To deploy Hybrid Apps to devices, you must create a certificate in your developer account and provision your device. See the *Apple Local and Push Notification Programming Guide* at <http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ProvisioningDevelopment/ProvisioningDevelopment.html> for details.
2. Use the iOS Provisioning Portal at <http://developer.apple.com/devcenter/ios/index.action> (you must log on or register as an Apple developer) to create the SSL certificate and Keys. Configure the certificate to enable for Apple Push Notification service.
3. On your Mac, launch the Keychain Access program. This is located in the `Utilities` folder.
 - a) In Keychain Access, select **Keychain Access > Certificate Assistant > Request a Certificate from Certificate Authority**.
 - b) In the Certificate Information window, enter the information. Use a unique common name.

Note: Make sure you use a different common name than a development certificate you already have. This creates a private key with the name you enter here.

A certificate request is created and saved in the Desktop folder by default.

4. In the Apple Provisioning Portal, continue with the App ID provisioning and browse to the certificate request file created in Keychain Access in the previous step, then click **Generate**.
5. Click **Continue**.
6. Click **Download Now**.

The certificate is downloaded onto your machine, the Keychain utility appears, and the certificate is imported into the "login" keychain.
7. Verify that the certificate is associated with a private key.
8. Create and install a Provisioning profile for the application.
9. In Xcode, open the `HWC.xcodeproj` project.

Note: Note the product name. This is used to configure the Hybrid Web Container in SAP Control Center and corresponds to the Application Name property in SAP Control Center.

Install and Configure the Hybrid Web Container On the Device

By default, the application name is HWC. This needs to be configured in the properties for the target. There is a 15-character limit for the product name.

10. Change `AppName` and `AppId` in the `Branding.strings` file for the necessary language resources.

This file is available under the **Resources** folder of the HWC Xcode project.

Note: The Bundle Identifier must correspond to the Bundle identifier specified in the App ID. Change it to something unique.

11. Copy the exported `<certificate_name>.p12` certificate to the machine where SAP Control Center is installed and follow the instructions in *Configuring Apple Push Settings for the Hybrid Web Container* and use the certificate you just created.

Note: Make sure you select only the certificate in the Keychain tool before exporting.

Configuring Apple Push Settings for the Hybrid Web Container

The certificate that was exported from the keychain corresponding to Apple Push settings must be configured with the correct application name in SAP Control Center.

Note: When configuring the Apple Push Notification Service, change the push gateway, push gateway port, feedback gateway, and feedback gateway port values only when configuring notifications in a development environment. To enable Apple push notifications, the firewall must allow outbound connections to Apple push notification servers on default ports 2195 and 2196. The default URL is for production and should be changed to `gateway.sandbox.push.apple.com` for development. After making these changes, you must restart your machine.

1. In the left navigation pane, select **Applications**.
2. In the right pane, select the **Applications** tab.
3. Select the **Application ID** for which you are configuring native notification and select **Properties**.
4. Select the **Push Configurations** tab and click **Add**.
5. Enter the **Application name**. Make sure this name matches the `AppId` entered in the `Branding.strings` file.

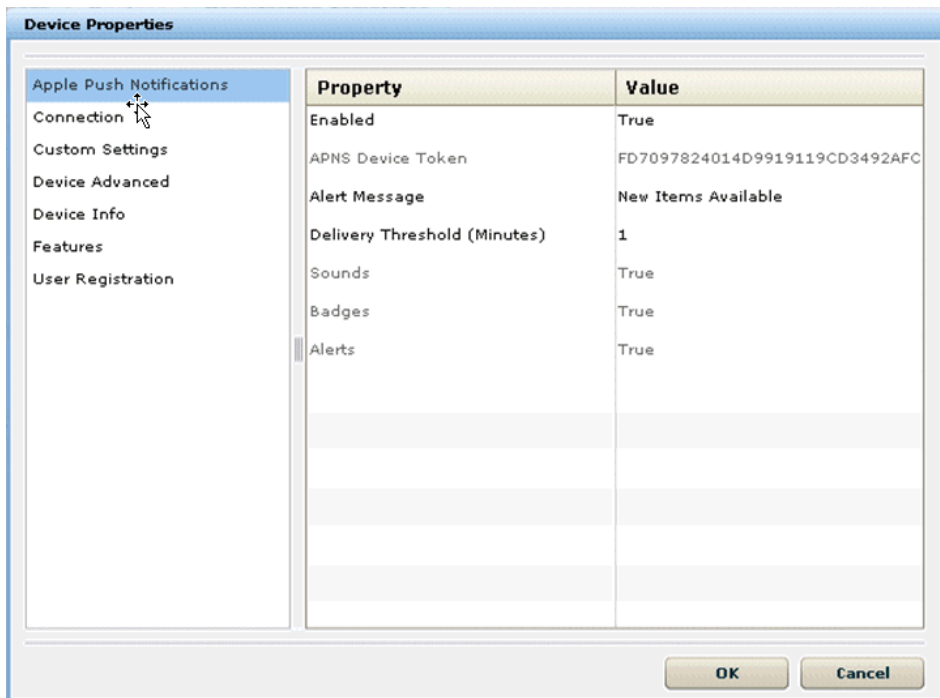
Enter:

Property	Description
Server	The push notification server.
Port	Push notification server port.
Feedback server	If a feedback service is enabled, the server to which APNS routes feedback information.
Feedback port	The feedback service port.

Install and Configure the Hybrid Web Container On the Device

Property	Description
Certificate (encoded)	The security certificate used for authentication.
Certificate password	The security certificate password.

6. Click **Browse** to use a security certificate file that already exists on the server.
 - a) Select the desired certificate from the list.
 - b) Enter and confirm the certificate password.
7. Click **OK**.
8. You can verify that the device is configured for APNS correctly by verifying that the device token has been passed from the application after the application runs once on the device.



Use the **Send a Notification** tool inside the Hybrid App Designer to send a test notification.

Apple Push Notification Properties

Apple push notification properties allow iOS users to install client software on their devices.

- **APNS Device Token** – the Apple push notification service token. An application must register with Apple push notification service for the iOS to receive remote notifications

sent by the application's provider. After the device is registered for push properly, this should contain a valid device token. See the iOS developer documentation.

- **Alert Message** – the message that appears on the client device when alerts are enabled. Default: `New items available`.
- **Delivery Threshold** – the frequency, in minutes, with which groupware notifications are sent to the device. Valid values: 0 – 65535. Default: 1.
- **Sounds** – indicates if a sound is made when a notification is received. The sound files must reside in the main bundle of the client application. Because custom alert sounds are played by the iOS system-sound facility, they must be in one of the supported audio data formats. See the iOS developer documentation.

Acceptable values: true and false.

Default: true

- **Badges** – the badge of the application icon.

Acceptable values: true and false

Default: true

- **Alerts** – the iOS standard alert. Acceptable values: true and false. Default: true.
- **Enabled** – indicates if push notification using APNs is enabled or not.

Acceptable values: true and false.

Default: true

Uninstall the Hybrid Web Container from the Device

Remove the Hybrid Web Container from the device.

Removing the Hybrid Web Container From the BlackBerry Device

Remove the Hybrid Web Container from the BlackBerry device.

You can remove the Hybrid Web Container using either the delete function on the device, or by using RIM Desktop Manager.

1. To remove the Hybrid Web Container using the delete function on the device;
 - a) On your BlackBerry device, navigate to **Options > Advanced Options > Applications**.
 - b) Scroll through the list of applications, highlight the Hybrid Web Container you want to remove and choose **Delete**.
 - c) When the confirmation dialog asks if you are sure, choose **Delete**. It may ask you to reset your device after removing the program

Install and Configure the Hybrid Web Container On the Device

When you delete the Hybrid Web Container from the device using this method, the data is removed by the `CodeModuleListener` method.

2. Use the RIM Desktop Manager to remove the Hybrid Web Container from the BlackBerry device.

See your BlackBerry documentation for how to remove applications using RIM Desktop Manager.

Note: If you delete the Hybrid Web Container using Desktop Manager or JavaLoader, the data is not deleted, as the `CodeModuleListener` is not used.

Hybrid Web Container Customization

The Hybrid Web Container project is accompanied by libraries and the source code necessary for you to build the Hybrid Web Container.

You can customize the Hybrid Web Container in a variety of ways. Whenever a customization requires a source code modification, there is a reference to “touch points” in the code. These references are annotated with

`<PLATFORM>_CUSTOMIZATION_POINT_<descriptor>` and a descriptor identifying the customization to which they belong.

For example, all code areas associated with changing the About screen are annotated with `<PLATFORM>_CUSTOMIZATION_POINT_BRAND`. The touch points are typically accompanied by brief comments in the code explaining the necessary changes. Only source code files contain these touch points. Many of the customizations are done in the CustomizationHelper file.

Note: After performing any customizations, you must rebuild the container. You can customize the Hybrid Web Container in a variety of ways. SAP recommends that you always test your changes before using the resulting application.

Adding a Custom Icon for the Hybrid App Package Using the Packaging Tool

Use the packaging tool to add a custom icon to the Hybrid App package.

1. Navigate to `SMP_HOME\MobileSDK22\HybridApp\PackagingTool` and double-click the `packagingtool.bat` file if you are using a 32-bit JDK, or `packagingtool64.bat` if you are using a 64-bit JDK.
2. Select the output directory for the Hybrid App package and click **OK**.
3. In Project Explorer, choose the project to which to add the custom icon.
4. Click the **Custom Icons** tab.
5. Click **Add** to add a custom icon.

When you add a custom icon, the `manifest.xml` file is updated when you generate the package.

6. Click **Save**.
7. Click **Generate** to generate the Hybrid App package.

Manually Adding a Custom Icon to the Manifest.xml File

The simplest way to add a custom icon for the Hybrid App package is by using the packaging tool, but you can also manually update the `manifest.xml` file to include a custom icon.

1. Open `manifest.xml` for editing.
2. Specify the custom icon image files in the `<Icons></Icons>` section of the file, for example:

The `<Icons>` element should be added under the root `<Manifest>` node.

```
<Icons>
  <Icon width="32" height="32" type="png" name="ambulance" />

  <Icon width="64" height="64" type="png" name="ambulance" />

  <Icon width="32" height="32" type="png" name="car" path="html/
car.png" processedpath="html/carp.png"/>
  <Icon width="32" height="32" type="png" name="train"
path="html/train.png" />
  <Icon width="48" height="48" type="gif" name="van" path="html/
image/van.gif" processedpath="html/image/vanp.gif"/>
</Icons>
```

The unique key of the icon element in the Icons collection is the combination of width, height, type, and name.

- `width` – (required) indicates the width of the image.
- `height` – (required) indicates the height of the image.
- `type` – (required) indicates the image type. The value should be same as image file suffix.
- `name` – (required) indicates the name of the icon. You can set it as an empty string.
- `path` – (optional) indicates the path of the normal icon image saved in the package. If the path attribute is missing or empty, the image for the normal icon is saved in the `html\icon` folder. The image file name is a combination of name, width, height and type. For example, the above ambulance icon file path is `html/icon/ambulance32x32.png`.
- `processedpath` – (optional) indicates the path of the processed icon image saved in the package. If the `processedpath` attribute is missing or empty, the image for the processed icon is saved in the `html\icon` folder. The image file name is a combination of name, width, height and type with the letter `p` appended. For example, the above ambulance processed icon file path is `html/icon/ambulance32x32p.png`.

Certain image formats, such as `.ico` files, might contain multiple resolutions in a single image file. Make sure that the `manifest.xml` file includes multiple entries for each of the different resolutions that all point to the same file through the `path` and `processedpath` attributes, as shown below:

```
<Icons>
<Icon width="32" height="32" type="ico" name="car" path="html/
car.ico" processedpath="html/carp.ico">
<Icon width="64" height="64" type="ico" name="car" path="html/
car.ico" processedpath="html/carp.ico">
<Icon width="128" height="128" type="ico" name="car" path="html/
car.ico" processedpath="html/carp.ico">
</Icons>
```

When there are multiple icon files declared, the Hybrid Web Container chooses the best matched icon based on the device's capability.

3. Add the icon file reference under the `<HtmlFiles>` element, for example:

```
<HtmlFile>html/icon/ambulance32x32.png</HtmlFile>
```

4. Save the `manifest.xml` file.

Changing the Hybrid App Package Icon

Modify the Hybrid App package application icon.

You cannot add new icons to the folder, but you can replace the existing icon images, using the same file name. The Hybrid App icons are named `ampicon<index>.png`, where `<index>` is a number between 30 and 116. The icon `ampicon48.png` is the default Hybrid App icon. This is also the icon that is shown on the menu item that shows all the Hybrid Apps.

Each Hybrid App icon has two associated image files that contain images for processed and unprocessed messages; `ampicon<index>.png` (unprocessed messages) and `ampicon<index>p.png` (processed messages). Processed means the message has been submitted to the server.

When you build the Hybrid Web Container with custom icons, the original icons still appear in SAP Control Center and in SAP Mobile WorkSpace. You must remember the original icon, so you can select it in SAP Mobile WorkSpace and in SAP Control Center.

1. Identify the image currently used by the Hybrid App package that you want to replace:
 - a) Log in to SAP Control Center.
 - b) In **Workflows**, select the Hybrid App package for which to replace the image.
 - c) Click the **General** tab.

The icon is shown in **Display icon**.

2. Go to the `...\HybridWebContainer\res\drawable\` folder and find and replace the `ampicon<index>.png` and `ampicon<index>p.png` image files with the new images.

Note: The new image files must use the same name as those you replaced, including the file extension, and they must have the same resolution as the original images.

If you do not want to overwrite the icon entirely, make a copy of it using another name and move it out of the folder. Extra files in the drawable folder may interfere with resource indexing.

3. Rebuild the Hybrid Web Container project.

Android Hybrid Web Container Customization

Customize the look and feel and default behavior of the Android Hybrid Web Container.

Before getting started:

- Install the Android Development Tools (ADT) plug-in for Eclipse. See <http://developer.android.com/sdk/installing/installing-adt.html>.

Note: If you are also developing for BlackBerry, it is recommended that you do not install the BlackBerry Java Plug-in for Eclipse and the ADT plug-in in the same Eclipse environment.

- Build the Hybrid Web Container project as described in *Building the Android Hybrid Web Container Using the Provided Source Code*. The `HybridWebContainer` directory contains directories such as `libs`, as well as `images` and other files.

Documentation for the application (`com.sybase.hwc`) and the library (`com.sybase.hybridApp`) are included in the `docs` directory of the `HybridWebContainer` project.

Android Customization Touch Points

All code areas associated with Hybrid Web Container customizations are annotated with `ANDROID_CUSTOMIZATION_POINT_<customization>` comment tags, or touch points.

Touch Point	Description
ANDROID_CUSTOMIZATION_POINT_COLORS	Use custom colors for the Hybrid Web Container.
ANDROID_CUSTOMIZATION_POINT_FONTS	Use custom fonts in the Hybrid Web Container.
ANDROID_CUSTOMIZATION_POINT_BRAND	Change application name, copyright, and developer information
ANDROID_CUSTOMIZATION_POINT_SPLASHSCREEN	Add a splash screen to the Hybrid Web Container.

Touch Point	Description
ANDROID_CUSTOMIZATION_POINT_DEFAULTSETTINGS	Set the defaults for the Settings screen.
ANDROID_CUSTOMIZATION_POINT_PRESETSETTINGS	Hard code settings for the Settings screen so they do not show up on the device. This prevents the user from changing the settings.
ANDROID_CUSTOMIZATION_POINT_PREPACKAGED_APP	Run the Hybrid Web Container as a single Hybrid App.
ANDROID_CUSTOMIZATION_POINT_PIN	Use for PIN screen customizations, or to remove the PIN screen.
ANDROID_CUSTOMIZATION_POINT_SORTING	Sort Hybrid App messages based on different criteria.
ANDROID_CUSTOMIZATION_POINT_FILTERING	Filter the list of Hybrid App messages so only messages meeting certain criteria are shown.
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSORT	Customize the criteria for how the Hybrid App list is sorted.
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH	Make the list of Hybrid App packages searchable.
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST	Customize the Hybrid App package list appearance.
ANDROID_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS	Create categorized views of the Hybrid App packages.
ANDROID_CUSTOMIZATION_POINT_HTTPHEADERS	Set HTTP headers for the Android Hybrid Web Container to include authentication tokens.
ANDROID_CUSTOMIZATION_POINT_PUSH_NOTIFICATION	Customize how the Hybrid Web Container handles the push notification.
ANDROID_CUSTOMIZATION_POINT_ANONYMOUS_USER	<p>Returns whether or not anonymous user support is being used. Change to YES to allow clients to register anonymously.</p> <hr/> <p>Note: For this to work, the HWC application connection template must be configured to use the anonymous security configuration. See <i>Application Connection Templates</i> in <i>SAP Control Center for SAP Mobile Platform</i>.</p>

Look and Feel Customization of the Android Hybrid Web Container

Customizations you can make to the look and feel include changing the splash screen, changing the Hybrid App icons and name, changing the Hybrid App package icons, changing labels and text, adding support for new languages, and so on.

Changing the Android Hybrid Web Container Icon

Modify the icon shown on the home screen by replacing the icon image files.

Changing this icon also changes the image used on the About screen, and the image that sometimes shows up in the title bar.

The icon image files are located in these directories:

- ...\\HybridWebContainer\\res\\drawable-hdpi
- ...\\HybridWebContainer\\res\\drawable-ldpi
- ...\\HybridWebContainer\\res\\drawable-mdpi

Go to each directory and replace the icon.png image file with another .png image of your choice.

Note: The new image files must use the same name as those you replaced, including the file extension, and they must have the same resolution as the original images.

Customizing the About Screen and Other Branding

Customize the About screen.

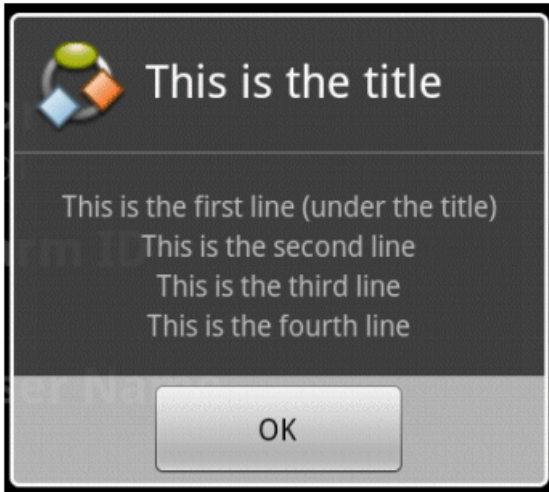
In some parts of the code, branding information is retrieved not from `strings.xml`, but from a constant in the `Brand` class. You cannot change these constants, but they are used only in a small number of places, and you can replace them where they are used. The `Brand` class is used mostly in the About screen, but there are a few other cases (all marked by the `ANDROID_CUSTOMIZATION_POINT_BRAND` comment tag).

1. Open the `CustomizationHelper.java` file, which is located in ...
\\HybridWebContainer\\src\\com\\sybase\\hwc.

This is where the strings in the About screen are set.

2. Locate the `customAbout` method.

Sample code is shown in this method. The default behavior is for the method to return false. The sample code produces the below dialog.



3. Uncomment the sample code, change the text to what you want to display, and change `return false;` to `return true;`.

Adding a Splash Screen

Add a splash screen to the Hybrid Web Container.

This procedure shows an example of a splash screen, which is the first screen that you see in the Hybrid Web Container. The related comment tag is

`ANDROID_CUSTOMIZATION_POINT_SPLASHSCREEN`.

1. Open the `SplashScreenActivity.java` file, which is located in the . . .
`\HybridWebContainer\src\com\sybase\hwc` folder.
2. Edit `SplashScreenActivity.java`.
 - a) You must call **finish()** on the splash screen as soon as you are finished displaying the screen.
 Currently this is done in the `onStart` method, so you must remove it from there.
 - b) Create an intent that launches the **EnterPasswordActivity** after **finish()** is called. You must do this even if you disable the PIN screen.
 It is important that **finish()** is called first. Currently this is done in the `onStop` method.

Changing Labels and Text

You can customize most of the text found in labels, dialogs, or error messages used by the Hybrid Web Container.

1. Open the `strings.xml` file, which is located in . . .`\HybridWebContainer\res\values` for editing.

This file contains the text for error messages, screen titles, screen labels, validation messages, and so on.

2. Make your changes and save the file.

Keep in mind that for any change you make, you must also make the same change for each language if you want your changes to translate across other languages. You must edit the `strings.xml` files located in the `values-<language_code>` folder for each language.

Adding a New Language

Add support to the Hybrid Web Container for a new language.

1. In the `...\HybridWebContainer\res` folder, create a new folder named `values-<xx>`, where `<xx>` is the ISO 639 code of the language, for example, `values-it`, for Italian.
2. Add a file called `strings.xml` to the new folder. Use the `strings.xml` file from the `values` folder as a template for the new `strings.xml` file.
3. Open the default `strings.xml` file, which is located in `...`
`\HybridWebContainer\res\values` and use it as a template for the new `strings.xml` file.

You need not include strings that do not require localization in the new `strings.xml` file. Strings that are missing from a localization are pulled from the default `strings.xml` file.

The new language is used automatically by a device that is set to that language.

Using Custom Colors

Use custom colors to change the look of Hybrid App messages and the Hybrid Web Container.

These examples modify the colors of the Hybrid App messages. You can also use custom colors for the Hybrid Web Container using similar steps. The related comment tag for customizing colors is `ANDROID_CUSTOMIZATION_POINT_COLORS`.

1. Open the `colors.xml` file, which is located in `...\HybridWebContainer\res\values`, for editing.
2. Find the `ANDROID_CUSTOMIZATION_POINT_COLORS` comment tag and add these tags inside the resources tag:

```
<color name="hybridapp_message_title_color">#F23431</color>
<color name="hybridapp_message_from_color">#FF1111</color>
<color name="hybridapp_message_date_color">#3234F1</color>
```

3. Open the `workflowmessages.xml` file, which is located in `...`
`\HybridWebContainer\res\layout`, for editing.
4. In the `msg_datetime` `TextView` tag, modify the `android:textColor` attribute to:


```
android:textColor="@color/hybridapp_message_date_color"
```

5. Make similar changes to the `msg_from` and the `msg_title` tags, using the color resource defined in step 2.

If you build the Hybrid Web Container without making any more changes, notice that the custom colors are used for `msg_datetime` and `msg_title`, but not for `msg_from`. This is because the color for `msg_from` is overridden by the Java code. To stop a custom attribute from being overridden:

- a) Select **Search > File** from the menu.
- b) For Containing text, enter `msg_from` and click **Search**.

The search result shows two files: `workflowmessages.xml` and `UiHybridAppMessagesScreen.java`.

- c) Open the `UiHybridAppMessagesScreen.java` file for editing.
- d) Search the file for "`msg_from`."

You will find this line: `TextView tf = (TextView) v.findViewById(R.id.msg_from);`

The `TextView` object `tf` represents `msg_from`.

- e) You are changing the color, so search for "`tf.setText`."

The search results return two occurrences because the color is set depending on whether the message has been read or not.

- f) Comment out both lines to ensure that `msg_from` is always the color you set in the `workflowmessages.xml` file. Save the file.

Using Custom Fonts

Customize fonts for Hybrid App messages and the Hybrid Web Container.

This example customizes the fonts for Hybrid App messages.

1. Create a new XML file named `attrs.xml` in the `...\HybridWebContainer\res\values\` folder.
2. Open the `attrs.xml` and add this code:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <declare-styleable name="com.sybase.hwc.CustomFontTextView" >
        <attr name="customFont" format="string"/>
    </declare-styleable>
</resources>
```

3. You cannot set the font attribute using the standard `TextView` control, so you must extend the `TextView` object by creating a new file named `CustomFontTextView.java`.
4. Add this code to the `CustomFontTextView.java` file:

```
package com.sybase.hwc;

import android.content.Context;
```

```
import android.widget.TextView;
import android.text.TextUtils;
import android.util.AttributeSet;
import android.content.res.TypedArray;
import android.graphics.Typeface;

public class CustomFontTextView extends TextView {

    public CustomFontTextView( Context oContext )
    {
        super( oContext );
    }

    public CustomFontTextView( Context oContext, AttributeSet
oAttrs )
    {
        super( oContext, oAttrs );
        setCustomFont( oContext, oAttrs,
R.styleable.com_sybase_hwc_CustomFontTextView,
R.styleable.com_sybase_hwc_CustomFontTextView_customFont );
    }

    private void setCustomFont( Context oContext, AttributeSet
oAttrs, int[] aiAttributeSet, int iFontId)
    {
        TypedArray taStyledAttributes =
oContext.obtainStyledAttributes( oAttrs, aiAttributeSet );
        String sCustomFont =
taStyledAttributes.getString( iFontId );
        if( !TextUtils.isEmpty( sCustomFont ) )
        {
            Typeface oTypeFace = null;

            try
            {
                oTypeFace = getFont( oContext, sCustomFont );
                setTypeface( oTypeFace );
            }
            catch (Exception e)
            {
                System.out.println( "Count not set font!" );
                // can't set the font
            }
        }
        else
        {
            System.out.println( "Custom font string was empty!" );
        }
    }

    private Typeface getFont( Context oContext, String
sCustomFont )
    {
        String sFullCustomFont = "fonts/" + sCustomFont;
        Typeface oTypeFace =
Typeface.createFromAsset( oContext.getAssets(),
```

```
sFullCustomFont );
    return oTypeFace;
}
}
```

5. Create a fonts folder in ... \HybridWebContainer\assets and add the TTF font file to this new folder.

For example, Windows fonts are usually in C:\Windows\Fonts\ if you want to use one of those.

6. Open the workflowmessages.xml file for editing and add this attribute to the RelativeLayout tag:

```
xmlns:custom="http://schemas.android.com/apk/res/com.sybase.hwc"
```

7. Find the TextView tag with the "ID msg_from" and change the tag from a TextView tag to a "com.sybase.hwc.CustomFontTextView" tag.

8. Add this attribute to the **com.sybase.hwc.CustomFontTextView** tag:

```
custom:customFont="<NAME_OF_YOUR_FONT_FILE.TTF>"
```

9. Repeat the above steps for tags with the "id msg_title" and "msg_datetime."

If you build the Hybrid Web Container without making any more changes, you see that "msg_title" and "msg_datetime" are shown with the custom font, but "msg_from" is not. This is because the font for "msg_from" is overridden in the Java code.

10. To prevent the font from being overridden:

- a) Select **Search > File** from the menu.
- b) For **Containing text**, enter msg_from and click **Search**.

The search result shows two files: workflowmessages.xml and UiHybridAppMessagesScreen.java.

- c) Open the UiHybridAppMessagesScreen.java file for editing.
- d) Search the file for "msg_from."

You will find this line: TextView tf = (TextView) v.findViewById(R.id.msg_from);

The TextView object tf represents msg_from.

- e) You are changing the font, so search for "tf.setTypeface."

The search results return two occurrences because the text is either bolded or not depending on whether the message has been read. Set bold, italic, or normal style for the text in the same way you specify the font.

- f) To ensure your custom font is used, make these modifications to the two occurrences of the method calls to **setTypeface**:

```
tf.setTypeface( tf.getTypeface(), Typeface.BOLD );

tf.setTypeface( tf.getTypeface(), Typeface.NORMAL );
```

Default Behavior Customization for the Android Hybrid Web Container

Default behavior that you can change includes removing a PIN screen, configuring default values for the Settings screen, sorting Hybrid App messages, and so on.

Removing Fields from the Settings Screen

You can hard-code settings for the Settings screen so they do not appear on the Settings screen on the device.

The comment tag associated with the fields on the Settings screen is
ANDROID_CUSTOMIZATION_POINT_DEFAULTSETTINGS.

1. Open the CustomizationHelper.java file, which is located in the ...
 \HybridWebContainer\src\com\sybase\hwc folder.
2. All of the settings screen customization functionality is grouped together under this comment in the file:

```
//-----  
-----  
    // Setting screen customization methods  
    //-----  
-----
```

3. To remove a field, set the associated property to false.

For example, if you want to remove the user name field, change:

```
public boolean isConnectionUserNameVisible()  
{  
    return true;  
}
```

to

```
public boolean isConnectionUserNameVisible()  
{  
    return false;  
}
```

Configuring Default Values for the Settings Screen

Set default values for the Settings screen.

The comment tag associated with customizations of the default settings is
ANDROID_CUSTOMIZATION_POINT_DEFAULTSETTINGS.

1. Open the CustomizationHelper.java file, which is located in the ...
 \HybridWebContainer\src\com\sybase\hwc folder.
2. Find the collection of methods named with the pattern
 getDefaultConnection<setting_name> or
 isDefaultConnect<setting_name>, where <setting_name> is the name of the
 setting.

3. Edit the methods to return the specific value you require.

The save button on the settings screen is enabled only when all of the fields requiring values are populated and a field is changed by the user, so if you change the return value for all of the methods to values that users do not have to modify on the device, you can run into a problem. To avoid this issue:

- a) Find the method in CustomizationHelper named `isSettingsSaveButtonAlwaysEnabled()`, which, by default, returns **false**.
- b) Change the method to return **true** so the save button is always enabled if all of the fields requiring values are populated.

Removing the PIN Screen

Remove the PIN screen (password screen) from the Hybrid Web Container.

The related comment tag is `ANDROID_CUSTOMIZATION_POINT_PIN`.

Note: Removing the PIN screen leaves data that is stored on the device less secure. You should remove the PIN screen only if you are not concerned about keeping your data secure.

1. Open the `CustomizationHelper.java` file, which is located in the `... \HybridWebContainer\src\com\sybase\hwc` folder.
2. Find the `enablePIN` method.
By default it returns **true** and shows the password screen.
3. Change the `enablePIN` method to return **false**.
The application does not show a password screen if it has been idle and is reactivated.
4. Test the application.

Using Multiple Hybrid Web Containers on the Same Android Device

Configure the Hybrid Web Container so that two or more Hybrid Web Containers co-exist on the same Android device.

1. Open the `AndroidManifest.xml` file, which is located under the HybridWebContainer project folder.
2. In the `manifest` tag, change the `"com.sybase.hwc"` package attribute to something else.
3. Search the file and change any references to `"com.sybase.hwc"` to the new package from step 2.

Note: Do not change any references to `com.sybase.hybridApp`, as these refer to the library jar files.

4. Save the file and choose **Yes** when asked if you want to change your launch configuration.
5. Change to the Eclipse Java perspective.

6. Right-click the package under `src` (it will be the old package name, `com.sybase.hwc`) and choose **Refactor > Rename**.
7. Set the name to be the package name you set in step 2.
8. Open the `CustomizationHelper.java` file, which is located in `... \HybridWebContainer\src\com\sybase\hwc`, and find the method named `getAppId()`:
By default `getAppId()` returns `Brand.OEM_HYBRIDAPP_APPID`. Change it to return a `String` that uniquely identifies your application.
9. You must now add an application with a matching App id in SAP Control Center, and if you want to use the automatic registration option, you must also add an Application Connection Template.
Now when you build the Hybrid Web Container, you can install it on a device that already has a Hybrid Web Container installed (but with a different package name). You should make other changes to your new Hybrid Web Container, such as `app_short_name` in the `strings.xml` file, or the icon .png image, to differentiate the Hybrid Web Containers on the device.

Sorting the List of Hybrid Apps

You can sort and filter the list of Hybrid Apps.

By default, the Hybrid Web Container displays Hybrid App packages in alphabetical order by package name. This procedure shows how to change the list so that it is case-sensitive. The related comment tag is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSORT`.

1. Open the `CustomizationHelper.java` file, which is located in the `... \HybridWebContainer\src\com\sybase\hwc` folder.
2. Find the `getHybridAppComparator()` method.
The comparator is used to order application (`HybridApp`) objects and is called by sort.
3. Modify the comparator to order the applications to meet your requirements.
4. Save the file.

Sorting Hybrid App Messages

Sort Hybrid App messages based on different criteria.

The comment tag associated with sorting Hybrid App messages is `ANDROID_CUSTOMIZATION_POINT_SORTING`.

1. Open the `CustomizationHelper.java` file, which is located in the `... \HybridWebContainer\src\com\sybase\hwc` folder.
2. Find the `getMessageComparator()` method.
The comparator is used to order `Message` objects and is called by sort.
3. Modify the comparator to order the messages to meet your requirements.

4. Save the file.

Filtering the Hybrid App Messages

Filter the list of Hybrid App messages so only messages that meet specified criteria are shown.

The comment tag associated with Hybrid App messages is

ANDROID_CUSTOMIZATION_POINT_FILTERING.

1. Open the CustomizationHelper.java file, which is located in the . . . \HybridWebContainer\src\com\sybase\hwc folder.
2. Find the getFilteredMessages() method.
The default behavior is to return all messages.
3. To return a subset of messages, you can modify getFilteredMessages() to return a list of messages based on your criteria.

For example, if you want only high priority messages to appear in the message list, you can change the code to the following:

```
// Display high priority messages only.
ArrayList<Message> filteredMessages =
MessageDb.getMessage( bCompleteList );
for( int iMessageIndex = 0; iMessageIndex <
filteredMessages.size(); iMessageIndex++ )
{
    if( filteredMessages.get(iMessageIndex).getMailPriority() !=
com.sybase.mo.AmpConsts.EMAIL_STATUS_IMPORTANCE_HIGH )
    {
        filteredMessages.remove(iMessageIndex);
        //we need to decrement the index so we don't skip an element now
        iMessageIndex--;
    }
}
return filteredMessages;
```

You must refresh the listview before the new messages are filtered. You can refresh the listview by switching to another view and then switching back.

Setting HTTP Headers

You can set HTTP headers for the Android Hybrid Web Container to include authentication tokens.

There are three sample methods showing how to do this in the Android Hybrid Web Container template source code, which include:

- setHttpHeaders() – use this method to set the authentication tokens. The tokens you set are used until setHttpHeaders is called again.
- setHybridAppTokenErrorListener() – use this method to call setHttpHeaders() to put the authentication tokens back in a good state, if, for example, they have expired.

- `setHttpErrorListener()` – use this method to handle HTTP errors.

The comment tag associated with setting HTTP headers is `ANDROID_CUSTOMIZATION_POINT_HTTPHEADERS`.

1. Open the `CustomizationHelper.java` file and make your changes.
2. Save the file.

Modifying the Hybrid App List Appearance

Change how the Hybrid Apps are shown on the device.

The comment tag associated with customizing the Hybrid App list appearance is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST`.

To show the list of applications, the Hybrid Web Container calls the `getHybridAppScreenClass()` method in `CustomizationHelper.java`. That method returns the class that displays the list. The default class is `UiHybridAppScreen`.

1. To make small changes to the list view, open the `UiHybridAppScreen.java` file, which is located in the `...\HybridWebContainer\src\com\sybase\hwc` folder, and make your changes.

Note: Optionally, you can create your own class that extends `UiHybridAppScreen`. If you do this, you must modify the `getHybridAppScreenClass()` method in the `CustomizationHelper` file to return the name of your new class.

2. Save the file.

Creating a Gallery View

Change the Hybrid App Package list view to a gallery view.

The comment tag associated with creating categorized views is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST`.

1. Add an XML layout called `hybridappgallery.xml` to the `HybridWebContainer` project.
2. Match your `hybridappgallery.xml` layout to:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Gallery xmlns:android="http://schemas.android.com/apk/res/
android"
        android:id="@+id/gallery"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
```



```
</LinearLayout>
```

3. Create a new activity for the HybridWebContainer.
 - a) Open the `AndroidManifest.xml` file.
 - b) Click the **Application** tab.
 - c) In the Application Nodes section (at the bottom left), click **Add**.
 - d) Choose **Activity** and click **OK**.
 - e) Select the new activity and change its name to `com.sybase.hwc.HybridAppGalleryActivity`.
 - f) Click **Name*** to generate the stub Java file.
 - g) Click **Finish**.
4. Enter this code into the `HybridAppGalleryActivity.java` file:

```
package com.sybase.hwc;

import java.util.ArrayList;
import java.util.Vector;
import java.util.Arrays;

import com.sybase.hybridApp.*;
import com.sybase.hybridApp.amp.Consts;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;

public class HybridAppGalleryActivity extends Activity {

    ImageAdapter m_adapter;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.hybridappgallery);

        Gallery oGallery = (Gallery) findViewById(R.id.gallery);
        m_adapter = new ImageAdapter(this);
        oGallery.setAdapter(m_adapter);

        oGallery.setOnItemClickListener(new OnItemClickListener ()
        {
            public void onItemClick(AdapterView parent, View v, int
```

```
position, long id)
    {
        startHybridApp(parent, v, position, id);
    }
    });
}

public void startHybridApp(AdapterView oParent, View v, int
iPos, long id )
{
    Intent oIntentHybridAppContainer = new Intent( this,
UiHybridAppContainer.class );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_START_MODE, Consts.START_MODE_HYBRIDAPP );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_ID, m_adapter.getItem( iPos ).getHybridAppId() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_PROGRESS_TEXT, m_adapter.getItem( iPos ).getDisplayName() );
    startActivityResult( oIntentHybridAppContainer,
Consts.INTENT_ID_HYBRIDAPP_CONTAINER );
}

@Override
public void onActivityResult( int iRequestCode, int
iResultCode, Intent relaunchData )
{
    super.onActivityResult( iRequestCode, iResultCode,
relaunchData );
    if ( iRequestCode == Consts.INTENT_ID_HYBRIDAPP_CONTAINER &&
iResultCode == Consts.RESULT_RELAUNCH )
    {
        Intent oIntentHybridAppContainer = new Intent( this,
UiHybridAppContainer.class );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_START_MODE, Consts.START_MODE_HYBRIDAPP );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_ID, relaunchData.getIntExtra( Consts.INTENT_PARAM_HYBRIDAPP_ID,
0 ));

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_PROGRESS_TEXT,
relaunchData.getStringExtra( Consts.INTENT_PARAM_HYBRIDAPP_PROGRE
SS_TEXT ));
        startActivityResult( oIntentHybridAppContainer,
Consts.INTENT_ID_HYBRIDAPP_CONTAINER );
    }
}

public class ImageAdapter extends BaseAdapter
{
```

```

//int mGalleryItemBackground;
private Context mContext;
private Vector<HybridApp> mHybridApps;

private ArrayList<Integer> mImageIds;

public ImageAdapter(Context c)
{
    mContext = c;
    mImageIds = new ArrayList<Integer>();

    //have to get a list of all installed HybridAppss
    mHybridApps = new
Vector<HybridApp>( Arrays.asList(HybridAppDb.getInvocableHybridAp
ps()) );
    for(int iHybridAppIndex = 0; iHybridAppIndex <
mHybridApps.size(); iHybridAppIndex++)
    {
        HybridAppDb oHybridApp = (HybridAppDb)
mHybridApps.get(iHybridAppIndex);
        int iconIndex = oHybridApp.getIconIndex();
        if(iconIndex >= 30 &&
            iconIndex <= 116)
        {
            //luckily the icon resources are consecutive
            int iResource = 0;
            if(iconIndex < 100)
            {
                iResource = 0x7f020022;
                iResource += (iconIndex - 30)*2;
            }
            else
            {
                iResource = 0x7f020000;
                iResource += (iconIndex - 100)*2;
            }
            mImageIds.add(new Integer(iResource));
        }
    }
}

public int getHybridAppId(int position)
{
    return
((HybridAppDb)mHybridApps.get(position)).getHybridAppId();
}

public String getDisplayName(int position)
{
    return
((HybridAppDb)mHybridApps.get(position)).getDisplayName();
}

public int getCount()
{
    return mImageIds.size();
}

```

```
    }

    public HybridAppDb getItem(int position)
    {
        return (HybridAppDb)mHybridApps.get(position);
    }

    public long getItemId(int position)
    {
        return position;
    }

    public View getView(int position, View convertView, ViewGroup
parent)
    {
        ImageView imageView = new ImageView(mContext);

        imageView.setImageResource(mImageIds.get(position).intValue());
        imageView.setLayoutParams(new
Gallery.LayoutParams(150,100));
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);

        return imageView;
    }
}
```

5. Save the file.
6. Open the `CustomizationHelper.java` file, which is located in the ...
\\HybridWebContainer\\src\\com\\sybase\\hwc folder and edit the
`getHybridAppScreenClass()` method, to change the class returned to your new
class.
That class must extend **Activity**.
7. Update the `manifest.xml` file to include the new activity you create.

Creating Categorized Views

Create categories so that Hybrid Apps and messages appear in lists under a category heading.

The comment tag associated with creating categorized views is
`ANDROID_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS`.

First, determine names for the categories. SAP recommends that you name the final category “Miscellaneous;” this adds all applications and messages that do not match a category to the Miscellaneous category. Also in this example, all applications that belong to a category must include the category name contained in their display name. For example, an application named “Financial Claim” belongs in the “Financial” category.

There are other ways to determine categories; if you know the names of the applications in advance, you can simply list all the application names that belong in each category.

1. Create a new XML layout called `category.xml` and paste the following code into the auto generated file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip">

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
        android:orientation="vertical"
        android:layout_width="0dip"
        android:layout_weight="1"
        android:layout_height="fill_parent">
        <TextView
            android:id="@+id/category"
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:singleLine="true"
            android:ellipsize="marquee"
            android:gravity="center_vertical"
        />
    </LinearLayout>

</LinearLayout>
```

2. Copy the `UiHybridAppScreen.java` file and rename it to your own class, for example, `CategorizedAppScreen.java`, and open it for editing.
3. Add the list of categories to the `UiHybridAppScreen` class, as a public static final member variable:

```
public static final String[] m_asHybridAppCategories =
{ "Financial", "Utilities", "Miscellaneous" };
```

4. Replace the `HybridAppAdapter` class with:

```
private class HybridAppAdapter extends ArrayAdapter<Object>
{
    private String[] m_asCategories;

    public HybridAppAdapter( Context context, int
textViewResourceId, List<Object> items, String[] categories ){
        super( context, textViewResourceId, items );

        m_asCategories = categories;

        for( int index = 0; index < m_asCategories.length; index
++ )
        {
            this.add( m_asCategories[index] );
        }
    }

    @Override
```

```

        public View getView(int position, View convertView,
        ViewGroup parent)
        {
            Object oObject = this.getItem(position);
            View v = null;
            if( oObject instanceof HybridApp )
            {
                HybridApp oHybridApp = ( HybridApp ) oObject;
                LayoutInflater vi =
                (LayoutInflater) getSystemService( Context.LAYOUT_INFLATER_SERVICE )
                ;
                v = vi.inflate(R.layout.workflows, null);

                if ( oHybridApp != null )
                {
                    ImageView ic = (ImageView)
                    v.findViewById( R.id.workflow_icon );

                    ic.setImageResource( UiIconIndexLookup.getNormalIconIdForIndex( o
                    HybridApp.getIconIndex() ));
                    TextView tt = (TextView)
                    v.findViewById(R.id.workflow_title);
                    if (tt != null) {
                        tt.setText( oHybridApp.getDisplayName());
                    }
                }
            }
            else
            { //This position is not a HybridApp, but a category
            heading
                String sString = ( String ) oObject;
                LayoutInflater vi = ( LayoutInflater )
                getSystemService( Context.LAYOUT_INFLATER_SERVICE );
                v = vi.inflate( R.layout.category, null );
                if( sString != null )
                {
                    TextView tt = (TextView)
                    v.findViewById( R.id.category );
                    if ( tt != null )
                    {
                        tt.setText( sString );
                    }
                }
            }
            return v;
        }

        public void remove( HybridApp oApp )
        {
            // The object to remove has a different pointer
            // so match it up with the one in the list
            for ( int i = 0; i < this.getCount(); i++ )
            {
                Object oObject = getItem( i );
                if( oObject instanceof HybridApp )
                {

```

```

        HybridApp oTemp = ( HybridApp ) oObject;

        if ( oApp.getModuleId() == oTemp.getModuleId()
            && oApp.getVersion() == oTemp.getVersion() )
        {
            super.remove( oTemp );
            return;
        }
    }

    }

    }

    public void sort()
    {
        // Sorts applications by name
        this.sort( new Comparator<Object>()
        {
            @Override
            public int compare( Object oObject1, Object
oObject2 )
            {
                if( oObject1 instanceof String && oObject2
instanceof String)
                {
                    String sString1 = ( String ) oObject1;
                    String sString2 = ( String ) oObject2;
                    for( int index = 0; index < m_asCategories.length;
index++ )
                    {
                        if( sString1.equals( m_asCategories[index] ) )
                        {
                            return -1;
                        }
                        if( sString2.equals( m_asCategories[index] ) )
                        {
                            return 1;
                        }
                    }
                }
                else if( oObject1 instanceof HybridApp && oObject2
instanceof HybridApp )
                {
                    HybridApp oHybridApp1 = ( HybridApp ) oObject1;
                    HybridApp oHybridApp2 = ( HybridApp ) oObject2;

                    int iCategoryIndex1 =
getCategoryIndex( oHybridApp1 );
                    int iCategoryIndex2 =
getCategoryIndex( oHybridApp2 );

                    if( iCategoryIndex1 == iCategoryIndex2 )
                    {

```

```

        return
oHybridApp1.getDisplayName().toLowerCase().compareTo( oHybridApp2
.getDisplayName().toLowerCase() );
    }
    else
    {
        return iCategoryIndex1 - iCategoryIndex2;
    }
}
else
{ //we have one String (category heading) and one
HybridApp
    HybridApp oHybridApp = null;
    String sString = null;
    int iSwitch = 1;
    if( oObject1 instanceof HybridApp)
    {
        oHybridApp = ( HybridApp ) oObject1;
        sString = ( String ) oObject2;
    }
    else
    {
        oHybridApp = ( HybridApp ) oObject2;
        sString = ( String ) oObject1;
        iSwitch = -1;
    }

    int iHybridAppCategoryIndex =
getCategoryIndex( oHybridApp );
    int iCategoryIndex = getCategoryIndex( sString );
    if( iCategoryIndex <= iHybridAppCategoryIndex )
    {
        return 1*iSwitch;
    }
    else
    {
        return -1*iSwitch;
    }
}

return 0;
}

private int getCategoryIndex( String sString )
{
for( int index = 0; index < m_asCategories.length;
index++ )
{
if( m_asCategories[index].equalsIgnoreCase( sString ) )
{
return index;
}
}
return m_asCategories.length - 1;
}

```



```

    }

    private int getCategoryIndex( HybridApp oHybridApp )
    {
        for( int index = 0; index < m_asCategories.length;
index++ )
        {
            if( oHybridApp.getDisplayName().toLowerCase().indexOf( m_asCategories[index].toLowerCase() ) >= 0 )
            {
                return index;
            }
        }
        return m_asCategories.length - 1;
    }
}
});
}
}

```

5. In the onResume method, make modifications to the following line (changes are shown in **bold**):

```

this.m_adapter = new HybridAppAdapter( this, R.layout.workflows,
new
ArrayList<Object>(Arrays.asList( HybridAppDb.getInvocableHybridApps() ) ), m_asHybridAppCategories );

```

6. Modify the onItemClick method as shown in the example code (changes are shown in **bold**):

```

public void onItemClick( ListView oParent, View v, int iPos,
long id )
{
    Object oObject = m_adapter.getItem( iPos );
    if( oObject instanceof HybridApp )
    {
        HybridApp oHybridApp = ( HybridApp ) oObject;
        Intent oIntentHybridAppContainer = new Intent( this,
UiHybridAppContainer.class );

        oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_START_MODE, Consts.START_MODE_HYBRIDAPP );

        oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_ID, ((HybridAppDb) oHybridApp).getHybridAppId() );

        oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_PROGRESS_TEXT, oHybridApp.getDisplayName() );
        startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_HYBRIDAPP_CONTAINER );
    }
}

```

7. Save the file.

8. Open the `UiHybridAppMessagesScreen.java` file for editing, and in the `onCreateContextMenu` method, make these modifications (changes are shown in **bold**):

```
public void onCreateContextMenu( ContextMenu oMenu, View v,
ContextMenuItem menuInfo)
{
    super.onCreateContextMenu( oMenu, v, menuInfo );

    AdapterContextMenuItem oInfo = (AdapterContextMenuItem)
menuInfo;
    Object oObject = m_adapter.getItem( oInfo.position );
    if( oObject instanceof Message )
    {
        Message oMsg = ( Message ) oObject;

        oMenu.setHeaderTitle( oMsg.getSubject() );
        oMenu.add( 0, CONTEXT_MENU_DELETE, 0,
R.string.Context_Menu_Delete );

        // Save the id for operations used in the context menu
        m_iContextMessageId = oMsg.getMessageId();
    }
}
```

9. In the `onContextItemSelected` method, make these modifications (changes are shown in **bold**):

```
public boolean onContextItemSelected( MenuItem oItem )
{
    if ( oItem.getItemId() == CONTEXT_MENU_DELETE )
    {
        AdapterContextMenuItem oInfo = (AdapterContextMenuItem)
oItem.getMenuItem();

        // The message might have been deleted while the context
menu was open.
        // Make sure the position is still present and matches
the id we expect
        if ( oInfo.position < m_adapter.getCount() )
        {
            Object oObject = m_adapter.getItem( oInfo.position );
            if( oObject instanceof Message )
            {
                Message oMsg = ( Message ) oObject;

                if ( oMsg.getMessageId() == m_iContextMessageId )
                {
                    // Remove message from database
                    MessageDb.delete( oMsg.getMessageId() );
                }
            }
        }
        return true;
    }
    return false;
}
```

```
}
```

10. Replace the MessageAdapter class:

```
private class MessageAdapter extends ArrayAdapter<Object>
{
    String[] m_asCategories;

    public MessageAdapter( Context context, int
textViewResourceId, ArrayList<Object> items, String[]
categories ){
        super( context, textViewResourceId, items );

        m_asCategories = categories;

        for( int index = 0; index < m_asCategories.length; index
++ )
        {
            this.add( m_asCategories[index] );
        }
    }

    @Override
    public View getView(int position, View convertView,
ViewGroup parent) {
        Object oObject = getItem( position );
        View v = null;
        if( oObject instanceof Message )
        {
            Message oMsg = (Message) oObject;
            LayoutInflater vi =
(LayoutInflater) getSystemService( Context.LAYOUT_INFLATER_SERVICE );
            v = vi.inflate(R.layout.workflowmessages, null);

            if ( oMsg != null )
            {
                //set the Hybrid App message priority icon
                ImageView imageForPriority = (ImageView)
v.findViewById( R.id.priority_icon );

                if ( oMsg.getMailPriority() ==
AmpConsts.EMAIL_STATUS_IMPORTANCE_HIGH )
                {
                    imageForPriority.setImageResource( R.drawable.readhi );
                    imageForPriority.setVisibility( View.VISIBLE );
                }
                else if ( oMsg.getMailPriority() ==
AmpConsts.EMAIL_STATUS_IMPORTANCE_LOW )
                {
                    imageForPriority.setImageResource( R.drawable.readlow );
                    imageForPriority.setVisibility( View.VISIBLE );
                }
            }
        }
    }
}
```

```

        }
        else
            imageForPriority.setVisibility( View.GONE );

        ImageView ic = (ImageView)
v.findViewById( R.id.msg_icon );
        if ( oMsg.isMsgProcessed() )

ic.setImageResource( UiIconIndexLookup.getProcessedIconIdForIndex
( oMsg.getIconIndex() ));
        else

ic.setImageResource( UiIconIndexLookup.getNormalIconIdForIndex( o
Msg.getIconIndex() ));

        TextView tf = (TextView)
v.findViewById(R.id.msg_from);
        TextView tt = (TextView)
v.findViewById(R.id.msg_title);
        TextView bt = (TextView)
v.findViewById(R.id.msg_datetime);
        if ( tf != null ) {
            tf.setText( oMsg.getMsgFrom() );
        }
        if (tt != null) {
            tt.setText( oMsg.getSubject());
        }
        if(bt != null){
            Calendar dtReceived =
Calendar.getInstance();

dtReceived.setTime( oMsg.getReceivedDate() );

            Calendar dtNow = Calendar.getInstance();

            if ( dtNow.get( Calendar.YEAR ) ==
dtReceived.get( Calendar.YEAR ) &&
                dtNow.get( Calendar.MONTH ) ==
dtReceived.get( Calendar.MONTH ) &&
                dtNow.get( Calendar.DAY_OF_MONTH ) ==
dtReceived.get( Calendar.DAY_OF_MONTH ) )
            {
                bt.setText( ( new
SimpleDateFormat( "hh:mm
a" ) ).format( oMsg.getReceivedDate() ) );
            }
            else {
                bt.setText( ( new SimpleDateFormat( "MM/
dd/yy" ) ).format( oMsg.getReceivedDate() ) );
            }
        }

        // Update appearance unread messages
        if ( tf != null && tt != null && bt != null )
        {
            if ( !oMsg.isMsgRead() )
            {

```

```

// Setup view for unread message
v.setBackgroundResource( R.drawable.unread_selector );

        tf.setTextColor( Color.WHITE );
        tf.setTypeface( null, Typeface.BOLD );
    }
    else
    {
        // Setup view for read message
        v.setBackgroundResource( 0 );

        TypedValue tv = new TypedValue();

getTheme().resolveAttribute( android.R.attr.textColorSecondary,
tv, true );

tf.setTextColor( getResources().getColor( tv.resourceId ) );
        tf.setTypeface( null, Typeface.NORMAL );
    }
}
}
else
{
    String sString = ( String ) oObject;
    LayoutInflater vi = ( LayoutInflater )
getSystemService( Context.LAYOUT_INFLATER_SERVICE );
    v = vi.inflate( R.layout.category, null );
    if( sString != null )
    {
        TextView tt = (TextView)
v.findViewById( R.id.category );
        if ( tt != null )
        {
            tt.setText( sString );
        }
    }
}
return v;
}

public void sort()
{
    // Sorts applications by name
    this.sort( new Comparator<Object>()
    {
        @Override
        public int compare( Object oObject1, Object
oObject2 )
        {
            if( oObject1 instanceof String && oObject2
instanceof String)
            {
                String sString1 = ( String ) oObject1;

```

```

        String sString2 = ( String ) oObject2;
        for( int index = 0; index <
m_asCategories.length; index++ )
        {

if( sString1.equals( m_asCategories[index] ) )
        {
            return -1;
        }

if( sString2.equals( m_asCategories[index] ) )
        {
            return 1;
        }
        }

        else if( oObject1 instanceof Message && oObject2
instanceof Message )
        {
            Message oMessage1 = ( Message ) oObject1;
            Message oMessage2 = ( Message ) oObject2;

            int iCategoryIndex1 =
getCategoryIndex( oMessage1 );
            int iCategoryIndex2 =
getCategoryIndex( oMessage2 );

            if( iCategoryIndex1 == iCategoryIndex2 )
            {
                return
oMessage1.getReceivedDate().compareTo( oMessage2.getReceivedDate(
) );
            }
            else
            {
                return iCategoryIndex1 - iCategoryIndex2;
            }
        }
        else
        { //we have one String (category heading) and one
HybridApp
            Message oMessage = null;
            String sString = null;
            int iSwitch = 1;
            if( oObject1 instanceof Message)
            {
                oMessage = ( Message ) oObject1;
                sString = ( String ) oObject2;
            }
            else
            {
                oMessage = ( Message ) oObject2;
                sString = ( String ) oObject1;
                iSwitch = -1;
            }
        }
    }

```

```

        int iMessageCategoryIdex =
getCategoryIdex( oMessage );
        int iCategoryIdex = getCategoryIndex( sString );
        if( iCategoryIdex <= iMessageCategoryIdex )
        {
            return 1*iSwitch;
        }
        else
        {
            return -1*iSwitch;
        }
    }

    return 0;
}

private int getCategoryIndex( String sString )
{
    for( int index = 0; index < m_asCategories.length;
index++ )
    {
        if( m_asCategories[index].equalsIgnoreCase( sString ) )
        {
            return index;
        }
    }
    return m_asCategories.length - 1;
}

private int getCategoryIndex( Message oMessage )
{
    MessageDb oMessageDb = (MessageDb) oMessage;
    if( oMessageDb != null )
    {
        HybridApp oHybridApp =
HybridAppDb.getHybridApp(oMessage.getModuleId(),
oMessage.getModuleVersion());
        String sModuleName =
oHybridApp.getDisplayName();
        if( sModuleName != null )
        {
            for( int index = 0; index <
m_asCategories.length; index++ )
            {
                if( sModuleName.toLowerCase().indexOf( m_asCategories[index].toLo
werCase() ) >= 0 )
                {
                    return index;
                }
            }
        }
    }
}

```

```

        return m_asCategories.length - 1;
    }
    });
}
}

```

11. In the **onResume** method, make these changes (changes are shown in **bold**):

```

try
{
    // ANDROID_CUSTOMIZATION_POINT_FILTERING
    ArrayList<Message> alMessages = MessageDb.getMessage();
    ArrayList<Object> alMessagesObjects = new
ArrayList( alMessages );
    this.m_adapter = new MessageAdapter( this,
R.layout.workflowmessages, alMessagesObjects,
UiHybridAppScreen.m_asHybridAppCategories );

    this.m_adapter.sort();
}

```

12. In the **onListItemClick** method, make these modifications (changes are shown in **bold**):

```

public void onListItemClick(ListView oParent, View v, int iPos,
long id )
{
    try
    {
        Object oObject = m_adapter.getItem( iPos );
        if( oObject instanceof Message )
        {
            Message oMsg = ( Message ) oObject;

            // Check if Hybrid App is available
            HybridApp oHybridApp =
HybridAppDb.getHybridApp( oMsg.getModuleId(),
oMsg.getModuleVersion());

            // CR668069 -Check if we can handle transform data -
1mb limit by sqllite database
            try
            {
                oMsg.getTransformData();
            }
            catch ( Exception ex )
            {
                MocaLog.getAmpHostLog().logMessage( "Failed to
read transform data", MocaLog.eMocaLogLevel.Normal );

                new AlertDialog.Builder( this )
                .setTitle( android.R.string.dialog_alert_title )
                .setMessage( R.string.IDS_MSG_ERR_MESSAGE_TOO_L
ARGE )

                .setIcon( android.R.drawable.ic_dialog_alert )
                .setPositiveButton( android.R.string.ok,
                    new DialogInterface.OnClickListener()

```



```

        {
            public void onClick( DialogInterface dialog, int
whichButton)
            {
                dialog.dismiss();
            }
        } )
        .show();

        return;
    }

    // Update read flag
    if ( !oMsg.isMsgRead() )
    {
        m_adapter.notifyDataSetChanged();
    }

    // Open Hybrid App
    Intent oIntentHybridAppContainer = new Intent( this,
UiHybridAppContainer.class );

    oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_START_MODE, Consts.START_MODE_MESSAGE );

    oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_MSG_ID, oMsg.getMessageId() );

    oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_MODULE_ID, oMsg.getModuleId() );

    oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_MODULE_VERSION, oMsg.getModuleVersion() );

    oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_HYBRIDAPP
_PROGRESS_TEXT, oMsg.getSubject() );
    startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_HYBRIDAPP_CONTAINER );
    }
    catch( Exception ex )
    {
        MocaLog.getAmpHostLog().logMessage( "Failed to open
message. Caught exception - " + ex.getMessage(),
MocaLog.eMocaLogLevel.Normal );
    }
}

```

13. Open the CustomizationHelper.java file, which is located in the ...
\HybridWebContainer\src\com\sybase\hwc folder and edit the
getHybridAppScreenClass() method, to change the class returned to your new
class, which you created in step 2.

That class must extend **Activity**.

14. Update the `manifest.xml` file to include the new activity you create.

Making the List of Hybrid App Packages Searchable

Make the list of Hybrid App packages searchable.

The comment tag associated with making the list of Hybrid App packages searchable is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH`.

1. Add an XML layout called `emptyview.xml`, and do not add anything to the resulting autogenerated XML file.
2. Open the `hybridapps_list.xml` file for editing and add the following tag above the `ListView` tag:

```
<EditText
    android:hint="@string/SEARCH_HINT"
    android:id="@+id/EditTextSearchHybridAppList"
    android:layout_width="match_parent"
    android:layout_height="47dp" />
```

3. Open `... \Values \Strings.xml` and, between the `<resource>` and `</resource>` tags, add:

```
<string name="SEARCH_HINT">search</string>
```

4. Copy the `UiHybridAppScreen.java` file to your own class name, for example, `SearchableAppScreen.java` and open it for editing.

- a) Add these import statements:

```
import android.widget.EditText;
import android.text.Editable;
import android.text.TextWatcher;
```

- b) Add the following code to the end of the `onCreate` method:

```
final EditText edittext = (EditText)
findViewById(R.id.EditTextSearchHybridAppList);
edittext.addTextChangedListener( new TextWatcher()
{
    public void afterTextChanged( Editable s)
    {
        String sSearchFor = s.toString();
        m_adapter.setSearch( sSearchFor );
        m_adapter.notifyDataSetChanged();
    }

    // stubs; have to implement the abstract methods
    public void beforeTextChanged( CharSequence s, int start, int
count, int after ) {}
    public void onTextChanged( CharSequence s, int start, int
before, int count) {}
});
```

- c) Add this member variable to the `HybridAppAdapter` class:

```
String m_sToSearchFor;
```

- d) Add this line of code to the end of the HybridAppAdapter constructor method:

```
m_sToSearchFor = "";
```

- e) Replace the code inside the getView method with:

```
public View getView(int position, View convertView, ViewGroup
parent)
{
    LayoutInflater vi =
    (LayoutInflater) getSystemService (Context.LAYOUT_INFLATER_SERVICE);
    View v = vi.inflate(R.layout.hybridapps, null);

    HybridApp oHybridApp = getItem( position );
    if( oHybridApp != null )
    {
        if( m_abDisplayThisApp == null || position >=
m_abDisplayThisApp.length || m_abDisplayThisApp[position])
        {
            ImageView ic = (ImageView)
v.findViewById( R.id.hybridApp_icon );

            ic.setImageResource( UiIconIndexLookup.getNormalIconIdForIndex
( oHybridApp.getIconIndex() ));
            TextView tt = (TextView)
v.findViewById(R.id.hybridApp_title);
            if (tt != null)
            {
                tt.setText( oHybridApp.getDisplayName());
            }
        }
        else
        {
            v = vi.inflate(R.layout.emptyview, null);
        }
    }
    return v;
}
```

- f) Add a search method to the HybridAppAdapter class:

```
public void search()
{
    m_abDisplayThisApp = new boolean[m_adapter.getCount()];

    for(int index = 0; index < m_adapter.getCount(); index++)
    {
        int iIndexOfResult =
m_adapter.getItem( index ).getDisplayName().indexOf( m_sToSearchFor );
        if( iIndexOfResult >= 0 )
        {
            m_abDisplayThisApp[index] = true;
        }
    }
}
```

```
    }  
}
```

- g) Add these methods to the HybridAppAdapter class:

```
public void notifyDataSetChanged()  
{  
    search();  
    super.notifyDataSetChanged();  
}  
public void setSearch( String sSearchFor )  
{  
    m_sToSearchFor = sSearchFor;  
}
```

- h) Add this member variable to the UiHybridAppScreen class:

```
private boolean[] m_abDisplayThisApp;
```

5. Open the CustomizationHelper.java file, which is located in the ... \HybridWebContainer\src\com\sybase\hwc folder and edit the getHybridAppScreenClass() method, to change the class returned to your new class.

That class must extend **Activity**.

6. Update the manifest.xml file to include the new activity you create.

Customizing the Push Notification Handler in the Android Hybrid Web Container

The comment tag associated with this customization is

ANDROID_CUSTOMIZATION_POINT_PUSH_NOTIFICATION.

By default, when a push notification is received by the Hybrid Web Container push listener, it returns the PushNotificationListener.NOTIFICATION_CONTINUE method, which allows the next push listener to handle the notification.

The comments in the onPushNotification method in the CustomizationHelper.java file include sample code that demonstrates how to open the default client-initiated Hybrid App if no Hybrid App is currently opened and also, optionally, calls a JavaScript method to initialize the Hybrid App once it is opened.

1. Open the CustomizationHelper.java file for editing.
2. Find the onPushNotification method and make your changes.

For example, if PushNotificationListener.CANCEL is returned, then the push listener manager will not invoke the next push notification listener.

3. Save the file.
4. Rebuild the project.

Testing Android Hybrid Web Containers

After making any customizations to the provided Hybrid Web Container source code, you should test the changes before using the application.

Note: The steps or interface may be different depending on which Android SDK version you are using.

This procedure assumes that you are using Eclipse.

1. Create a new Android virtual device.
 - a) a. Open the Android SDK Manager. If you are using Eclipse choose **Window > AVD Manager**.
 - b) b. Select **Tools > Manage AVDs**.
 - c) Click **New**.
 - d) Enter a name for the device and select **Android 2.2** as the target.
 - e) Click **Create AVD**.
2. Create a debug configuration for Android applications.
 - a) In Eclipse, in Workspace Navigator, right-click the Hybrid Web Container project and select **Debug as > Debug Configurations**.
 - b) Right-click **Android Application**.
 - c) Click **Target**.
 - d) In Deployment Target Selection Mode, select **Manual** and click **Debug**.
In the future you will only need to right-click the project and choose **Debug As > Android Application**.
 - e) In the Android Device Chooser, select **Launch a New Android Virtual Device (AVD)** and select the AVD you created in step 1.
 - f) Click **Start**.
 - g) Click **Launch**.

The Hybrid Web Container automatically launches when the emulator is fully started.

BlackBerry Hybrid Web Container Customization

Customize the look and feel and default behavior of the BlackBerry Hybrid Web Container.

Before getting started:

- Install the BlackBerry Java Plug-in for Eclipse. For information about the BlackBerry Java Plug-in for Eclipse, see <https://developer.blackberry.com/java/download/eclipse/>.

Note: If you are also developing for Android, SAP recommends that you do not install the BlackBerry Java Plug-in for Eclipse and the ADT plug-in in the same Eclipse environment.

- Build the Hybrid Web Container project as described in *Building the BlackBerry Hybrid Web Container Using the Provided Source Code*. The `HybridWebContainer` directory contains directories such as `libs`, as well as `images` and other files.

BlackBerry Customization Touch Points

All code areas associated with BlackBerry Hybrid Web Container customizations are annotated with `BLACKBERRY_CUSTOMIZATION_POINT_<customization>` comment tags, or touch points.

Touch Point	Description
<code>BLACKBERRY_CUSTOMIZATION_POINT_COLORS</code>	Use custom colors for the Hybrid Web Container.
<code>BLACKBERRY_CUSTOMIZATION_POINT_FONTS</code>	Use custom fonts in the Hybrid Web Container.
<code>BLACKBERRY_CUSTOMIZATION_POINT_BRAND</code>	Change application name, copyright, and developer information.
<code>BLACKBERRY_CUSTOMIZATION_POINT_SPLASHSCREEN</code>	Add a splash screen to the Hybrid Web Container.
<code>BLACKBERRY_CUSTOMIZATION_POINT_DEFAULTSETTINGS</code>	Set the defaults for the Settings screen.
<code>BLACKBERRY_CUSTOMIZATION_POINT_PRESETSETTINGS</code>	Hard-code Settings screen options so they do not show up on the device, preventing the user from changing the settings.
<code>BLACKBERRY_CUSTOMIZATION_POINT_PIN</code>	Use for PIN screen customizations, or to remove the PIN screen.
<code>BLACKBERRY_CUSTOMIZATION_POINT_SORTING</code>	Sort application messages based on a variety of criteria.
<code>BLACKBERRY_CUSTOMIZATION_POINT_FILTERING</code>	Filter the message list so only messages meeting certain criteria are shown.
<code>BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSORT</code>	Customize the criteria for sorting the Hybrid App list.

Touch Point	Description
BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH	Make the list of Hybrid App packages searchable.
BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPLIST	Customize the Hybrid App package list appearance.
BLACKBERRY_CUSTOMIZATION_POINT_CATEGORIZED-VIEWS	Create categorized views of the Hybrid App packages.
BLACKBERRY_CUSTOMIZATION_POINT_HTTPHEADERS	Set HTTP headers for the BlackBerry Hybrid Web Container to include authentication tokens.
BLACKBERRY_CUSTOMIZATION_POINT_MULTIHWC	Install more than one Hybrid Web Container on one device.
BLACKBERRY_CUSTOMIZATION_POINT_PREPACKAGE_APP	Run the Hybrid Web Container as a single Hybrid App.
BLACKBERRY_CUSTOMIZATION_POINT_PUSH_NOTIFICATION	Customize the way the Hybrid Web Container handles push notifications.
BLACKBERRY_CUSTOMIZATION_POINT_ANONYMOUS_USER	<p>Returns whether or not anonymous user login is supported. Change to YES to allow clients to register anonymously.</p> <p>Note: For this to work, the HWC application connection template must be configured to use the anonymous security configuration. See <i>Application Connection Templates</i> in <i>SAP Control Center for SAP Mobile Platform</i>.</p>

Look and Feel Customization of the BlackBerry Hybrid Web Container

Customizations you can make to the look and feel include changing the splash screen, changing the Hybrid App icons and name, changing the Hybrid App package icons, changing labels and text, adding support for new languages, and so on.

Changing the BlackBerry Hybrid Web Container Icon

Replace the BlackBerry Hybrid Web Container icon image file.

1. Navigate to the `HybridWebContainer\res\images` folder.
2. Replace the `icon.png` file with another `.png` image of your choosing.
The new image must use the same name, resolution, and extension as the original file.
3. Rebuild the project.

Rebranding the BlackBerry Hybrid Web Container

Modify the strings used in the `Brand` class for the BlackBerry Hybrid Web Container.

Almost all company and product specific strings used in the Hybrid Web Container are accessed through the `Brand` class.

1. Open the `HybridWebContainer.java` file for editing.
2. Make your modifications at the beginning of the main method (if you do not want to modify a default value, simply omit the line that changes it):

```
Brand.OEM_COMPANY_NAME = "Your Company Name";
Brand.OEM_FORMAL_COMPANY_NAME = "Your Formal
Company Name";
Brand.OEM_ROBIE_PRODUCT = "Your Name of the
Product";
Brand.OEM_COPYRIGHT = "Your Copyright String";
Brand.OEM_CORPDIR_OB_NAME = "HybridAppList Title";
```

3. Save the file.
4. To change the title, which uses the string `HybridWebContainer`, that appears on the Hybrid Web Container settings Screen:
 - a) In the Package Explorer view, right-click the BlackBerry application project and click **Properties**.
 - b) In the Properties for pane, click **BlackBerry Project**.
 - c) Click **Application Descriptor**.
 - d) Click the **Application** tab and change the Title.
 - e) In Package Explorer, right-click the `BlackBerry_App_Descriptor.xml` file and choose **Open With > Text Editor**.
 - f) Find the tag named `Packaging` and change the value of the `OutputFileName` to the name you used in step 4d.

Note: Remove any spaces or dashes, since these are illegal characters for output files.

- g) Open the `HybridWebContainer.java` file for editing.
- h) Add this line at the beginning of the `postEvent` method:

```
Brand.OEM_ENGINE_EXE_NAME = "HybridWebContainer";
```

Replace `HybridWebContainer` with the name you used in step 4d.

Note: If you modify `Brand.OEM_HYBRIDAPP_APPID`, you must have a matching Application ID in SAP Control Center.

Adding a Splash Screen

Add a splash screen to the BlackBerry Hybrid Web Container.

The splash screen is the first screen you see in the Hybrid Web Container. The related comment tag is `BLACKBERRY_CUSTOMIZATION_POINT_SPLASHSCREEN`.

1. Open the `CustomizationHelper.java` file for editing.
2. Find the `getSplashScreenClass` method.
3. Write your own splash screen class.
4. Have `getSplashScreenClass` return the class that you wrote for your splash screen, for example:

```
return SplashScreen.class;
```

Your class must extend `MainScreen`, call `pushScreen` on itself so that it appears, then `popScreen` on itself when it is finished.

```
package com.sybase.hwc;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;
/**
 * A simple splash screen.
 */
public class SplashScreen extends MainScreen
{
    private Timer timer = new Timer();

    public SplashScreen()
    {
        setTitle("Splash Screen");
        add( new LabelField( "Splash" ) );
        addKeyListener( new SplashScreenListener( this ) );

        // Dismiss the splash screen after 5 seconds.
        timer.schedule( new Countdown(), 5000 );

        UiApplication.getUiApplication().pushScreen( this );

        UiApplication.getUiApplication().requestForeground();
    }

    public void dismiss()
    {
        timer.cancel();
        UiApplication.getUiApplication().popScreen( this );
    }

    private class Countdown extends TimerTask
```

```

    {
        public
            void run()
        {

            UiApplication.getUiApplication().invokeLater( new
DismissThread() );
        }
    }

    private class DismissThread implements Runnable
    {
        public void run() {
            dismiss();
        }
    }

    protected boolean navigationClick( int status, int time )
    {
        dismiss();
        return true;
    }

    protected boolean navigationUnclick( int status, int time )
    {
        return false;
    }

    protected boolean navigationMovement( int dx, int dy, int
status, int time )
    {
        return false;
    }

    private static class SplashScreenListener implements
KeyListener
    {
        private SplashScreen screen;

        public SplashScreenListener( SplashScreen splash )
        {
            screen = splash;
        }

        public boolean keyChar( char key, int status,
int time )
        {
            // Quit the splash screen if ESC or MENU
            key pressed.
            switch ( key )
            {
                case
Characters.CONTROL_MENU:
                case Characters.ESCAPE:
                    screen.dismiss();
                    return true;
            }
        }
    }

```

```

        }
        return false;
    }

    public boolean keyDown( int keycode, int time )
    {
        return false;
    }

    public boolean keyRepeat( int keycode, int time )
    {
        return false;
    }

    public boolean keyStatus( int keycode, int time )
    {
        return false;
    }

    public boolean keyUp( int keycode, int time )
    {
        return false;
    }
}

```

5. Save the file and rebuild the project.

Changing Labels and Text in the BlackBerry Hybrid Web Container

You can customize most of the text found in labels, dialogs, and error messages used by the Hybrid Web Container.

All of the text that is not branding related and that appears as part of the Hybrid Web Container is contained in the `HybridWebContainer.rrc` file.

1. Open the `HybridWebContainer\res\com\sybase\hwc\HybridWebContainer_<language>.rrc` file, where *<language>* is the language code.

This file contains the text for error messages, screen titles, screen labels, validation messages, and so on.

2. Make your changes and save the file.

Keep in mind that you must also make the same changes for each language you want to translate into.

Adding a New Language

Add support for a new language to the BlackBerry Hybrid Web Container.

The default language for the Hybrid Web Container is English, and the English strings are located in `HybridWebContainer\res\com\sybase\hwc\HybridWebContainer.rrc`. The strings for different languages are located in the

resources folder. In general, strings of a language are located in a file named `HybridWebContainer_<language_code>.rrc`. For example, the German resource file is named `HybridWebContainer_de.rrc`.

1. Right-click the **resources** folder and choose **Create new file in resources**.
2. Name the file `HybridWebContainer_<language_code>.rrc`, where `<language_code>` is the language code of the language you want to add.
3. Double-click the new file to open it.
4. Set all the values to be in the new language.
5. Save the file and rebuild the project.

When the Hybrid Web Container is built with the resource file you added, it automatically uses the values it contains when the language on the BlackBerry device is set to the matching language.

Customizing the About Screen for the BlackBerry Hybrid Web Container

The related comment tag for customizing the About screen is

`BLACKBERRY_CUSTOMIZATION_POINT_BRAND`.

1. Open the `CustomizationHelper.java` file for editing.
2. Find the `customAbout` method, which contains commented-out code in the `customAbout` method, and Replace the text with whatever values you require.
3. Save the file and rebuild the project.

Using Custom Colors

The comment tag for customizing colors is

`BLACKBERRY_CUSTOMIZATION_POINT_COLORS`. There are a few places where you can change colors.

These steps provide an example of how to change the colors of different Hybrid Web Container components.

1. To change the highlight color of the selected Hybrid App in the Hybrid App list:
 - a) Open the `AppScreen.java` file for editing.
 - b) Make these modifications to the `drawListRow` method, found in the `ListFieldCallback` (the changes are in **bold**).

The changes in this example make the highlighted color orange and the unhighlighted color black (by default, they are blue and white, respectively).

```
public void drawListRow(ListField listField, Graphics graphics,
int index, int y, int width) {
    // y parameter is already offset to center text
    int iOffset = (listField.getRowHeight() -
    getFont().getHeight()) >> 1;

    HybridApp oApp = ( HybridApp ) m_oApps.elementAt( index );
```

```

if( listField.getSelectedIndex() == index )
{
    graphics.setColor( Color.ORANGE );
}
else
{
    graphics.setColor( Color.BLACK );
}

graphics.fillRect( 0, y - iOffset, width,
listField.getRowHeight() + y - iOffset );

final int iMargin = 2;

// Draw image
EncodedImage oImage
= EncodedImage.getEncodedImageResource( "ampicon" +
oApp.getIconIndex() + ".png");
Bitmap oBitmap = oImage.getBitmap();

graphics.drawBitmap( iMargin, y - iOffset +
( listField.getRowHeight() -oBitmap.getHeight() ) / 2,
oBitmap.getWidth(), oBitmap.getHeight(), oBitmap, 0, 0);

// Draw text
graphics.drawText( oApp.getDisplayName(), 2 * iMargin +
oBitmap.getWidth(), y );
}

```

2. To change the text color of the Hybrid App names in the Hybrid App list:

- a) In the AppScreen.java file, go to the drawListRow method, which is in the ListFieldCallback.

The color of the text is set by the code below. The first color (white, by default) is used when the field is in focus. The second color is used when the field is not in focus. This example coordinates these colors with the colors used in step 1. The changed code is in **bold**.

- b) Modify the code. For example:

```

// Draw text
if( listField.getSelectedIndex() == index )
{
    graphics.setColor( Color.BLACK );
}
else
{
    graphics.setColor( Color.WHITE );
}

```

```
graphics.drawText( oApp.getDisplayName(), 2 * iMargin +  
oBitmap.getWidth(), y );
```

3. To change the background color of the Hybrid Web Container:

- a) Add these import statements to the `AppScreen.java` file:

```
import net.rim.device.api.ui.decor.Background;  
import net.rim.device.api.ui.decor.BackgroundFactory;
```

- b) In the `AppScreen.java` file, go to the constructor method and add these lines after the `setTitle(...);` line:

```
Background bg =  
BackgroundFactory.createSolidBackground( Color.BLACK );  
this.getMainManager().setBackground( bg );
```

4. Change the background color and text color of label and edit fields.

To change the background and text colors of a label or edit field, you must override its paint method. This is done when you create the label. Below is an example of how to set the background color to black and the text color to white for a label. You can also do this, similarly, for edit fields.

- a) Open the `HWCSettingsScreen.java` file for editing.
- b) Make the following modifications (changes in bold). These changes make the background of the label black, and the text white. To use the same background color as the rest of the screen, you can leave out the first two lines in the paint method below:

```
// Connection Header  
m_oConnection = null;  
m_oConnection = new  
LabelField( m_res.getString( HybridWebContainerResource.IDS_CO  
NNECTION ),  
  
Field.FIELD_HCENTER )  
{  
    public void paint(Graphics g){  
        g.setColor( Color.BLACK );  
        g.fillRect( 0, 0, getWidth(), getHeight() );  
        g.setColor( Color.WHITE );  
        super.paint( g );  
    }  
};
```

5. Save the file and rebuild the project.

Using Custom Fonts

The customization tag for customizing fonts is

`BLACKBERRY_CUSTOMIZATION_POINT_FONTS.`

Use custom `.ttf` font files, which have a maximum size of 60KB, to install and use a custom font. You can set the default font for the Hybrid Web Container (described in step 1), or change the fonts for individual labels (described in step 2). Fonts for the list of Hybrid Apps are a special case (described in step 3).

1. Set the default font for the Hybrid Web Container:

- a) Add the .ttf font file to the resources folder of the HybridWebContainer project.
- b) Open the HWCSettingsScreen.java file and navigate to the constructor method, and add the following code to the beginning of that method.

The value FELIXTI.TTF in the second line is used. This is the name of the font file, and you should replace this value with the name of the font file you added in step 1a.

```
String sCustomFontName = "MyCustomFont";
int iFontLoadCode =
FontManager.getInstance().load( "FELIXTI.TTF",
sCustomFontName,

FontManager.APPLICATION_FONT);
if( iFontLoadCode == FontManager.SUCCESS)
{
    try
    {
        FontFamily oFamily =
FontFamily.forName( sCustomFontName );
        Font oFont = oFamily.getFont( Font.PLAIN, 23 );
        FontManager.getInstance().setApplicationFont( oFont );
    }
    catch (ClassNotFoundException e)
    {
        // the font was not found, so it cannot be set
    }
}
else
{
    // error loading font
}
```

The default font is applied to menu items, but not to the menu item that has focus. The following steps correct this.

- c) Open the AppScreen.java file and add:

```
import net.rim.device.api.ui.Font;
import net.rim.device.api.ui.FontFamily;
```

- d) Add this code to the end of the makeMenu method:

```
try
{
    FontFamily oFamily =
FontFamily.forName( "MyCustomFont" );
    Font oFont = oFamily.getFont( Font.PLAIN, 23 );
    menu.setFont( oFont );
}
catch ( ClassNotFoundException e )
{
    // problem finding the custom font
    String errorMsg = e.getMessage();
}
```

- e) Open the `LogScreen.java` file and add:

```
import net.rim.device.api.ui.FontFamily;
import net.rim.device.api.ui.component.Menu;
```

- f) Add the following method to both the `LogScreen` class (in `LogScreen.java`) and to the `HWCSettingsScreen` class (in `HWCSettingsScreen.java`):

```
protected void makeMenu( Menu menu, int context )
{
    try
    {
        FontFamily oFamily =
FontFamily.forName( "MyCustomFont" );
        Font oFont = oFamily.getFont( Font.PLAIN, 23 );
        menu.setFont( oFont );
    }
    catch ( ClassNotFoundException e )
    {
        String errormsg = e.getMessage();
        System.out.println( errormsg );
    }
    super.makeMenu( menu, context );
}
```

- g) In the `HWCSettingsScreen.java` file, add:

```
import net.rim.device.api.ui.FontFamily;
import net.rim.device.api.ui.Font;
import net.rim.device.api.ui.component.Menu;
```

2. Set the font for an individual label:

This example shows how to change the font for the screen title. Changing the font for any label is similar.

- a) Add the font file (a `.ttf` file) to the `resources` folder of the `HybridWebContainer` project.
- b) To the `AppScreen.java` file, add:

```
import net.rim.device.api.ui.Font;
import net.rim.device.api.ui.FontFamily;
```

- c) If you are going to set the font on more than one label, have a helper method. Add the following method to the `AppScreen` class:

```
public void setCustomFont( LabelField oLabel, String
sCustomFontName, int iSize )
{
    try
    {
        FontFamily oFamily =
FontFamily.forName( sCustomFontName );
        Font oFont = oFamily.getFont( Font.PLAIN, iSize );
        oLabel.setFont( oFont );
    }
    catch ( ClassNotFoundException e )
    {
        // the font was not found, so it cannot be set
        System.out.println( "Exception: font not found!" );
    }
}
```



```
    }
}
```

- d) In the `AppScreen` constructor, replace the `setTitle(...)` line with the code below.

"SHOWG.TTF" is the name of the font file. Replace this with the name of the font file you added in step 2a.

```
LabelField oTitleLabel = new LabelField( Consts.APP_TITLE,
DrawStyle.ELLIPSIS );
FontManager.getInstance().load( "SHOWG.TTF",
"CustomTitleFont", FontManager.APPLICATION_FONT);
setCustomFont( oTitleLabel, "CustomTitleFont", 23 );
this.setTitle( oTitleLabel );
```

3. To change the font for the names of the Hybrid Apps in the list of Hybrid Apps:

- Add the font file (a .ttf file) to the resources folder of the HybridWebContainer project.
- Open the `AppScreen.java` file for editing.
- Navigate to the `drawListRow` in `ListFieldCallback` and make the changes below, shown in bold.

"HARLOWSI.TTF" is the name of the font file. Replace this with the name of the font file you added in step 3a.

```
// Draw text
FontManager.getInstance().load( "HARLOWSI.TTF",
"CustomHybridAppFont", FontManager.APPLICATION_FONT);
try
{
    FontFamily oFamily =
FontFamily.forName( "CustomHybridAppFont" );
    Font oFont = oFamily.getFont( Font.PLAIN,
23 );
    graphics.setFont( oFont );
    graphics.drawText( oApp.getDisplayNa
me(), 2 * iMargin + oBitmap.getWidth(), y );
}
catch ( ClassNotFoundException e )
{
    //can't load the font
}
```

Default Behavior Customization for the BlackBerry Hybrid Web Container

Remove a PIN screen, configure default values for the Settings screen, customize the About screen, sort Hybrid App messages, and so on.

Removing Fields from the Settings Screen

Hard-code the Settings screen so options do not appear on the Settings screen on the BlackBerry device.

The comment tag associated with the fields on the Settings screen is `BLACKBERRY_CUSTOMIZATION_POINT_DEFAULTSETTINGS`.

1. Open the `CustomizationHelper.java` file, which is located in the ...
 `\HybridWebContainer\src\com\sybase\hwc` folder.
2. Search for the method named with the pattern `isConnection***Visible`, where
 `***` is the name of the connection setting field.

By default, each method returns true. To remove a field from the screen, change the appropriate method to return false.

3. Save the file.
4. Rebuild the project.

Configuring Default Values for the Settings Screen

All customization functionality for the Settings screen is grouped together in the `CustomizationHelper.java` file. The associated comment tag is `BLACKBERRY_CUSTOMIZATION_POINT_DEFAULTSETTINGS`.

1. Open the `CustomizationHelper.java` file for editing.
2. Search for the methods named with this pattern:
 - `getDefaultConnection***`
 - `isDefaultConnect***`where `***` is the name of the setting.
3. Edit the methods to return the value you specify.
4. Save the file.
5. Rebuild the project.

Using Multiple Hybrid Web Containers on the Same BlackBerry Device

Configure the Hybrid Web Container so that two or more Hybrid Web Containers can coexist on the same BlackBerry device.

Use a different COD module name, and make other changes to your new Hybrid Web Container, such as for the icon .png image, to differentiate between the Hybrid Web Containers on the device.

1. Right-click the **HybridWebContainer** project and click **Properties**.
2. In the General tab, change the title of the Hybrid Web Container.

3. In the Build tab, change the output file name to the name you used in step 2, but remove any spaces or dashes, since these are illegal characters for output files.
4. Open the `CustomizationHelper.java` file for editing.
5. Find the method named `getAppId()` and replace `Brand.OEM_HYBRIDAPP_APPID` with a unique name for your application.

The user must be registered in SAP Control Center with a device ID that matches the value you use in this step. You might need to create the device ID in SAP Control Center.

6. Open the `CustomizationHelper.java` file for editing.
7. Change the return value of `getApplicationIndicatorIconName` to the new indicator icon name, for example:

```
public class CustomizationHelper
{
    ....
    public final String getApplicationIndicatorIconName()
    { //return HWCMessagesScreen.INDICATOR_PNG; return
      "MetaData.png"; }
}
```

Sorting the List of Hybrid Apps

By default, Hybrid Apps are sorted alphabetically, ignoring case. The customization tag associated with sorting the list of Hybrid Apps is

`BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSORT.`

1. Open the `CustomizationHelper.java` file for editing.
2. Search for the method named `getHybridAppComparator()` and modify the code to suit your sorting requirements.

This example shows the Hybrid App being sorted by display name in reverse alphabetical order:

```
public Comparator getHybridAppComparator() {
    return new Comparator() {
        public int compare(Object oApp1, Object oApp2) {
            String sDisplayName1 = ((HybridApp) oApp1).getDisplayName()
                .toLowerCase();
            String sDisplayName2 = ((HybridApp) oApp2).getDisplayName()
                .toLowerCase();
            return (-1)*sDisplayName1.compareTo(sDisplayName2);
        }
    };
}
```

3. Save the file.
4. Rebuild the project.

Sorting Hybrid AppMessages

The default sorting behavior for Hybrid App messages is to list messages in the order they are received, newest first. The customization tag for sorting messages is `BLACKBERRY_CUSTOMIZATION_POINT_SORTING`.

1. Open the `CustomizationHelper.java` file for editing.
2. Search for the method named `getMessageComparator()` and modify the code to your sorting requirements.
3. Save the file.
4. Rebuild the project.

Filtering Hybrid App Messages

Filter the list of Hybrid App messages so only messages that meet specified criteria are shown. The default behavior is to return all messages. The comment tag associated with filtering Hybrid App messages is `BLACKBERRY_CUSTOMIZATION_POINT_FILTERING`.

1. Open the `CustomizationHelper.java` file for editing.
2. Find the method named `getFilteredMessages()` and modify it to meet your criteria.
`getFilteredMessages()` includes commented-out sample code that demonstrates how to filter out low-importance messages.
3. Save the file.
4. Rebuild the project.

Setting HTTP Headers

Set HTTP headers for the BlackBerry Hybrid Web Container to include authentication tokens.

These sample methods show how to do this in the BlackBerry Hybrid Web Container template source code.

- `setHttpHeaders()` – use this method to set the authentication tokens. The tokens you set are used until `setHttpHeaders` is called again.
- `setWorkflowTokenErrorListener()` – use this method to call `setHttpHeaders()` to put the authentication tokens back in a good state, if, for example, they have expired.
- `setHttpErrorListener()` – use this method to handle HTTP errors.

The comment tag associated with setting HTTP headers is `BLACKBERRY_CUSTOMIZATION_POINT_HTTPHEADERS`.

1. Open the `CustomizationHelper.java` file and make your changes.

2. Save the file.
3. Rebuild the project.

Modifying the Hybrid App List Appearance

The comment tag associated with customizing the Hybrid App list appearance is `BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPLIST`.

To show the list of Hybrid Apps, the Hybrid Web Container calls the `getHybridAppScreenClass()` method in the `CustomizationHelper.java` file. `getHybridAppScreenClass()` returns the default class `AppScreen` that displays the list.

1. To make small changes edit `AppScreen`, or create your own class that extends `UiHybridAppScreen`.
2. If you write your own class to extend `UiHybridAppScreen`, update `getHybridAppScreenClass` to return the name of your new class.
3. Save the file.
4. Rebuild the project.

Creating a Tree View

Modify the BlackBerry Hybrid Web Container so that Hybrid Apps appear in a tree view.

1. In the BlackBerry HybridWebContainer template project, in the `src` folder, right-click the **com.sybase.hwc.amp** package and choose **New > File**.
2. Enter `TreeViewAppScreen.java` for the file name, and click **Finish**.
3. Open the `TreeViewAppScreen.java` file for editing, and paste this code into the file.

```
/*
Copyright (c) SAP, Inc. 2012 All rights reserved.

In addition to the license terms set out in the SAP License
Agreement for
the SAP Mobile Platform ("Program"), the following additional or
different
rights and accompanying obligations and restrictions shall apply
to the source
code in this file ("Code"). SAP grants you a limited, non-
exclusive,
non-transferable, revocable license to use, reproduce, and modify
the Code
solely for purposes of (i) maintaining the Code as reference
material to better
understand the operation of the Program, and (ii) development and
testing of
applications created in connection with your licensed use of the
Program.
The Code may not be transferred, sold, assigned, sublicensed or
otherwise
```

```
    conveyed (whether by operation of law or otherwise) to another
    party without
    SAP's prior written consent. The following provisions shall apply
    to any
    modifications you make to the Code: (i) SAP will not provide any
    maintenance
    or support for modified Code or problems that result from use of
    modified Code;
    (ii) SAP expressly disclaims any warranties and conditions,
    express or
    implied, relating to modified Code or any problems that result
    from use of the
    modified Code; (iii) SAP SHALL NOT BE LIABLE FOR ANY LOSS OR
    DAMAGE RELATING
    TO MODIFICATIONS MADE TO THE CODE OR FOR ANY DAMAGES RESULTING
    FROM USE OF THE
    MODIFIED CODE, INCLUDING, WITHOUT LIMITATION, ANY INACCURACY OF
    DATA, LOSS OF
    PROFITS OR DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL
    DAMAGES, EVEN
    IF SAP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES; (iv)
    you agree
    to indemnify, hold harmless, and defend SAP from and against any
    claims or
    lawsuits, including attorney's fees, that arise from or are
    related to the
    modified Code or from use of the modified Code.
    */
package com.sybase.hwc.amp;

import com.sybase.mo.*;
import com.sybase.hybridApp.*;

import java.util Enumeration;

import net.rim.device.api.i18n.ResourceBundle;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.util.SimpleSortingVector;
import com.sybase.hwc.*;

// BLACKBERRY_CUSTOMIZATION_POINT_AUTOSTART
// BLACKBERRY_CUSTOMIZATION_POINT_COLORS
// BLACKBERRY_CUSTOMIZATION_POINT_FONTS
// BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPLIST

/**
 * This class displays a list of user invokable widgets currently
 * present on the
 * device.
 */
public class TreeViewAppScreen extends MainScreen {

    // Create a ResourceBundle object to contain the localized
```

```

resources.
    // Here is a little bit of MAGIC. How do you know what there is
    // a class HybridWebContainerResource? (hint: its not from the docs)
    // It is auto generated by the JDE. Convention is
    AppNameResource.BUNDLE_ID, AppNameResource.BUNDLE_NAME
    // http://www.codeproject.com/KB/mobile/EndToEndBBApp5.aspx
    public static final ResourceBundle RESOURCE =
        ResourceBundle.getBundle(
            HybridWebContainerResource.BUNDLE_ID,
            HybridWebContainerResource.BUNDLE_NAME );

    public TreeViewAppScreen() {
        super(Manager.VERTICAL_SCROLL |
Manager.NO_HORIZONTAL_SCROLLBAR);

setTitle( RESOURCE.getString( HybridWebContainerResource.IDS_HYBR
IDAPPS ) );

        // Sort apps by their display name
        m_oApps = new SimpleSortingVector();

m_oApps.setSortComparator( CustomizationHelper.getInstance().getH
ybridAppComparator() );

        m_oApps.setSort(false);

        // Populate and sort list
        BBHybridAppHelper.addAppStoreListener( m_oAppListener );
        // Add list field to screen
        m_oTreeField = new TreeField( m_oTreeFieldCallback,
TreeField.FOCUSABLE );

m_oTreeField.setEmptyString( BBHybridWebContainer.getMocaStringRe
source( MocaClientLibResource.LBL_NO_WIDGETS_FOUND ),
DrawStyle.HCENTER );
        // set the size of the indentation
        m_oTreeField.setIndentWidth( 30 );
        populateList();
        updateScreen();

        // add the tree field to the screen
        add( m_oTreeField );
    }

    /**
     * Handle clicking on an application
     */
    protected boolean navigationClick(int status, int time)
    {
        Field oField = getFieldWithFocus();
        // only handle if it was the tree field that was clicked
        if ( oField instanceof TreeField )
        {

```

```

        Object obj = m_oTreeField.getCookie( ( ( TreeField )
oField ).getCurrentNode() );
        // only handle the click if it was a hybrid app (not a tree
label)
        if( obj instanceof HybridApp )
        {
            // launch the clicked hybrid app
            HybridApp oApp = ( HybridApp ) obj;
            XmlHybridApp.startHybridApp( oApp.getModuleId(),
oApp.getVersion(), false );
            return true;
        }
    }

    return super.navigationClick(status, time);
}

/**
 * Override the default Screen.close method
 */
public void close()
{
    BBHybridAppHelper.removeAppStoreListener( m_oAppListener );

    UiApplication oApp = UiApplication.getUiApplication();
    oApp.popScreen(this);

    if ( oApp.getScreenCount() == 0 )
    {
        oApp.requestBackground();
    }
}

protected void makeMenu( Menu menu, int instance )
{
    menu.deleteAll();

    if ( CustomizationHelper.getInstance().enableSettings() )
    {
        menu.add(m_mniSettings);
    }

    menu.add(MenuItem.getPrefab(MenuItem.CLOSE));
}

/**
 * Fills in list of apps
 */
private void populateList()
{
    m_oApps.removeAllElements();

    for ( Enumeration e =
BBHybridAppHelper.getClientHybridApps().elements();
e.hasMoreElements(); )
    {

```



```

        HybridApp oHybridApp = ( HybridApp )e.nextElement();
        m_oApps.addElement( oHybridApp );
    }
    m_oApps.reSort();
}

/**
 * Updates the screen
 */
private void updateScreen()
{
    // have to do stuff to the UI on a separate thread
    UiApplication.getUiApplication().invokeLater(
        new Runnable()
        {
            public void run()
            {
                m_oTreeField.deleteAll();
                // if there're no hybrid apps then we do not even
want to add the tree labels
                // so that the empty string will be displayed
                if( m_oApps.size() > 0 )
                {
                    // In this example, there are 3 top level
categories of hybrid apps: Forms, Expense, and Miscellaneous.
                    // Forms has a sub-category of SpecialForms. In
practice you can have as many or as few categories
                    // and sub-categories as you like. Here the
category of a hybrid app is determined by whether
                    // keywords exist in the display name of that
hybrid app, but you could use anything else (for example
                    // you could determine the category of a hybrid
app by its icon).
                    int iMiscel = m_oTreeField.addChildNode( 0,
"Miscellaneous Hybrid Apps");
                    int iForms = m_oTreeField.addChildNode( 0, "Form
Hybrid Apps");
                    int iSpecialForms =
m_oTreeField.addChildNode( iForms, "Special Forms");
                    int iExpense = m_oTreeField.addChildNode( 0,
"Expense Hybrid Apps");
                    //have to iterate backwards through m_oApps
since addChildNode adds the new node
                    //to the first position (appears above the nodes
previously added).
                    for( int index = m_oApps.size()-1; index >= 0;
index-- )
                    {
                        HybridApp oHybridApp = (HybridApp)
m_oApps.elementAt( index );
                        int iParent = iMiscel;

                        if( oHybridApp.getDisplayName().indexOf("Expense") >= 0 )
                        {
                            iParent = iExpense;
                        }
                    }
                }
            }
        }
    );
}

```

```

        else
        if( oHybridApp.getDisplayName().indexOf("Form") >= 0 )
        {

        if( oHybridApp.getDisplayName().indexOf("Special") >= 0 )
            {
                iParent = iSpecialForms;
            }
            else
            {
                iParent = iForms;
            }
        }
        m_oTreeField.addChildNode( iParent,
m_oApps.elementAt( index ) );
        }
    }
}
} );
}

// Settings menu item
private MenuItem m_mniSettings =
    new
MenuItem( m_res.getString(HybridWebContainerResource.IDS_SETTINGS
),
        100001,
        10)
{
    public void run()
    {
        XmlHybridApp.startHybridAppSettings(false);
    }
};

// Listener for app changes
private HybridAppsListener m_oAppListener =
    new HybridAppsListener()
{
    public void onRefreshRequired()
    {
        populateList();
        updateScreen();
    }

    public void onHybridAppAdded(HybridApp oHybridApp)
    {
        populateList();
        updateScreen();
    }

    public void onHybridAppRemoved(HybridApp oHybridApp)
    {
        populateList();
        updateScreen();
    }
}

```

```

    }

    public void onHybridAppUpdated(HybridApp oHybridApp)
    {
        populateList();
        updateScreen();
    }
};

private SimpleSortingVector m_oApps;

private TreeField m_oTreeField;

private static ResourceBundle m_res =
ResourceBundle.getBundle(
    HybridWebContainerResource.BUNDLE_ID,
    HybridWebContainerResource.BUNDLE_NAME );

private TreeFieldCallback m_oTreeFieldCallback = new
TreeFieldCallback()
{
    public void drawTreeItem( TreeField oTree, Graphics
oGraphics, int iNode, int iY, int iWidth, int iIndent )
    {
        Object obj = oTree.getCookie( iNode );
        if( obj instanceof String )
        {
            oGraphics.setColor( Color.BLACK );
            oGraphics.drawText( (String)obj, iIndent, iY);
        }
        else if( obj instanceof HybridApp )
        {
            // y parameter is already offset to center text
            int iOffset = (oTree.getRowHeight() -
getFont().getHeight()) >> 1;

            // Draw a background color for the hybrid apps to
distinguish them from the tree labels.
            // However, if this node has focus we don't want to draw
the grey rectangle because it
            // will cover up the blue color indicating the node is
selected.
            if( iNode != m_oTreeField.getCurrentNode() )
            {
                oGraphics.setColor( Color.LIGHTGRAY );
                oGraphics.fillRect( iIndent, iY - iOffset, iWidth,
m_oTreeField.getRowHeight() );
            }

            HybridApp oApp = ( HybridApp ) obj;

            final int iMargin = 2;

            // Draw image
            EncodedImage oImage =

```

```

EncodedImage.getEncodedImageResource( "ampicon" +
oApp.getIconIndex() + ".png" );
    int iBitmapWidth = 0;

    if ( oImage != null )
    {
        CustomIcon oIcon = oApp.getDefaultCustomIcon();

        if ( oIcon != null )
        {
            EncodedImage oImageTmp =
oApp.getCustomIconImage( oIcon );

            if ( oImageTmp != null )
            {
                if ( oImageTmp.getHeight() != oImage.getHeight()
|| oImageTmp.getWidth() != oImage.getWidth() )
                {
                    MocaLog.getAmpHostLog().logMessage(
                        "Icon image size doesn't match the
built-in icon size, the layout result could be different.",
                        MocaLog.eMocaLogLevel.Normal );
                }

                oImage = oImageTmp;
            }

            Bitmap oBitmap = oImage.getBitmap();
            int iRowHeight = oTree.getRowHeight();

            int iSize = oImage.getHeight() > oImage.getWidth() ?
oImage.getHeight() : oImage.getWidth();

            if ( iSize >= iRowHeight )
            {
                oBitmap =
HWCMessagesListField.getScaledBitmapImage( oImage, iRowHeight -
iMargin, iSize );
            }

            oGraphics.drawBitmap(
                iMargin + iIndent,
                iY - iOffset + ( oTree.getRowHeight() -
oBitmap.getHeight() ) / 2,
                oBitmap.getWidth(), oBitmap.getHeight(),
oBitmap, 0, 0 );

            iBitmapWidth = oBitmap.getWidth();
        }
        else
        {
            MocaLog.getAmpHostLog().logMessage( "Can not find
application icon image of application " +
oApp.getDisplayName() + ".",

```

```

MocaLog.eMocaLogLevel.Normal );
    }

    // Draw text
    oGraphics.setColor( Color.BLACK );
    oGraphics.drawText( oApp.getDisplayName(), 2 * iMargin
+ iBitmapWidth + iIndent, iY );
    }
    }
};
}

```

This file is based on the `AppScreen.java` file. The main differences are in the constructor, `navigationClick`, `populateList`, and `updateScreen` functions. Also, the `TreeFieldCallback` class replaces the `ListFieldCallback` class from `AppScreen.java`.

4. Open the `CustomizationHelper.java` file for editing, find the `getHybridAppScreenClass` function, and replace the existing return statement with this line:


```
return com.sybase.hwc.amp.TreeViewAppScreen.class;
```
 5. Save the `CustomizationHelper.java` file.
 6. Rebuild the `HybridWebContainer` project.
- When you run the Hybrid Web Container, the Hybrid Apps are shown in a tree field.

Creating Categorized Views

Create a set of categories for the list of Hybrid Apps. The comment tag associated with this customization is `BLACKBERRY_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS`.

First, determine names for the categories. SAP recommends that you name the final category “Miscellaneous;” this adds all applications and messages that do not match a category to the Miscellaneous category. Also in this example, all applications that belong to a category must include the category name contained in their display name. For example, an application named “Financial Claim” belongs in the “Financial” category.

There are other ways to determine categories; if you know the names of the applications in advance, you can simply list all the application names that belong in each category.

1. Open the `AppScreen.java` file for editing and add:


```
import java.util.Vector;
import net.rim.device.api.util.Comparator;
```
2. Add a list of categories as a private final member variable to the `AppScreen` class, for example:


```
private final String[] m_asHybridAppCategories = { "Financial",
"Utilities", "Miscellaneous" };
```
3. In the constructor of `AppScreen`, replace the compare method in the `Comparator` with the following modified version:

```
// BLACKBERRY_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS
m_oApps.setSortComparator(new Comparator()
{
public int compare(Object oApp1, Object oApp2)
{
return 0;
}
});
```

Although you can sort with categories, doing so becomes complicated since you must check whether an element is a category name or a Hybrid App, and you typically want to sort only Hybrid Apps within a common category.

4. Replace the populateList method with this modified version:

```
private void populateList()
{
m_oApps.removeAllElements();
Vector vHybridApps = BBHybridAppHelper.getClientHybridApps();
for (int i = 0; i < m_asHybridAppCategories.length; i++)
{
m_oApps.addElement(m_asHybridAppCategories[i]);
for (int j = 0; j < vHybridApps.size(); j++)
{HybridApp ha = (HybridApp) vHybridApps.elementAt(j);
if (ha.getDisplayName().indexOf(m_asHybridAppCategories[i]) >= 0
|| i + 1 == m_asHybridAppCategories.length)
{m_oApps.addElement(ha);vHybridApps.removeElementAt(j--);
}
}
}
}
```

5. Replace the drawListRow method in ListFieldCallback with this modified version:

```
public void drawListRow(ListField listField, Graphics graphics,
int index, int y, int width) {
//
BLACKBERRY_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS

// y parameter is already offset to center
text
int iOffset = (listField.getRowHeight() -
getFont().getHeight()) >> 1;

//
BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPLIST
// HybridApp oApp = ( HybridApp )
m_oApps.elementAt( index );

// BLACKBERRY_CUSTOMIZATION_POINT_COLORS

final int iMargin = 2;

Object element =
m_oApps.elementAt( index );
```

```

        if( element instanceof HybridApp )
        {
            HybridApp oApp = ( HybridApp ) element;

            // Draw image

            EncodedImage oImage
= EncodedImage.getEncodedImageResource( "ampicon" +
oApp.getIconIndex() + ".png" );

            Bitmap oBitmap = oImage.getBitmap();

            graphics.drawBitmap( iMargin, y - iOffset +
( listField.getRowHeight() - oBitmap.getHeight() ) / 2,
oBitmap.getWidth(), oBitmap.getHeight(), oBitmap, 0, 0 );

            // Draw text
            graphics.drawText( oApp.getDisplayName(),
2 * iMargin + oBitmap.getWidth(), y );
        }
        else
        {
            // element must be a String
            String sCategoryName = (String) element;

            graphics.drawText( sCategoryName, iMargin, y );
        }
    }
}

```

- 6. Replace the navigationClick method in the AppScreen class with this modified version:**

```

protected boolean navigationClick(int status, int time)
{
    Field oField = getFieldWithFocus();
    if ( oField instanceof ListField )
    {
        int iIndex = ( ( ListField )
oField ).getSelectedIndex();

        if ( iIndex != -1 && m_oApps.size() > 0 )
        {
            Object oElement = m_oApps.elementAt( iIndex );

            if( oElement instanceof HybridApp )
            {
                HybridApp oApp = ( HybridApp )
oElement;

```

```
        XmlHybridApp.startHybridApp( oApp.getModuleId(),
oApp.getVersion(), false );
                                return true;
                                }
        }
    }
    return super.navigationClick(status, time);
}
```

7. Replace the `onHybridAppAdded` method in the `HybridAppsListener` with this modified version:

```
public void onHybridAppAdded(HybridApp oHybridApp) {
    onRefreshRequired();
}
```

8. Save the `AppScreen.java` file.
9. Open the `CustomizationHelper.java` file, which is located in the ...
\\HybridWebContainer\\src\\com\\sybase\\hwc folder and edit the
`getHybridAppScreenClass()` method, to change the class returned to your new
class.

Making the List of Hybrid Apps Searchable

Add a search field to the top of the Hybrid App list.

Whenever the contents of the search field change, only Hybrid Apps with matching names are listed. The comment tag associated with this customization is

`BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH`.

1. Open the `AppScreen.java` file for editing and add the following member variable to the `AppScreen` class:

```
private String m_sSearchFor;
```

2. Add the following code in the constructor of `AppScreen`, before the line that says `// Add list field to screen`:

```
//add in the search UI
LabelField searchLabel = new LabelField( "Search: " );
add( searchLabel );
EditField searchEdit = new EditField();
searchEdit.setChangeListener( new SearchFieldListener() );
add( searchEdit );
m_sSearchFor = "";
```

3. Add the following code to the end of the `populateList` method:

```
// BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH
for (int i = 0; i < m_oApps.size(); i++) {
    HybridApp ha = (HybridApp) m_oApps.elementAt(i);
    if( m_sSearchFor == null || m_sSearchFor.equals("") ||
    ha.getDisplayName().indexOf( m_sSearchFor ) >= 0 )
```



```

    {
        // there is no search, or this Hybrid App matches the
        search.
        // do nothing since the Hybrid App is already in the list
    }
    else
    {
        // there is a search and this Hybrid App does not match
        // remove this Hybrid App from the list
        m_oApps.removeElementAt(i);
        i--;
    }
}

```

4. Add the following class to the AppScreen class:

```

final class SearchFieldListener implements FieldChangeListener
{
    public void fieldChanged( Field field, int context)
    {
        if( field instanceof EditField )
        {
            EditField oEditField = (EditField) field;
            m_sSearchFor = oEditField.getText();
            populateList();
            updateScreen();
        }
    }
}

```

5. Open the CustomizationHelper.java file, which is located in the ... \HybridWebContainer\src\com\sybase\hwc folder and edit the getHybridAppScreenClass() method, to change the class returned to your new class.

That class must extend **Activity**.

6. Update the manifest.xml file to include the new activity you create.

Customizing the Push Notification Handler in the BlackBerry Hybrid Web Container

The comment tag associated with this customization is

BLACKBERRY_CUSTOMIZATION_POINT_PUSH_NOTIFICATION.

By default, when a push notification is received by the Hybrid Web Container push listener, it returns the PushNotificationListener.NOTIFICATION_CONTINUE method, which allows the next push listener to handle the notification.

The comments in the onPushNotification method in the CustomizationHelper.java file include sample code that demonstrates how to open the default client-initiated Hybrid App if no Hybrid App is currently opened and also, optionally, calls a JavaScript method to initialize the Hybrid App once it is opened.

1. Open the CustomizationHelper.java file for editing.

2. Find the `onPushNotification` method and make your changes.

For example, if `PushNotificationListener.CANCEL` is returned, the push listener manager does not invoke the next push notification listener.

3. Save the file.
4. Rebuild the project.

iOS Hybrid Web Container Customization

The Hybrid Web Container project that comes with SAP Mobile Platform is accompanied by libraries and the source code necessary for you to build the Hybrid Web Container.

Before getting started, unzip the directory that contains the Hybrid Web Container project as outlined in *Building the Hybrid Web Container Using the Provided iOS Source Code*. The Hybrid Web Container project unzips to a directory called HWC. Any references to a directory path in these procedures are relative to that top-level HWC directory.

The HWC directory contains directories such as `Classes`, `libs`, and `includes`, as well as images and other files. It also contains the `HWC.xcodeproj`, which is the Xcode project that builds the Hybrid Web Container, and is the project that is referenced in the customization procedures.

Whenever a customization requires a source code modification, there is a reference to “touch points” in the code. These references are annotated with `IOS_CUSTOMIZATION_POINT` and a descriptor identifying the customization to which they belong.

For example, all code areas associated with removing the PIN screen are annotated with `IOS_CUSTOMIZATION_POINT_PIN`. The touch points are typically accompanied by brief comments in the code explaining the necessary changes. Only source code files contain these touch points. The procedures describe where to modify plist files, strings files, and other non-source code files, but you must locate where to apply those changes.

The `CustomizationHelper.m` file included in the HWC project under the `Classes` group folder in the Xcode Project Navigator is used to encapsulate some of your customizations in a single place. In many cases, this file contains sample implementations of the customizations that you can follow.

Note: After performing any customizations, you must rebuild the project. SAP recommends that you always test your changes before using the resulting application.

iOS Customization Touch Points

All code areas associated with iOS Hybrid Web Container customizations are annotated with `IOS_CUSTOMIZATION_POINT_<customization>` comment tags, or touch points.

Touch Point	Description
IOS_CUSTOMIZATION_POINT_PRESET-SETTINGS	Provides alternative ways to get connection settings so they do not show up on the Settings screen. This prevents the user from changing them. There are variations on this customization.
IOS_CUSTOMIZATION_POINT_DEFAULT-SETTINGS	Set the defaults for the Settings screen.
IOS_CUSTOMIZATION_POINT_PREPACKAGED_APP	Include a prepackaged Hybrid App that launches automatically when the Hybrid Web Container starts.
IOS_CUSTOMIZATION_POINT_PIN	Use for PIN screen customizations, or to remove the PIN screen.
IOS_CUSTOMIZATION_POINT_SORTING	Sort Hybrid Apps or messages based on different criteria.
IOS_CUSTOMIZATION_POINT_FILTERING	Filter the list of Hybrid Apps or messages so only items meeting certain criteria are shown.
IOS_CUSTOMIZATION_POINT_HTTPHEADERS	Set HTTP headers for the iOS Hybrid Web Container to include authentication tokens.
IOS_CUSTOMIZATION_POINT_FONTS	Customize fonts in the Hybrid Web Container.
IOS_CUSTOMIZATION_POINT_SPLASH-SCREEN	Change the splash screen, or the length of time for which it is shown.
IOS_CUSTOMIZATION_POINT_COEXISTING	Run two or more independent Hybrid Web Containers on the same device.
IOS_CUSTOMIZATION_POINT_PUSH_NOTIFICATION	Customize how the Hybrid Web Container handles the push notification.

Touch Point	Description
IOS_CUSTOMIZATION_POINT_ANONYMOUS_USER	<p>Returns whether or not anonymous user support is being used. Change to YES to allow clients to register anonymously.</p> <hr/> <p>Note: For this to work, the HWC application connection template must be configured to use the anonymous security configuration. See <i>Application Connection Templates</i> in <i>SAP Control Center for SAP Mobile Platform</i>.</p>

Look and Feel Customization of the iOS Hybrid Web Container

Customizations you can make to the look and feel include changing the splash screen, changing the Hybrid App icons and name, changing the Hybrid App package icons, changing labels and text, and adding support for new languages.

Changing the Hybrid Web Container Application Icon

Modify the application icon shown on the home screen by replacing the image files in the HybridWebContainer directory.

1. Go to the HybridWebContainer directory, which is in the location where you unpacked the `iOS_HWC_<version.>tar.gz` file, and replace the `Icon-72.png` (iPad) and `Icon.png` (iPhone) image files with the new images.

Note: The new image files must use the same name as those you replaced, including the file extension, and they must have the same resolution as the original images.

2. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.
3. Click **Run**.

Changing the iOS Hybrid App Name

Edit a `plist` file to modify the application name.

1. In Xcode, use Project Navigator to find the file named `HWC-Info.plist`.
2. Open the file and change the **Bundle display name** to the new name.
3. Save the file.
4. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Customizing the Splash Screen

The splash screen is the first screen that appears when you start the Hybrid Web Container.

You can change either the image that is shown, or you can change the length of time that it appears. The splash screen is stored on a per-language basis in the `HybridWebContainer/<language>.lproj` directories. In each of these directories, there are three files that contain the splash screens for iPhone (`Default.png`) and iPad (`Default-Landscape.png` and `Default-Portrait.png`).

You must replace the file in each language subdirectory, or your new splash screen does not appear when the language setting is changed. The splash screen does not include any localizable strings, so you must provide the correct screen for each language, if you plan to support multiple languages.

1. Add a custom splash screen by replacing the appropriate files in the `HybridWebContainer/<language>.lproj` directory.

Note: The new image files must use the same name as those you replaced, including the file extension, and they must have the same resolution as the original images.

2. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Changing Labels and Text

You can customize most of the text found in labels, dialogs, or error messages used by the Hybrid Web Container.

Changes that you can make include:

- Buttons, labels, and error messages – these strings are in `Localizable.strings`, under the `Resources/<language>.lproj` group folders in the Xcode Project Navigator.
- Application branding – strings that identify the application, among other things. These strings are in `Branding.strings`, under the `Resources/<language>.lproj` group folder in the Xcode Project Navigator.
- About box – these strings are in `About.strings`, under the `Resources/Settings.bundle/en.lproj` folder. Expand the `Settings.bundle` under the `Resources` group folder in the Xcode Project Navigator. Here, you can change the company name or the version number that is shown in the About box in the Settings screen.

Keep in mind that for any change you make you must also make equivalent changes for each language if you want your changes to translate across other languages.

When modifying one of the `*.strings` files, you need only to change the second string value. For example, to change the `AppId` in `Branding.strings`, on this line: `AppId = HWC`, change only the "HWC."

Adding a New Language

Add support for new languages by dropping new `<language>.lproj` directories into the project.

By default, the **hybrid-container** is localized to several different languages. Localized resources are in `<language>.lproj` directories and group folders throughout the project, where *<language>* may be the full language name, or a two-digit country code. The simplest way to add a new language is to copy existing `lproj` directories for another language, translate the strings into the new language, and add the new `lproj` directories to the project.

This procedure uses English as a starting point.

1. Copy `HybridWebContainer/English.lproj` directory to `HybridWebContainer/<new_language>.lproj`.
This contains resources for the PIN screens and for the splash screen. You can localize or entirely redesign the PIN screen .
2. Add the newly created `HybridWebContainer/<new_language>.lproj` directory to the project, at the top level (not under any group folders).
3. In Finder, right-click `HybridWebContainer/Settings.bundle`, and select **Show Package Contents**.
The `Settings.bundle` directory opens.
4. Copy `en.lproj` to `<new_language>.lproj`.
5. Translate the strings in `Root.strings` (these are the strings that identify names of settings in the Settings screen) and `About.strings` (associated with the About box).
6. In Xcode, in the Project Navigator, find the newly created `<new_language>.lproj` directory under the `Resources/Settings.bundle`.
You do not need to explicitly add the new directory to the project, but you should verify it is there.
7. Copy `HybridWebContainer/strings/English.lproj` to `HybridWebContainer/strings/<new_language>.lproj`.
8. Translate the strings in `Branding.strings` and `Localizable.strings`.
9. In Project Navigator, add the newly created `HybridWebContainer/strings/<new_language>.lproj` directory to the project under the **Resources** group folder.

Default Behavior Customization for the iOS Hybrid Web Container

You can change the default behavior of the iOS Hybrid Web Container, including customizing or removing the PIN screen, changing the default behavior for the way the application launches, sorting and filtering the list of Hybrid App packages and messages, and so on.

Customizing PIN Screens on iOS

PIN screens prompt the user to either create or enter a password, respectively.

You can modify the PIN screens with custom text, or you can redesign them entirely. PIN screens include Create PIN and Enter PIN screens.

The PIN screens are stored in `.xib` files in the `HybridWebContainer/`
`<language>.lproj` directories:

- `CreatePasswordViewController.xib` – constructs the Create Password screen
- `EnterPasswordViewController.xib` – constructs the Enter Password screen

Creating New PIN Screens

You can completely redesign the PIN screens by modifying the `.xib` files.

1. Using Interface Builder, open the `CreatePasswordViewController.xib` and `EnterPasswordViewController.xib` files located in `HybridWebContainer/<language>.lproj`.

2. Make your modifications.

You can change the look and feel of buttons, change the text, or change the background. You likely do not want to remove buttons or fields, as doing so interferes with the functioning of the application.

Note: You must make the equivalent changes to each language for your new PIN screen to show correctly in other languages.

3. Rebuild the HWC `.xcodproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Changing Localizable Strings in the PIN Screen

To modify the text, you must change `strings` files.

Each of the PIN screen `.xib` files has a corresponding `strings` file with the same name with `.strings` appended to the end, for example, `HybridWebContainer/`
`<French>.lproj\CreatePasswordViewController.xib.strings`.

1. Open the `CreatePasswordViewController.xib.strings` and `EnterPasswordViewController.xib.strings` files, which are located in `HybridWebContainer/<language>.lproj`.
2. Modify and save the files.
3. Regenerate the `.xib` files:

- a) Open a Terminal window.
- b) Navigate to the HybridWebContainer directory, and execute:

```
ibtool --strings-file <language>.lproj/<strings file>  
<language>.lproj/<xib file> --write <language>.lproj/  
<xib file>
```

Note: *<language>* must be the same throughout, and the .strings file must correspond with the .xib file.

4. After rebuilding the .xib files, you can return to Xcode and view the new screens before rebuilding the Hybrid Web Container.

Removing the PIN Screen

You can disable and remove the PIN screen by making a minor code modification to the CustomizationHelper.m file.

Note: If you have previously used the Hybrid Web Container with a password on a particular device, you will no longer be able to access the encrypted database, or any data stored there, and the application may not work correctly if you remove the PIN screen. In this case, uninstall the Hybrid Web Container from the device before using the Hybrid Web Container without a PIN screen. For a simulator, click **Reset Content and Settings** first.

Note: Removing the PIN screen leaves data that is stored on the device less secure. You should remove the PIN screen only if you are not concerned about keeping your data secure.

All code areas associated with removing the PIN screen are annotated with `IOS_CUSTOMIZATION_POINT_PIN`.

1. In Xcode Project Navigator, open the CustomizationHelper.m file, which is located in HWC\Classes.
2. Find the usePIN function and change it to return NO instead of YES.
3. Save the file.
4. Rebuild the HWC.xcodeproj project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Using Default Connection Settings

You can customize the Hybrid Web Container so that it is pre-populated with connection settings, or to use default values if nothing is provided by the user, or to always use default values on startup.

These customizations involve changes to either Root.plist or CustomizationHelper.m.

All code areas associated with removing fields from the Settings screen are annotated with `IOS_CUSTOMIZATION_POINT_DEFAULTSETTINGS`. The customizations described

here assume the Settings screen is used as the interface for providing input from the user. For alternatives to using the default Settings screen, see *Removing Fields from the Settings Screen*.

1. In the Xcode project, in the Project Navigator, expand **Resources > Settings.bundle** and open the `Root.plist` file.
2. Expand the item for the settings you want to preset, and fill in the **DefaultValue** attribute. Most settings do not have default values, with the exception of the protocol and the registration method. Because these settings have a "Multi Value" **Type** in the `.plist` file (instead of Text Field), they always have a default value that is one of the accepted values listed in Values. You can open the **Values** tab to see the acceptable values for these settings. This example sets a default value of **443** for the server port, and sets the default protocol to **HTTPS**. The **Values** item is expanded and shows the acceptable values.

HWC > Resources > Settings.bundle > Root.plist > No Selection		
Key	Type	Value
Strings Filename	String	Root
▼ Preference Items	Array	(19 items)
▶ Item 0 (Group -	Diction...	(2 items)
▼ Item 1 (Text Field -	Diction...	(7 items)
Type	String	Text Field
Title	String	ServerNameSetting
Identifier	String	servername_preference
Default Value	String	
Text Field Is Secure	Boolean	NO
Keyboard Type	String	URL
Autocorrection Style	String	No Autocorrection
▼ Item 2 (Text Field -	Diction...	(6 items)
Type	String	Text Field
Title	String	ServerPortSetting
Identifier	String	serverport_preference
Default Value	String	443
Text Field Is Secure	Boolean	NO
Keyboard Type	String	Number Pad
▼ Item 3 (Text Field -	Diction...	(7 items)
Type	String	Text Field
Title	String	CompanyIDSetting
Identifier	String	companyid_preference
Default Value	String	
Text Field Is Secure	Boolean	NO
Keyboard Type	String	Alphabet
Autocorrection Style	String	No Autocorrection
▼ Item 4 (Multi Value -	Diction...	(6 items)
Type	String	Multi Value
Title	String	ProtocolSetting
Identifier	String	protocol_preference
▼ Values	Array	(2 items)
Item 0	String	HTTP
Item 1	String	HTTPS
▶ Titles	Array	(2 items)
Default Value	String	HTTPS
▶ Item 5 (Text Field - URL Prefix)	Diction...	(7 items)
▶ Item 6 (Group -	Diction...	(2 items)
▼ Item 7 (Multi Value -)	Diction...	(6 items)
Type	String	Multi Value
Title	String	
Identifier	String	registrationmethod_preference
▶ Values	Array	(3 items)
▶ Titles	Array	(3 items)
Default Value	String	0

Note: Pre-populating a value only sets its initial value on a one-time basis; it does not prevent the user from later changing it, nor does it prevent a server change from overwriting it. This approach also cannot be combined with the *Removing Fields from the Settings Screen* customization because it relies on using the settings bundle.

3. Save the file.
4. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Removing Fields from the Settings Screen

Customize the Settings screen to prevent certain settings from showing.

For example, you can preset the server port connection value, and then choose not to display that field in the Settings screen, bypassing the user's ability to change or see that field. If you want this behavior, but you want the user to also see the property value, see *Using Default Connection Settings*.

All code areas associated with removing fields from the Settings screen are annotated with `IOS_CUSTOMIZATION_POINT_PRESETSETTINGS`.

Keep in mind that connection settings sometimes have more than one “internal” name because different developers may reference the same settings using different names, particularly in local variable names. For example:

- server name = server id
- company id = farm id
- activation code = validation code

1. In the Xcode project, in the Project Navigator, expand **Resources > Settings.bundle** and open the `Root.plist` file.
2. Delete the dictionary item that corresponds to the setting to remove from the Settings screen.

For example, to remove the server port setting, delete the Text Field item with the title `ServerPortSetting`.

3. Save the file.
4. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.
5. For each property you remove from the Settings screen, you need to provide a way to configure that property.

See *Using Default Connection Settings*.

Using Multiple Hybrid Web Containers on the Same iOS Device

You can configure two or more Hybrid Web Containers to coexist on the same device.

All code areas associated with creating co-existing applications are annotated with `IOS_CUSTOMIZATION_POINT_COEXISTING`.

This customization allows two or more independent users to use the same device, but with their own private version of the application. In summary, you need to change the application ID, the bundle identifier, and possibly the URL scheme.

The application ID is used by the server to identify the application, and because of this, you cannot run two applications on the same device with the same application ID. By default, the Hybrid Web Container uses “HWC” for its application ID. Changing the application ID involves a minor change to `CustomizationHelper.m`. Additionally, you must signify to iOS that this is a distinct application. This requires a minor change to a `plist` file. Finally, if you are using Afaria to provision your application, you need to specify a unique URL scheme. This requires changes to the same `plist` file.

1. Change the application ID:
 - a) In Xcode Project Navigator, find and open the `CustomizationHelper.m` file, which is located in the `Classes` group folder,
 - b) Locate the customization point that accompanies the `getAppId` function, and change it so that it returns a unique name.
 - c) Save and close the file.
2. To differentiate this version of the Hybrid Web Container from another:
 - a) In Xcode Project Navigator, find and open the `HWC-Info.plist` file, which is located in the `Resources` group folder.
 - b) Change the bundle identifier value to something unique.
 - c) Save and close the file.
3. If you are using Afaria to provision your application, you must specify a unique URL scheme for your application.
 - a) In Xcode Project Navigator, find and open the `CustomizationHelper.m` file, which is located in the `Classes` group folder.
 - b) Locate the customization point that accompanies the `getAppUrlScheme` function, and change it so that it returns a unique name.
 - c) In Xcode Project Navigator, find and open the `HWC-Info.plist` file, which is located in the `Resources` group folder.
 - d) Expand the **URL types** item, and expand **Item 0**.
 - e) Change the URL identifier value to the value you specified for the Bundle identifier in the previous section.
 - f) Save and close the file.

Sorting and Filtering the List of Hybrid App Packages and Messages

By default, the Hybrid Web Container sorts the list of applications and messages in alphabetical order by package name.

There is no filtering by default.

You can sort and filter this list in any way you want. For example, you can filter Hybrid App packages from appearing according to whatever criteria you specify. You can filter out particular Hybrid App packages by name, or you can sort Hybrid App messages by subject. Hybrid App messages are server-initiated messages associated with a Hybrid App package, and appear in a separate TableView.

The sorting and filtering is done using arrays of NSSortDescriptor and NSPredicate objects, respectively. These arrays can be initialized at application startup, and can also be changed dynamically, giving you the ability to change the sorting or filtering criteria while the application is running.

The `HybridAppViewController.h` file defines the interface for a Hybrid App object. You can sort and filter the properties of this object.

1. Locate the `HybridAppViewController.h` file.

You do not need to modify this file, but you can view the properties of a Hybrid App object on which you might want to filter or sort.

This file is included in the `HWC/includes` directory, but it is not explicitly included in the Xcode project. To get the file to appear in the Xcode editor:

- a) In Xcode, open the `HWC.xcodeproj`.
- b) Open the `WidgetFolderController.h` file.
- c) Locate this line: `#import "HybridAppViewController.h"`, right-click inside the quotes, then select **Jump to Definition**.

Xcode opens the file.

2. Customizations involving filtering and sorting for both Hybrid App packages and messages can be made in the `CustomizationHelper.m` file.

- a) In Xcode Project Navigator, open the `CustomizationHelper.m` file, which is located in `HWC\Classes`.
- b) If you are customizing sorting behavior, locate the `IOS_CUSTOMIZATION_POINT_SORTING` customization tags that accompany these functions:

- `initializeHybridAppSortingDescriptors`
- `initializeMessageSortingDescriptors`
- `addHybridAppSortDescriptor`
- `addMessageSortDescriptor`
- `clearHybridAppSortDescriptors`

- `clearMessageSortDescriptors`

Customize the initialize functions to add sort descriptors at application startup. If you want to dynamically change the sorting criteria, you can call the add functions to add a sort descriptor to the end of the array, or you can call the clear functions to start over and then add to a clean array. Typically, you do not need to modify the add or clear functions.

The sort descriptor array is processed in order, so descriptors that appear toward the end of the array are only used when descriptors earlier in the array result in a tie between two elements. This allows you to sort on multiple property keys.

- c) If you are customizing filtering behavior, locate the `IOS_CUSTOMIZATION_POINT_FILTERING` customization tags that accompany these functions:

- `initializeHybridAppFilterPredicates`
- `initializeMessageFilterPredicates`
- `addHybridAppFilterPredicate`
- `addMessageFilterPredicate`
- `clearHybridAppFilterPredicates`
- `clearMessageFilterPredicates`

Customize the **initialize** functions to add filter predicates at application startup. If you want to dynamically change the filtering criteria, you can call the **add** functions to add a filter predicate to the end of the array, or you can call the **clear** functions to start over and then add to a clean array. Typically, you do not need to modify the **add** or **clear** functions.

3. Save the file.
4. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Changing to a New UI Control

You can change the way the list of Hybrid App packages and messages appear.

Hybrid Web Container uses `UITableView` objects to display the list of Hybrid App packages and messages. To change this behavior, you must completely rewrite some files. This procedure shows an example of a fully functional Cover Flow style view. You can use any UI library.

This customization involves rewriting one or two classes, depending on whether you want to customize the appearance of the application list or the messages list, or both. The application list view is in the `HybridAppsFolderView (.m and .h)` files, while the messages list view is in the `MessagesFolderView (.m and .h)` files. You can change the appearance of one or the other independently of one another.

This customization is not too difficult if you use the existing classes as an example. For the most part, you can (and probably should) reuse a lot of the code in the original classes. You will

likely see the biggest divergence when you replace the `UITableViewDelegate` and `UITableViewDataSource` functions, as well as the code that creates cells. This code is tailored to a `UITableView`, but you will probably find that the UI library you are trying to replace it with will have callback functions that accomplish similar things. In many cases, you will be able to copy and paste code from the original functions into your new class with very few modifications needed. The sample code provides very rudimentary views, but you can experiment with different views.

This example uses an open source UI library called `iCarousel`, available under the `zlib` License. The source is at <http://cocoacontrols.com/platforms/ios/controls/icarousel>. This example replaces the UI for the applications folder, while leaving the messages folder unchanged.

1. Download the `iCarousel` source code.
2. Copy the `iCarousel.h` and `iCarousel.m` files to the `HWC/Classes` directory, then add these files to the `Classes` group folder in the Project Navigator in Xcode.
Do not drag and drop the files into the `Classes` group folder, or they will not be incorporated into the project build phase. Instead, right-click the `Classes` group folder, and select **Add Files to HWC...**
3. If you are viewing this guide online from the Product Documentation Web site, click *iOS_HWC_Customization_Supplement.zip* to access the ZIP file containing new copies of `HybridAppsFolderView.h` and `HybridAppsFolderView.m`.
4. Drop the unzipped `HybridAppsFolderView` files into the `HybridWebContainer/Classes` directory, overwriting the original files.
You can customize the code to suit your needs, for example, you may want to design your own `UIViews`, or change from a cover flow to any of the other supported view types within `iCarousel`, or to a different UI library altogether.

Setting HTTP Headers

You can set HTTP headers for the iOS Hybrid Web Container to include authentication tokens.

There are three sample methods showing how to do this in the iOS Hybrid Web Container template source code, which include:

- `setHttpHeaders` – use this method to set the authentication tokens. The tokens you set are used from then on until `setHttpHeaders` is called again.
- `onHybridAppTokenError` – use this method to call `setHttpHeaders` to put the authentication tokens back in a good state, if, for example, they have expired.
- `onHTTPError` – use this method to handle HTTP errors.

All code areas associated with HTTP header customization are annotated with `IOS_CUSTOMIZATION_POINT_HTTPHEADERS`.

1. Open the `CustomizationHelper.m` file, which is located in `HybridWebContainer\Classes`.
2. Locate the `setHttpHeaders` method, and uncomment its contents.

The stub code that is provided shows an example of how to add headers and cookies. You simply need to replace the header and cookie assignments with your own. The `setHttpHeaders` function is already called in the `startEngine` function just before the client engine starts, so you need to provide the implementation of `setHttpHeaders`.

3. `CustomizationHelper.m` also includes stub implementations of `onHybridAppTokenError` and `onHTTPError` that you can implement.

The `onHybridAppTokenError` method is called when Hybrid App token authentication failure occurs, so it is a good idea to use this callback as an opportunity to refresh the HTTP headers again. A common way to do this is to maintain member variables that contain the values for the headers you want to set. Implement the `setHttpHeaders` function to use the values in those member variables when it sets the headers, then, in `onHybridAppTokenError`, you can update the member variables with the new header values, and then call `setHttpHeaders` again, for example:

```
[[CustomizationHelper getInstance] setHttpHeaders];
```

4. If you have custom code to run when an HTTP error occurs, add it to the `onHTTPError` function.

This method is called any time there is an HTTP error. You can use this to inform the user of errors, or log errors, or perform other custom steps in response to particular error codes.

Customizing the Push Notification Handler in the iOS Hybrid Web Container

Customize the way the Hybrid Web Container handles push notifications.

By default, when a push notification is received by the Hybrid Web Container push listener, the `kNotificationContinue` method is returned, which allows the next push listener to handle the notification. The comments in the `onPushNotification` method in the `HWCAppDelegate.m` file includes some sample code that demonstrates how to open the default client-initiated Hybrid App if no Hybrid App is currently opened.

The comment tag associated with this customization is `IOS_CUSTOMIZATION_POINT_PUSH_NOTIFICATION`.

1. Open the `HWCAppDelegate.m` file for editing.
2. Find the `onPushNotification` method and make your changes.

For example, if `kNotificationCancel` is returned, the push listener manager does not invoke the next push notification listener.

3. Save the file.
4. Rebuild the `HWC.xcodeproj` project.

- a) From the Xcode menu, select **Product > Clean**.
- b) Select **Product > Build**.

Hiding the Listview on iPad

Hide the listview on the iPad when in landscape orientation so the Hybrid App opens in the full screen.

When the Hybrid Web Container runs on iPad, it uses a `UISplitViewController` to display its main views. The list of Hybrid Apps and messages occupies the left-hand view (the master view), while the Hybrid App contents occupy the right-hand view (the details view). By default, the master view hides away while the device is in the portrait orientation, and can be accessed using a button on the navigation bar. The master view is presented side-by-side with the detail view while the device is in the landscape orientation. To hide the listview when using landscape orientation so the Hybrid App opens in full screen, use the customization tag `IOS_CUSTOMIZATION_POINT_IPAD_LIST_VIEW`.

Note: This customization is not supported on iOS 4.3. On iOS 5.1 and later, this customization disables the ability to present the master view with a swipe gesture, which is enabled by default.

1. In Xcode Project Navigator, find and open the `CustomizationHelper.m` file, which is located in the `Classes` group folder.
2. Locate the `shouldHideIpadListView` function and change it so it returns `YES`.
3. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Windows Mobile Hybrid Web Container Customization

Customize the look and feel and default behavior of the Windows Mobile Hybrid Web Container.

Before getting started, build the Hybrid Web Container project in Visual Studio, as described in *Building the Windows Mobile Hybrid Web Container Using the Provided Source Code*. In Solution Explorer, the `HybridWebContainer` directory contains directories such as `libs`, as well as images and other files.

The `HybridWebContainer` solution includes a set of sample files that you can include in your project. After modifying the code in the sample files, rebuild your project: to preserve your changes in the generated code. Always test your changes before using the resulting application.

In the `HybridWebContainer` project, the `docs` directory includes JavaDoc documentation for applications in `com.sybase.hwc`, and the library in `com.sybase.hybridApp`.

Windows Mobile Customization Touch Points

Touch points for Hybrid Web Container customizations are indicated in code by comments of the form `WM_CUSTOMIZATION_POINT_`*customization*.

Touch Point	Description
WM_CUSTOMIZATION_POINT_BRAND	Change application name, copyright, and developer information in the About form.
WM_CUSTOMIZATION_POINT_HYBRID-APPSEARCH	Make the list of Hybrid App packages searchable.
WM_CUSTOMIZATION_POINT_HYBRID-APPLIST	Change the appearance of the Hybrid App package list.
WM_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS	Create categorized views of the Hybrid App packages.
WM_CUSTOMIZATION_POINT_HYBRID-APPSORTING	Customize the criteria for sorting the Hybrid App package list.
WM_CUSTOMIZATION_POINT_MESSAGE-SORTING	Customize the criteria for sorting the message list.
WM_CUSTOMIZATION_POINT_MESSAGE-FILTERING	Change the filter used to sort the list of messages.
WM_CUSTOMIZATION_POINT_ANONYMOUS_USER	Indicates if the login mode is anonymous.
WM_CUSTOMIZATION_POINT_DEFAULT-SETTINGS	Change default server settings.
WM_CUSTOMIZATION_POINT_PRESET-SETTINGS	Hard-code settings for the Settings screen so they do not appear on the device. This prevents the user from changing the settings.
WM_CUSTOMIZATION_POINT_HTTPHEADERS	Set HTTPS headers for the Windows Mobile Hybrid Web Container to include authentication tokens.
WM_CUSTOMIZATION_POINT_HTTPERRORHANDLERS	Change the handling of HTTP errors.
WM_CUSTOMIZATION_POINT_TOKENERROR	Change how the client engine handles authentication token errors (for example, when a token expires).

Look and Feel Customization of the Windows Mobile Hybrid Web Container

Customizations you can make to the look and feel include changing the splash screen, changing the Hybrid App icons and name, changing the Hybrid App package icons, changing labels and text, adding support for new languages, and so on.

Changing the Hybrid Web Container Icon

Replace the icon shown on the home screen.

Changing the container icon also changes the image used on the About screen, and the image that sometimes shows up in the title bar.

1. In Solution Explorer, navigate to `HybridWebContainer\Resources\Images`.
2. Replace the `icon.ico` file with your version.

The new image must use the same name and extension as the original file, and the same resolution.

3. Rebuild and test the project.

Changing the Windows Mobile Hybrid App Package Icon

Modify the Hybrid App package application icon.

You cannot add new icons to the folder, but you can replace the existing icon images, using the same file name. The Hybrid App application icons are named `ampiconindex.png`, where *index* is a number between 30 and 116. The default Hybrid App icon is `ampicon48.png`. This is also the icon shown on the menu item that lists all the Hybrid Apps.

Each Hybrid App icon uses a pair of associated images:

- **`ampiconindexp.png`** – represents a processed message (indicated by the *p* suffix). Processed means the message has been submitted to the server.
- **`ampiconindex.png`** – is for unprocessed messages, which have not been submitted to the server.

1. Identify the image currently used by the Hybrid App package that you want to replace. When you build the Hybrid Web Container with custom icons, the original icons still appear in SAP Control Center and in SAP Mobile WorkSpace.
2. In Solution Explorer, navigate to the `HybridWebContainer\Resources\Images` folder.
3. Replace the `ampiconindex.png` and `ampiconindexp.png` image files with the new images.

Note: For each icon file that you replace, use the same name, extension, and resolution as the original. To preserve the original image make a copy of it. To prevent the copy from interfering with resource indexing, place it in a different folder.

4. Rebuild and test the Hybrid Web Container.

Implementing a Custom HybridAppList Screen

Add a custom HybridAppList screen.

Use the CustomCode sample files as the starting point for your customization.

1. In Visual Studio Solution Explorer, click the **Show All** button.
2. Include all the files in the **CustomCode** folder.
3. Modify the code in your copy of the included files.

You can modify these files to customize the HybridAppList screen:

- **MyHybridAppListScreen** – class used to implement the HybridAppList screen.
- **HybridAppComparer** – comparer used by MyHybridAppListScreen to sort the Hybrid Application order.
- **HybridAppFilter** – filter used by MyHybridAppListScreen to filter the Hybrid App.
- **CustomizationHelper** – class that integrates the HybridAppListScreen into the Hybrid Web Container.

4. Rebuild and test your project.

Customizing the About Screen and Other Branding

Customize the About screen.

1. In Solution Explorer, click the **Show All** button.
2. Include all the files in the **CustomCode** folder.
3. Modify the code in your copy of the included files.

Code related to this customization is:

```
public override void ShowAboutForm()
{
    System.Text.StringBuilder _sb = new
System.Text.StringBuilder();
    _sb.Append("Copyright 2012 Esabys, Inc.");
    _sb.Append("\r\n");
    _sb.Append("Version: 1.0"); _
sb.Append("\r\n"); _
sb.Append("Build id:20120518-0123");
    MessageBox.Show(_sb.ToString(), Consts.APP_TITLE,
    MessageBoxButtons.OK,
    MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button1);
}
```

4. Rebuild and test your project.

Adding a Splash Screen

Add a splash screen to the Hybrid Web Container.

1. In Visual Studio Solution Explorer, click the **Show All** button.
2. Include all the files in the **CustomCode** folder.
3. Modify the code in your copy of the included files.
 - **SplashForm** – class used to implement the Splash screen. It starts a timer to show the splash image in about one second.
 - **SplashBitmap.png** – image shown in the splash screen.
 - **CustomRes.resx** – resource file that contains the image file.
 - **CustomizationHelper** – class that integrates the Splash Screen into the Hybrid Web Container. When the application starts, CustomizationHelper displays the splash screen.
4. Rebuild and test your project.

Changing Labels and Text

You can customize most of the text found in labels, dialogs, or error messages used by the Hybrid Web Container.

1. In your project, open `HybridWebContainer\strings.resx` for editing.

This file contains the text for error messages, screen titles, screen labels, validation messages, and so on.
2. Make your changes to `strings.resx` and save the file.

Note: Make the same changes for each language to which you translate your text. Edit the `Strings.xx.res` file, where `xx` is the ISO639 code for the language (for example, `it` for Italian).

Adding a New Language

Add support for a new language to the Hybrid Web Container.

1. In Solution Explorer, create a new subfolder under `HybridWebContainer\Resources` named `Strings_xx.res`, where `xx` is the ISO639 code for the language (for example, `it` for Italian).
2. Add a file called `Strings.xx.res` to the new folder.

You can copy the default `Strings.res` file from `HybridWebContainer\Resources\Strings`, and use the copy as a template for the new `Strings.xx.res` file.
3. In the language-specific `Strings.xx.res` file, add your translated text.

You need not include strings that do not require localization. Any strings that are omitted from localization are removed from the default `Strings.res` file.

Default Behavior Customization of the Windows Mobile Hybrid Web Container

You can add or remove screens from the Hybrid Web Container, and change the behavior, such as sorting and filtering of messages.

Customizing Settings Screen Fields

Hide fields in the Settings screen or change their default values.

1. In Visual Studio, open the `CustomizationHelper` class in the `CustomCode` folder.
2. Override the `DefaultServerSettings` method.
3. Initialize the default server settings and return them outside of the `DefaultServerSettings` method.
4. For each field you want to remove from the Settings screen, set its value to value to null. In this example, the server name field is visible but no default value is assigned; the server port is set to 5001 but the field is hidden:

```
public override ServerSettings DefaultServerSettings
{
    get
    {
        if (m_ServerSettings == null)
        {
            m_ServerSettings = new ServerSettings();

            // Server name will be shown and initialized as empty.
            m_ServerSettings.ServerName.IsVisible = true;
            m_ServerSettings.ServerName.HasValue = false;

            // Server port will NOT be shown and initialized as 5001.
            m_ServerSettings.ServerPort.IsVisible = false;
            m_ServerSettings.ServerPort.HasValue = true;
            m_ServerSettings.ServerPort.Value = 5001;

            // Other fields will be shown.
        }
        return m_ServerSettings;
    }
}
private ServerSettings m_ServerSettings;
```

Notes:

- By default, all fields are shown.
- To hide a field, set its `IsVisible` property to “false”.

- To change a field's initial value, set HasValue to “true”, and specify a value in the Value property.
-

Using Multiple Hybrid Web Containers on the Same Windows Mobile Device

You can configure two or more Hybrid Web Containers on a Windows Mobile device.

Each container can be installed separately on the same device, can connect to a different server, and can be used independently.

1. Create a Visual Studio project for each container.
 2. For each container, edit the project's `config.properties` file and specify a unique AppID property for your container.
For example: `AppID="HWC1"`.
-

Note: Do not change the AppID property at runtime.

3. Rebuild the project, as described in *Building the Windows Mobile Hybrid Web Container Using the Provided Source Code*.
4. Configure the container's CAB build. In each project, edit the `OneBridge_ppc.inf` file and customize these properties:

AppName – provide a unique name for each container.

InstallDir – enter the path where the container is to be installed on the device. Each container must have a different path.

Shortcuts – declare a shortcut that launches the container application. Users can change shortcut names. Shortcut names do not have to be unique.

Here are sample customized lines in `OneBridge_ppc.inf`:

```
[CEStrings]
AppName = "HWC"
InstallDir=%CE1%\Sybase\%AppName%
...
[Shortcuts.All]
Hybrid Web Container,0,HWCA.exe,%CE1%
```

5. Build the CAB file for each container, as described in *Packaging a CAB File*.

Sorting the List of Hybrid App Packages

Change the default sorting of the list of Hybrid App packages.

By default, the Hybrid Web Container displays Hybrid App package names in alphabetical order. This example changes the list to sort case-sensitively

1. Add a `HybridWebAppComparer` class that uses the base class `IComparer<HybridWebAppInfo>`.
2. Override the `Compare` method using:

```
public int Compare(HybridWebAppInfo x, HybridWebAppInfo y)
{
```

```
return string.Compare(x.DisplayName, y.DisplayName, false);
}
```

3. Open the CustomizationHelper class in the CustomCode folder.
4. Override the HybridAppComparator method using:

```
public override IComparer<HybridWebAppInfo> HybridAppComparator
{
    get { return new HybridWebAppComparer(); }
}
```

5. Save the file.

Sorting Hybrid App Messages

Sort Hybrid App messages based on different criteria.

1. Add a MessageComparer class that uses the base class IComparer<Message>.
2. Override the Compare method using this code:

```
public int Compare(Message x, Message y)
{
    int iModuleId1 = x.ModuleId;
    int iModuleId2 = y.ModuleId;

    int iCompareResult = 0;
    if (iModuleId1 < iModuleId2)
    {
        iCompareResult = -1;
    }
    if (iModuleId1 > iModuleId2)
    {
        iCompareResult = 1;
    }
    if (iCompareResult == 0)
    {
        iCompareResult = x.ReceiveDate.CompareTo(y.ReceiveDate);
    }
    return iCompareResult;
}
```

3. Open the CustomizationHelper class in the CustomCode folder.
4. Override the MessageComparator using:

```
public override IComparer<Message> MessageComparator
{
    get { return new MessageComparer(); }
}
```

5. Save the file.

Filtering Hybrid App Messages

Prevent the Hybrid App from displaying some messages.

1. Add a MessageFilter class that uses the base class IFilter<Message>.

2. Override the select method using code similar to:

```
public bool Select(Message subject)
{
    if (subject.Priority ==
        MessageConsts.EMAIL_STATUS_IMPORTANCE_HIGH)
    {
        return false;
    }
    return true;
}
```

3. Open the CustomizationHelper class in the CustomCode folder.
4. Override the MessageFilter method using:

```
public override IFilter<Message> MessageFilter
{
    get
    {
        return new MessageFilter();
    }
}
```

5. Save the file.

Setting HTTP Headers

Set HTTP headers for the Hybrid Web Container to include authentication tokens.

These methods in the Hybrid Web Container template source code show how to set HTTP headers:

- **getHttpHeaders** – override this method to set the authentication tokens.
- **OnHTTPError** – listener called by the communication layer when an HTTP error occurs.
- **OnTokenError** – listener called by the client engine when Hybrid App token authentication failure occurs.

1. In Visual Studio, open the CustomizationHelper class in the CustomCode folder.

2. Override the getHttpHeaders method and uncomment its contents.

The stub code shows how to add headers and cookies. Simply replace the header and cookie assignments with your own.

3. Refresh the HTTP headers.

It is a good idea to refresh the HTTP headers in the OnTokenError method, which is called when a Hybrid App token authentication failure occurs.

Here is a common way to do this:

- a. Maintain member variables that contain the values for the headers you want to set.
- b. Override the GetHttpHeaders method to use the value in those member variables when it sets the headers.
- c. In OnTokenError, update the member variables with the new header values.

- d. Call `UpdateHttpHeaders` again.
4. If you have custom code to run when an HTTP error occurs, add the code to override the `OnHttpError` method.

Your method is called any time there is an HTTP error. You can use it to inform the user of errors, or to perform other custom steps in response to particular error codes.

Customizing OK Button Behavior

Control behavior when the OK button is clicked in Hybrid App forms.

To customize the OK button in the `MessageList`, `ApplicationList`, and `Application` forms, override the `OnClosing` methods for those forms:

```
internal virtual void OnClosingMessageListForm( MessageListForm
form )
{
}
```

```
internal virtual void
OnClosingApplicationListForm( HybridWebAppListForm form )
{
}
```

```
internal virtual void OnClosingHybridAppForm( HybridWebAppForm
form )
{
}
```

Packaging a CAB File

After rebuilding your customized Hybrid Web Container, package the generated files into a cab file that can be installed on a device.

Prerequisites

Install ActivePerl, available for download from <http://www.activestate.com/>. After installing ActivePerl, add it to the environment path. When you run Perl at the command prompt, the script is executed by the first Perl.exe it encounters in the list of paths in the PATH environment variable. To ensure the script is executed by the correct Perl interpreter, specify the complete path to the Perl.exe you want to use.

Task

When you build the template project, the binary release files are generated into the template output folder.

1. Open a Command Prompt.
2. In the Command Prompt, navigate to the `template\Tools` folder of your project.
3. Run the `buildcab` script, specifying the path to the location of the release files generated when you built the project.

For example:

```
perl buildcab.pl ..\bin\Release
```

The packaged CAB file is generated in `template\Tools`.

Prepackaged Hybrid Apps

You can use the Hybrid Web Container as the runtime shell for a single Hybrid App.

When you use the prepackaged Hybrid App, the application is launched immediately and there is no listview of Hybrid Apps. This allows for a single view of the Hybrid App. You can still assign other applications to the Hybrid Web Container, but while running in this new mode, only the Hybrid App designated as the default is active.

Note: Connection settings for the Hybrid Web Container must be configured before the prepackaged Hybrid Web Container can launch.

When the user closes the default Hybrid App, he or she can then view the messages associated with that application in the Hybrid Web Container.

Including a Prepackaged Hybrid App in the Android Hybrid Web Container

Run a prepackaged Hybrid App so that the Hybrid Web Container functions as a single-purpose application rather than a general purpose one.

1. Package the Hybrid App files.

You can use a Hybrid App that was generated with the Hybrid App Designer, or you can use the packaging tool to generate a new Hybrid App.

When packaging the Hybrid App, optimize the size by generating a version for each specific platform that includes only files for that platform.

See *Packaging Hybrid Apps Using the Packaging Tool*.

2. Copy the Generated Hybrid App folder under the package tool workspace, or copy the Generated Hybrid App folder under the SAP Mobile WorkSpace, to the `assets` directory of the Android Hybrid Web Container template.
3. Remove the ZIP file from the folder.
4. Refresh the Eclipse workspace.
5. Open the `CustomizationHelper.java` file, locate the `ANDROID_CUSTOMIZATION_POINT_PREPACKAGED_APP` customization point that accompanies the `getPrepackageAppPath` function, and change the contents of this function to return the name of the top-level directory you just added to the project.

If the prepackaged Hybrid App manages the server connection by itself and wants to exit the Hybrid Web Container after exiting the prepackaged Hybrid App, change return value of the method `exitHWConPrepackagedAppClose` to `true`.

6. To optionally enable the Hybrid Web Container to exit after closing the prepackaged Hybrid App, change the return value of the `exitHWConPrepackagedAppClose` method to `true`.

The default return value of the method is `false`.

Including a Prepackaged Hybrid App in the BlackBerry Hybrid Web Container

Run a prepackaged so that the Hybrid Web Container functions as a single-purpose application rather than a general purpose one.

Prerequisites

Install the BlackBerry Java Plug-in for Eclipse.

Task

1. Package the Hybrid App files.

You can use a Hybrid App that was generated with the Hybrid App Designer, or you can use the packaging tool to generate a new Hybrid App.

When packaging the Hybrid App, optimize the size by generating a version for each specific platform that includes only files for that platform.

See *Packaging Hybrid Apps Using the Packaging Tool*.

2. In Eclipse, import the BlackBerry Hybrid Web Container template as a legacy BlackBerry project:
 - a) Select **File > Import**.
 - b) Expand the **BlackBerry** folder.
 - c) Select **Import Legacy BlackBerry Projects**.
 - d) Click **Next**.
 - e) Specify the JRE and, in the BlackBerry Workspace field, browse to the `HWCTemplate.jdw` file and select the project to import.
 - f) Select **Copy BlackBerry projects into workspace** to create a copy of the imported project in the Eclipse workspace.
 - g) Click **Finish**.
3. Copy the generated Hybrid App folder under the package tool workspace to the `res` directory of the imported Eclipse BlackBerry Hybrid Web Container project.
4. Remove the ZIP file from the folder, and refresh the Eclipse workspace.
5. Open the `CustomizationHelper.java` file for editing.

6. Find the `BLACKBERRY_CUSTOMIZATION_POINT_PREPACKAGE_APP` that accompanies the `getPrepackagedAppPath` function, and change the contents of the function to return the name of the top-level directory you just added to the project.

If the prepackaged Hybrid App manages the server connection by itself and wants to exit the Hybrid Web Container after exiting the prepackaged Hybrid App, change return value of the method `exitHWCOnPrepackagedAppClose` to `true`.

7. Save the `CustomizationHelper.java` file.

Including a Prepackaged Hybrid App in the iOS Hybrid Web Container

Run a prepackaged Hybrid App in the iOS Hybrid Web Container so that the Hybrid Web Container functions as a single-purpose application rather than a general purpose one.

1. Package the Hybrid App files.

You can use a Hybrid App that was generated with the Hybrid App Designer, or you can use the packaging tool to generate a new Hybrid App.

When packaging the Hybrid App, optimize the size by generating a version for each specific platform that includes only files for that platform.

See *Packaging Hybrid Apps Using the Packaging Tool*.

2. Copy the generated Hybrid App folder to a location that is accessible to your Xcode project.
3. In the Xcode Project Navigator, right-click the **Resources** group folder, and select **Add Files to HWC**.
4. Navigate to the directory you just created that contains the generated package, and select the top-level directory of the package.
Create folder references, not group references, when you add the files. The directories appear directly under `Resources`.
5. In the Project Navigator, find and open the `CustomizationHelper.m` file, which is located in the `Classes` group folder.
6. Locate the customization point, designated by the comment `IOS_CUSTOMIZATION_POINT_PREPACKAGED_APP`, that accompanies the `getPrepackagedAppPath` function, and change the contents of this function to return the name of the top-level directory you just added to the project.
7. Rebuild the `HWC.xcodeproj` project.
 - a) From the Xcode menu, select **Product > Clean**.
 - b) Select **Product > Build**.

Including a Prepackaged Hybrid App in the Windows Mobile Hybrid Web Container

Run a prepackaged so that the Hybrid Web Container functions as a single-purpose application rather than a general purpose one.

1. Package the Hybrid App files.

You can use a Hybrid App that was generated with the Hybrid App Designer, or you can use the packaging tool to generate a new Hybrid App.

When packaging the Hybrid App, optimize the size by generating a version for each specific platform that includes only files for that platform.

See *Packaging Hybrid Apps Using the Packaging Tool*.

2. Include the generated Hybrid App files in a Visual Studio project:

- a) Copy the generated Hybrid App files to your Visual Studio project.
- b) Open the `HybridWebContainer.csproj`, which is in the `WM_HWC<version>.zip` file.
- c) In Visual Studio Solution Explorer, select **Show All Files**.
- d) Right-click the Hybrid App folder and select **Include in Project**.
- e) Set the **Copy to Output Directory** property to **Copy if newer** for all the files under this folder.

Note: You can select all the files using the SHIFT CTRL keys, and then set the property for all the selected files.

3. In the CustomCode folder, create a Partial class for

`CustomizationHelper.cs`.

4. In the Partial class of the CustomizationHelper.cs file, create a method to override the property `PrepackageAppPath` to return the full installation path of the Hybrid App on the device.

```
public override string PrepackageAppPath
{
    get
    {
        return @"\"Program Files\\sybase\\hwc\\iMOWebProto";
    }
}
```

5. Rebuild the project.

6. Include the prepackaged Hybrid App in a CAB file:

Most Windows Mobile applications are deployed as CAB files. You can find information about creating CAB files at <http://msdn.microsoft.com/en-us/library/aa448616.aspx> and information about the `.inf` file at <http://msdn.microsoft.com/en-us/library/aa448654.aspx>.

- a) Open the `onebridge_ppc.inf` file, which is located in the `Tools` folder of the Hybrid Web Container template project.
- b) Add the prepackaged Hybrid App folders in the `[SourceDisksNames.ARM]` section:

```
[SourceDisksNames.ARM]
1="PPC",,unsigned
3="zh-CN",,"unsigned\zh_CN"
4="zh-HK",,"unsigned\zh_HK"
5="de",,"unsigned\de"
6="fr",,"unsigned\fr"
7="fr-CA",,"unsigned\fr_CA"
8="ja",,"unsigned\ja"
9="es",,"unsigned\es"
10="prepackage",,"unsigned\prepackage"
11="prepackage.css",,"unsigned\prepackage\html\css"
12="prepackage.default",,"unsigned\prepackage\html\default"
13="prepackage.en",,"unsigned\prepackage\html\en"
14="prepackage.en_US",,"unsigned\prepackage\html\en_US"
15="prepackage.icon",,"unsigned\prepackage\html\icon"
16="prepackage.images",,"unsigned\prepackage\html\images"
17="prepackage.js",,"unsigned\prepackage\html\js"
18="prepackage.html",,"unsigned\prepackage\html"
```

- c) List all the required files in the `[SourceDisksFiles.ARM]` section:

```
[SourceDisksFiles.ARM]
CMessagingClient.2.2.0.dll=1
OBSetup.dll=1
HWCA.exe=1
HWCEngine.lnk=1
Plugins.xml=1
; other files
hybridapplib.dll=1
SQLite.Interop.DLL=1
System.Data.SQLite.dll=1
version.txt=1
config.properties=1
WorkflowClient.xml=10
index.xml=10
manifest.xml=10
"Stylesheet.css"=11
"hybridapp.html"=12
"hybridapp.html"=13
"hybridapp.html"=14
"API.js"=17
"Callbacks.js"=17
```

- d) Define the installation target in the `[DestinationDirs]` section:

```
[DestinationDirs]
Files.ARM = 0,%InstallDir%
Shortcuts.All = 0,%CE4%
System.ARM = 0,%CE2%
zh-CN = 0,%InstallDir%\zh-CN
zh-HK = 0,%InstallDir%\zh-HK
de = 0,%InstallDir%\de
fr = 0,%InstallDir%\fr
```

```

fr-CA = 0,%InstallDir%\fr-CA
ja = 0,%InstallDir%\ja
es = 0,%InstallDir%\es
prepackage.css = 0,"%InstallDir%\prepackage\html\css"
prepackage.default = 0,"%InstallDir%\prepackage\html\default"
prepackage.en = 0,"%InstallDir%\prepackage\html\en"
prepackage.en_US = 0,"%InstallDir%\prepackage\html\en_US"
prepackage.icon = 0,"%InstallDir%\prepackage\html\icon"
prepackage.images = 0,"%InstallDir%\prepackage\html\images"
prepackage.js = 0,"%InstallDir%\prepackage\html\js"
prepackage.html = 0,"%InstallDir%\prepackage"
prepackage = 0,"%InstallDir%\prepackage"

```

e) Describe each file mapping in the File List section:

```

[prepackage.css]
Stylesheet.css,,0

[prepackage.default]
hybridapp.html,,0

[prepackage.en]
"hybridapp.html"

[prepackage.en_US]
"hybridapp.html"

[prepackage.icon]

[prepackage.images]

[prepackage.js]
"API.js"
"Callbacks.js"
"Camera.js"
"Certificate.js"
"Custom.js"
"datajs-1.0.2.js"
"ExternalResource.js"
"json2.js"
"MAKit.js"
"Resources.js"
"SUP0.js"
"SUPStorage.js"
"Timezone.js"
"Utils.js"
"HybridApp.js"
"WorkflowMessage.js"

[prepackage]
"index.xml"
"manifest.xml"
"WorkflowClient.xml"

[prepackage.html]
hybridapp.html

```

```
[System.ARM]
manifest.xml,,0
```

- f) Include all the file lists in section [DefaultInstall.ARM]:

```
[DefaultInstall.ARM]
CopyFiles=Files.ARM, System.ARM, de, fr, fr-CA, es, zh-CN, zh-
HK, ja, prepackage,
prepackage.css, prepackage.default, prepackage.en, prepackage.en_
US,
prepackage.icon, prepackage.images, prepackage.js, prepackage.htm
l
```

- g) Run: `buildcab.pl <Path to project output>`.

7. Deploy and run the customized Hybrid Web Container on the device or emulator.
- a) Compile the Hybrid Web Container.
 - b) Deploy the Hybrid Web Container to the device or emulator.
 - c) Run and test the prepackaged Hybrid App.

Adding Native Device Functionality to the Hybrid Web Container

PhoneGap (now known as Apache Cordova) is an open source framework that leverages Web technologies such as HTML, CSS, and JavaScript to access native (system and third-party) functionality across platforms.

SAP Mobile Platform comes with the Cordova libraries, which handle common tasks supported by most devices, linked in and ready to use. Integrating PhoneGap plug-ins with Hybrid Web Containers allows you to extend the set of APIs available within a Hybrid App. See www.phonegap.com for information about the supported PhoneGap APIs.

PhoneGap API calls are made from the Hybrid App JavaScript files.

Supported JavaScript PhoneGap APIs

The Hybrid Web Container comes with the PhoneGap library linked in and ready to use.

The PhoneGap library included with SAP Mobile Platform handles common native tasks supported by Android, BlackBerry, iOS and Windows Mobile devices, for example, accessing geolocation, accessing contacts, and invoking calls to make those common functions available to JavaScript.

Note: Keep in mind that PhoneGap APIs cannot be accessed successfully until initialization has taken place. If you make calls to the PhoneGap API from the `customAfterShowScreen` function, they should occur only after the PhoneGap subsystem is initialized and ready to execute these calls. For more information, see <http://wiki.phonegap.com/w/page/36868306/ UI%20Development%20using%20jQueryMobile#HandlingPhoneGapsdevicereadyevent>.

You can make PhoneGap calls from the Hybrid Web Container JavaScript, such as `Custom.js`. For example, to save an entry to the contacts database, you can implement something similar to:

```
var contact = navigator.contacts.create();
    contact.nickname = "Plumber";
    var name = new ContactName();
    name.givenName = "Jane";
    name.familyName = "Doe";
    contact.name = name;
    // save
    contact.save (onSaveSuccess, onSaveError);
```

You can use both Hybrid Web Container JavaScript APIs and PhoneGap APIs in a single application. For information about PhoneGap APIs, see <http://docs.phonegap.com>.

Table 9. PhoneGap Supported Features

API	Object and Function	Platform
Accelerometer		
	accelerometer <ul style="list-style-type: none"> • <code>getCurrentAcceleration</code> <hr/> Note: On iOS, this function must be called after <code>watchAcceleration</code> . <hr/> <ul style="list-style-type: none"> • <code>watchAcceleration</code> • <code>clearWatch</code> 	<ul style="list-style-type: none"> • Android • iOS • BlackBerry
	Acceleration <ul style="list-style-type: none"> • <code>x</code> • <code>y</code> • <code>z</code> • <code>timeStamp</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
Camera		
	Camera <ul style="list-style-type: none"> • <code>getPicture (Camera.PictureSourceType.CAMERA)</code> • <code>getPicture (Camera.PictureSourceType.PHOTOLIBRARY)</code> • <code>getPicture (Camera.PictureSourceType.SAVEDPHOTOALBUM)</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile

API	Object and Function	Platform
	CameraOptions <ul style="list-style-type: none"> quality dedestinationType.DATA_URL dedestinationType.FILE_URI FILE_URI is the default. allowEdit encodingType targetWidth targetHeight 	<ul style="list-style-type: none"> Android BlackBerry iOS Windows Mobile
Capture		
	Capture <ul style="list-style-type: none"> captureAudio <p>Note: On Android, whether this works depends on which application the device uses to record the audio. You can use <code>media.record</code> instead to work around this issue.</p> <ul style="list-style-type: none"> captureImage captureVideo 	<ul style="list-style-type: none"> Android BlackBerry iOS
	MediaFile <ul style="list-style-type: none"> getFormatData 	<ul style="list-style-type: none"> Android iOS
Compass		
	compass <ul style="list-style-type: none"> getCurrentHeading watchHeading clearWatch watchHeadingFilter 	<ul style="list-style-type: none"> Android iOS
	compass.Heading <ul style="list-style-type: none"> magneticHeading trueHeading headingAccuracy timestamp 	<ul style="list-style-type: none"> Android iOS
Connection		

API	Object and Function	Platform
	network.connection.type	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
Contacts		
	contacts.create	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	contacts.find	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	contact.clone	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	Contacts.remove Note: On Android, there is an issue with contacts not being fully removed. See https://issues.apache.org/jira/browse/CB-75 .	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	Contacts.save	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
Device		
	Device.name	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	Device.phonegap	<ul style="list-style-type: none"> • Android • BlackBerry • iOS

API	Object and Function	Platform
	Device.platform	<ul style="list-style-type: none"> Android BlackBerry iOS
	Device.uuid	<ul style="list-style-type: none"> Android BlackBerry iOS
	Device.version	<ul style="list-style-type: none"> Android BlackBerry iOS
Events		
	Deviceready	<ul style="list-style-type: none"> Android iOS
	Pause	<ul style="list-style-type: none"> Android
	Resume	<ul style="list-style-type: none"> Android
	Online	<ul style="list-style-type: none"> Android iOS
	Offline	<ul style="list-style-type: none"> Android iOS
	Batterycritical	iOS
	Batterylow	iOS
	Batterystatus Note: On Android, PhoneGap 1.4.1, this does not work due to a known issue. See https://issues.apache.org/jira/browse/CB-173 .	iOS
	Menubutton	<ul style="list-style-type: none"> Android
	Searchbutton	<ul style="list-style-type: none"> Android
File		

API	Object and Function	Platform
	DirectoryEntry <ul style="list-style-type: none"> • copyTo • moveTo • toURI • remove • removeRecursively • getParent • createReader • getDirectory • getFile 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	FileEntry <ul style="list-style-type: none"> • copyTo • moveTo • toURI • remove • getParent • createWriter • file 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	FileReader <ul style="list-style-type: none"> • abort • readAsDataURL • readAsText 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	FileWriter <ul style="list-style-type: none"> • abort • seek • truncate • write 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	DirectoryReader <ul style="list-style-type: none"> • readEntries 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile

API	Object and Function	Platform
	LocalFileSystem <ul style="list-style-type: none"> requestFileSystem resolveLocalFileSystemURI 	<ul style="list-style-type: none"> Android BlackBerry iOS Windows Mobile
	FileTransfer <ul style="list-style-type: none"> upload download 	<ul style="list-style-type: none"> Android BlackBerry iOS Windows Mobile
Geolocation		
	geolocation <ul style="list-style-type: none"> getCurrentPosition <hr/> Note: This function does not work on the Android Galaxy Tab P1000 device. <hr/> <ul style="list-style-type: none"> watchPosition clearWatch 	<ul style="list-style-type: none"> Android BlackBerry iOS Windows Mobile
	Position <ul style="list-style-type: none"> coords timestamp 	<ul style="list-style-type: none"> Android BlackBerry iOS Windows Mobile

API	Object and Function	Platform
	<p>Coordinates</p> <ul style="list-style-type: none"> latitude longitude altitude accuracy <hr/> <p>Note: On Android, the returned accuracy property is always null.</p> <hr/> <ul style="list-style-type: none"> altitudeAccuracy <hr/> <p>Note: On Android, the returned altitudeAccuracy property is always null.</p> <hr/> <ul style="list-style-type: none"> heading <hr/> <p>Note: Android only. The returned heading property is always null.</p> <hr/> <ul style="list-style-type: none"> speed <hr/> <p>Note: On Android, the returned speed property is always null.</p> <hr/>	<ul style="list-style-type: none"> Android BlackBerry iOS Windows Mobile
Media		
	Media.play	<ul style="list-style-type: none"> Android iOS Windows Mobile
	Media.pause	<ul style="list-style-type: none"> Android iOS
	Media.stop	<ul style="list-style-type: none"> Android iOS Windows Mobile
	Media.release	<ul style="list-style-type: none"> Android iOS
	Media.record	<ul style="list-style-type: none"> Android iOS
	Media.startRecord	<ul style="list-style-type: none"> Android iOS

API	Object and Function	Platform
	Media.stopRecord	<ul style="list-style-type: none"> • Android • iOS • Windows Mobile
	Media.getCurrentPosition	<ul style="list-style-type: none"> • Android • iOS • Windows Mobile
	Media.seekTo	<ul style="list-style-type: none"> • Android • iOS • Windows Mobile
	Media.getDuration Note: On Android, this function returns a value without an error but always returns -1, which indicates duration is not available.	<ul style="list-style-type: none"> • Android • iOS • Windows Mobile
Notification		
	Notification.beep	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	Notification.confirm	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	Notification.alert	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
	Notification.vibrate	<ul style="list-style-type: none"> • Android • BlackBerry • iOS • Windows Mobile
Storage		

API	Object and Function	Platform
	<p>window</p> <ul style="list-style-type: none"> • <code>OpenDatabase</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	<p>Database</p> <ul style="list-style-type: none"> • <code>transaction</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	<p>SQLTransaction</p> <ul style="list-style-type: none"> • <code>executeSQL</code> <p>Note: On Android, queries on the first database created do not work. You can work around this by creating and opening two databases, the first of which can have the size of 0, and the second to use as you normally do. For example:</p> <pre>var db = window.openDatabase("aName1", "1.0", "aName1", 0); db = window.openDatabase("aName2", "1.0", "aName2", 200000);</pre>	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	<p>SQLResultSet</p> <ul style="list-style-type: none"> • <code>insertId</code> • <code>rowAffected</code> <p>Note: The returned <code>SQLResultSet</code> object does not contain a <code>rowAffected</code> property, as the PhoneGap API states. Instead, use <code>rowsAffected</code>.</p> <ul style="list-style-type: none"> • <code>rows</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	<p>SQLResultSetList</p> <ul style="list-style-type: none"> • <code>item</code> • <code>length</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS
	<p>SQLError</p> <ul style="list-style-type: none"> • <code>code</code> • <code>message</code> 	<ul style="list-style-type: none"> • Android • BlackBerry • iOS

API	Object and Function	Platform
	localStorage <ul style="list-style-type: none"> • key • getItem • setItem • removeItem • clear 	iOS

Implementing PhoneGap

The recommended methods of implementing PhoneGap are to use the AppFramework, or to load PhoneGap in the same way as the Apache Cordova installation does.

To use the same HTML for every platform, include the Cordova files as javascript files, then dynamically load that code based on the platform that is running. The Cordova files are packaged in the *SMP_HOME\UnwiredPlatform\MobileSDKversion\HybridApp\Containers\Platform* directories.

```
function loadPhoneGap() {
var jsfile = null;
var pre = "";
var language = hwc.getURLParam("lang");
if (!(language === undefined) && (language.length > 0)){
pre = "../";
}
if (hwc.isAndroid()) {
jsfile = pre + "js/android/cordova-2.0.0.javascript";
}
else if (hwc.isIOS()) {
jsfile = pre + "js/ios/cordova-2.0.0.javascript";
}
else if (hwc.isBlackBerry()) {
jsfile = pre + "js/blackberry/cordova-2.0.0.javascript";
}
if (jsfile) {
var req = null;
if (window.XMLHttpRequest) {
req = new XMLHttpRequest();
}
else { // code for IE6, IE5
req = new ActiveXObject("Microsoft.XMLHTTP");
}
req.open("GET", jsfile, false);
req.send(null);
// Need to call eval with the global context
window[ "eval" ].call( window, req.responseText );
}
}
loadPhoneGap();
```

Initializing PhoneGap for Storage Methods

If your application calls a storage function (hwc.SUPStorage or hwc.SharedStorage, the PhoneGap must have been initialized first. If you generate your application in the Hybrid App Designer, the application detects the initialization automatically. However, if you do not generate your application using Designer, you must add code to recognize when PhoneGap is initialized.

For example, in Custom.js, add this code:

- This new function displays a notification:

```
function phoneGapIsready() {
    showAlertDialog("PhoneGap is ready");
}
```

- This customization detects when the PhoneGap initialization occurs and displays your notification:

```
function hwc.customAfterHybridAppLoad() {
    document.addEventListener("deviceready",
    phoneGapIsReady, false);
}
```

Alternatively, detect the initialization directly in your Hybrid App HTML file:

```
<body onload='document.addEventListener("deviceready",
phoneGapIsReady, false)'/>
```

PhoneGap Custom Plug-ins

You can write custom plug-ins for PhoneGap.

Custom PhoneGap plug-ins have a JavaScript component that exposes the custom native component and a native component. See the *PhoneGap* documentation for information about PhoneGap plug-ins.

Custom Plug-ins for the Android Hybrid Web Container

Integrate PhoneGap (Cordova) plug-ins with the Android Hybrid Web Container.

In general, adding a custom plug-in to Hybrid Web Container is identical to adding a plug-in to any PhoneGap application. The basic steps are as follows (see the *PhoneGap Wiki* for details).

1. Create an Android project.
2. Include Cordova dependencies.
3. Implement the plug-in class.
4. Implement the plug-in JavaScript.

Adding a Custom Plug-in to the Android Hybrid Web Container

Add a PhoneGap (now called Cordova) plug-in to the Android Hybrid Web Container.

Prerequisites

Download and install the Android Developer Tools (ADT), available from <http://developer.android.com/sdk/index.html>.

Task

1. In Eclipse, import the HybridWebContainer project:
 - a) Select **File > Import**.
 - b) Expand **Android**, choose **Existing Android Code into Workspace**, and click **Next**.
 - c) In **Import Projects**, click **Browse** and select the root directory of the Android project to import.
For example, if you have previously unpacked the Android HWC container to `SMP_HOME\MobileSDKversion\HybridApp\Containers\Android\Android_HWC_version`, select that folder and click **OK**.
 - d) Click **Finish**.

2. In the HybridWebContainer project, open `res/xml/config.xml`.

3. Add your custom plug-in.

For example:

```
<plugin name="DirectoryListPlugin"
value="com.sybase.hwc.DirectoryListPlugin" />
```

4. Add plug-in images to the HybridWebContainer project.

The plug-in used in this example does not include images, but they are allowed in plug-ins. Images for plug-ins are usually stored in `res\drawable`.

5. Add the Java source file that implements the custom plug-in, for example, `DirectoryListplugin.java`.

This example PhoneGap plug-in lists all files on the SDCard of the device.

```
/**
 * Example of Android PhoneGap Plugin
 */
package com.sybase.hwc;

import java.io.File;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;

import org.apache.cordova.api.Plugin;
```

```

import org.apache.cordova.api.PluginResult;
import org.apache.cordova.api.PluginResult.Status;

/**
 * PhoneGap plugin which can be involved in following manner from
 * javascript
 * <p>
 * result example - {"filename":"/
 * sdcard","isdir":true,"children":
 * [{"filename":"a.txt","isdir":false},{...}]
 * </p>
 * <pre>
 * {@code
 * successCallback = function(result) {
 *     //result is a json
 *
 * }
 * failureCallback = function(error) {
 *     //error is error message
 * }
 *
 * window.plugins.DirectoryListing.list("/sdcard",
 *                                     successCallback
 *                                     failureCallback);
 *
 * }
 * </pre>
 * @author Rohit Ghatol
 *
 */
public class DirectoryListPlugin extends Plugin {

    /** List Action */
    public static final String ACTION="list";

    /**
     * (non-Javadoc)
     * @see
     * org.apache.cordova.api.Plugin#execute(java.lang.String,
     * org.json.JSONArray, java.lang.String)
     */
    @Override
    public PluginResult execute(String action, JSONArray data,
    String callbackId) {
        Log.d("DirectoryListPlugin", "Plugin Called");
        PluginResult result = null;
        if (ACTION.equals(action)) {
            try {

                String fileName = data.getString(0);
                JSONObject fileInfo = getDirectoryListing(new
                File(fileName));
                Log
                    .d("DirectoryListPlugin", "Returning "
                    + fileInfo.toString());
            }

```

```

        result = new PluginResult(Status.OK, fileInfo);
    } catch (JSONException jsonEx) {
        Log.d("DirectoryListPlugin", "Got JSON Exception "
            + jsonEx.getMessage());
        result = new PluginResult(Status.JSON_EXCEPTION);
    }
    } else {
        result = new PluginResult(Status.INVALID_ACTION);
        Log.d("DirectoryListPlugin", "Invalid action : "+action
+" passed");
    }
    return result;
}

/**
 * Gets the Directory listing for file, in JSON format
 * @param file The file for which we want to do directory
listing
 * @return JSONObject representation of directory list. e.g
{"filename":"/sdcard","isdir":true,"children":
[{"filename":"a.txt","isdir":false},{..}]
 * @throws JSONException
 */
private JSONObject getDirectoryListing(File file)
    throws JSONException {
    JSONObject fileInfo = new JSONObject();
    fileInfo.put("filename", file.getName());
    fileInfo.put("isdir", file.isDirectory());

    if (file.isDirectory()) {
        JSONArray children = new JSONArray();
        fileInfo.put("children", children);
        if (null != file.listFiles()) {
            for (File child : file.listFiles()) {
                children.put(getDirectoryListing(child));
            }
        }
    }

    return fileInfo;
}
}

```

6. Save the file.

These are all the changes needed for the Hybrid Web Container; you can now build it and install it on the device. What the plug-in actually does is implemented in the Java file in the **execute** function.

Testing the Plug-in

Test the PhoneGap plug-in for the Android Hybrid Web Container.

1. Create a new Mobile Application project:

- a) Select **File > New > Mobile Application Project**.

- b) In Project name, enter PhonegapTest.
- c) Click **Finish**.
- 2. Right-click the PhonegapTest project folder and select **NewHybrid App Designer**.
- 3. Click **Next**.
- 4. Select **Can be started on demand from the client** and click **Finish**.
- 5. Click **Screen Design**.
- 6. Add a Menu Item control of type Custom to the Menu, and in the General properties, enter "c" for the menu item name.

This is the key name you will use for the `customAfterMenuItemClick ()` function in the `custom.js` file.

- 7. Run the Hybrid App Generation wizard to create the directory structure Generated Hybrid App\PhonegapTest\ html\js.
- 8. Open the `custom.js` file for editing and add this code before the line `(function(hwc, window, undefined) :`

```
var dirlist = {
    getlist: function(successCallback, errorCallback) {
        PhoneGap.exec(successCallback, errorCallback,
        'DirectoryListPlugin', 'list', ["/mnt/sdcard"]);
    }
};

function getDlist() {
    dirlist.getlist(function(r) {
        var theHtml = "";
        if(r.children)
        {
            var index = 0;
            for(index = 0; index <= r.children.length; index++)
            {
                if(r.children[index]){
                    theHtml += r.children[index].filename + " \n ";
                }
            }
        }
        else
        {
            alert("No r.children!!");
        }
        alert(theHtml);
    },
    function(error) {
        alert('Error:' + error);
    });
}
```

- 9. Add this code in the `customAfterMenuItemClick ()` function:

```
if(menuItem == "c"){  
    getDlist();  
}
```

10. Regenerate the Hybrid App.

11. Assign the Hybrid App to a device that has the Hybrid Web Container with the custom plug-in.

12. On the device, run the Hybrid App, click **Menu**, and click **c**.

The `custom_plug-in.java` file appears on the SD card in the list of files.

Note: The code returns a list of files only if an SD card is configured on the device (or, on an emulator, if an SD Card is configured in AVD). If no SD card is configured, the code returns no list.

Custom Plug-ins for the BlackBerry Hybrid Web Container

Integrate PhoneGap plug-ins with the BlackBerry Hybrid Web Container.

In general, adding a custom plug-in to Hybrid Web Container is identical to adding a plug-in to any PhoneGap application. See the *PhoneGap Wiki*. The basic steps are:

1. Set up a BlackBerry Eclipse development IDE. See <http://us.blackberry.com/developers/javaappdev/javaplugin.jsp>.
2. Create the plug-in source code.
3. Provide the JavaScript API.
4. Package the plug-in source code.
5. Include the PhoneGap dependencies.

Adding a Custom Plug-in to the BlackBerry Hybrid Web Container

Add a PhoneGap plug-in to the BlackBerry Hybrid Web Container

Prerequisites

Set up the BlackBerry Eclipse development IDE. See <http://us.blackberry.com/developers/javaappdev/javaplugin.jsp>

Task

This example procedure shows the steps to create and use a custom plug-in to get battery information for the device.

1. In Eclipse, import the HybridWebContainer project.
2. Open the `plugins.xml` file, which is located in `res/xml`, and add this tag:

```
< plugin name="Battery1" value="com.sybase.hwc.Battery1"/>
```
3. Add a new Java source file called `Battery1.java` to the `src` folder, and paste in this code:

```
package com.sybase.hwc;  
import org.apache.cordova.api.Plugin;
```



```

import org.apache.cordova.api.PluginResult;
import org.apache.cordova.json4j.JSONArray;

public class Battery1 extends Plugin {
    public static final String GET_LEVEL = "getLevel";

    /**
     * Executes the requested action and returns a PluginResult.
     *
     * @param action      The action to execute.
     * @param callbackId  The callback ID to be invoked upon action
completion.
     * @param args        JSONArray of arguments for the action.
     * @return            A PluginResult object with a status and
message.
     */
    public PluginResult execute(String action, JSONArray args,
String callbackId) {
        PluginResult result = null;
        if (GET_LEVEL.equals(action)) {
            // retrieve the device battery level
            int level =
net.rim.device.api.system.DeviceInfo.getBatteryLevel();
            result = new PluginResult(PluginResult.Status.OK,
level);
        }
        else {
            result = new
PluginResult(PluginResult.Status.INVALID_ACTION,
                "Battery: Invalid action: " + action);
        }
        return result;
    }

    /**
     * Called when Plugin is paused.
     */
    public void onPause() {
    }

    /**
     * Called when Plugin is resumed.
     */
    public void onResume() {
    }

    /**
     * Called when Plugin is destroyed.
     */
    public void onDestroy() {
    }
}

```

4. Save the file.

These are all the changes needed for the Hybrid Web Container; you can now build it and install it on the device. What the plug-in actually does is implemented in the Java file in the

execute function. The rest of this example explains how to test and use the PhoneGap plug-in.

5. Create a new Hybrid App.
 - a) Select **File > New > Mobile Application Project**.
 - b) In Project name, enter PhonegapTest.
 - c) Click **Finish**.
6. Right-click the **PhonegapTest** project folder and select **New > Hybrid App Designer**.
7. Click **Next**.
8. Select **Can be started, on demand, from the client** and click **Finish**.
9. Add an **HtmlView** control to the start screen of the Hybrid App.
10. Run the Hybrid App Package Generation wizard to create the Generated Hybrid App directory structure Generated Hybrid App\PhonegapTest\ html\js.
11. Open the Custom.js file and add this code:

```
var Battery1 = {
    level: function(successCallback, errorCallback) {
        PhoneGap.exec(successCallback, errorCallback,
        'Battery1', 'getLevel', []);
    }
};

function getBatteryLevel() {
    Battery1.level(function(level) {
        alert('Battery level is ' + level);
    },
    function(error) {
        alert('Error retrieving battery level:' + error);
    });
}
```

12. Find the customAfterHybridAppLoad() function, and add this code:

```
function customAfterHybridAppLoad() {
    document.addEventListener("deviceready", getBatteryLevel,
    false );
}
```

This is the code that makes use of the plug-in.

13. Generate the Hybrid App package again.
14. Assign the Hybrid App to a device that has the modified Hybrid Web Container installed.
15. On the device, run the Hybrid App.

You see the alert message with the battery level information.

Custom Plug-ins for the iOS Hybrid Web Container

Integrate PhoneGap plug-ins with the iOS Hybrid Web Container.

In general, adding a custom plug-in to Hybrid Web Container is identical to adding a plug-in to any PhoneGap application. The basic steps are as follows (see *the PhoneGap Wiki for details*).

1. Implement the plug-in class that extends PGPlugin in an .h and .m file.
2. Implement the PhoneGap plug-in JavaScript.
3. Edit the PhoneGap plist file with a new plug-in entry.
4. Use the plug-in from JavaScript.

Adding a Custom Plug-in to the iOS Hybrid Web Container

An example plug-in class that allows access to the iOS network activity monitor is available in HybridWebContainer/Classes/Plugins.

1. Copy the `networkActivityMonitor.h` and `networkActivityMonitor.m` files from HybridWebContainer/Classes/Plugins to the HWC.xcodeproj project.
2. Add the `networkActivityMonitor.js` to the Hybrid App /html/js/ directory that corresponds with the Eclipse project that generated the Hybrid App files.
3. Modify your JavaScript for any event desired to call the new plug-in.

Here is an example that reacts to a menu item and uses a global variable to toggle the activity indicator on and off.:

```
var gActivityIndicator = true; // global variable

function customAfterMenuItemClick(screen, menuItem) {
    if (screen === "Start" && menuItem === "networkActivityIndicator") {
        {
            window.plugins.networkActivityIndicator.set( gActivityIndicator,
                aiSuccess, aiFail );
            // Toggle the network activity indicator each time plugin is
            selected
            if ( gActivityIndicator )
                gActivityIndicator = false;
            else
                gActivityIndicator = true;
            return false;
        }
    }

    function aiSuccess() {
        alert("Successfully enabled activity indicator");
    }

    function aiFail() {
```

```
alert("Failed to enable activity indicator");  
}
```

4. Add a plug-in entry to `Cordova.plist`:

```
Key: networkActivityIndicator  
Type: String  
Value: networkActivityIndicator
```

5. Generate the Hybrid App files and deploy the package to the server..
6. Test the event in the JavaScript file that is hooked into the new plug-in.

If the plug-in requires additional resources, such as images or other files, these should be added to the project under the `Resources` group folder. For example, the `ChildBrowser` plug-in available at github.com contains icons that are stored in a file called `ChildBrowser.bundle`. In this example, the `ChildBrowser.bundle` should be added to the `Resources` group folder in the project in Xcode.

Some plug-ins also require files to be in a `www/` directory. The `notification.beeper` API is one example. If this is the case, add the resources to the `www` directory that is referenced by the project under the `Resources` group folder as described in Step 7 in *Upgrading the PhoneGap Library used by the iOS Hybrid Web Container*.

Custom Plug-ins for the Windows Mobile Hybrid Web Container

Integrate PhoneGap plug-ins with the Windows Mobile Hybrid Web Container.

In general, adding a custom plug-in to Hybrid Web Container is identical to adding a plug-in to any PhoneGap application. The basic steps include:

1. Implement the plug-in class that extends the class "PluginBase."
2. Implement the PhoneGap plug-in JavaScript.
3. Add the plug-in class to the plug-in configuration file.
4. Use the plug-in from JavaScript.

Adding a Custom Plug-in to the Windows Mobile Hybrid Web Container

This procedure shows an example of adding a plug-in class that allows access to the Windows Mobile calculator.

The plug-in class is available under the `TPTools\phoneGap\wm` directory. To include this plug-in in the Hybrid Web Container, follow these steps:

1. Add a new class called `Calculator` into the folder `CustomCode` and implement the code:

```
using WMGapClassLib.Cordova;  
namespace Sybase.Hwc.CustomCode  
{  
    public class Calculator : PluginBase  
    {  
        public void sum(Session session,  
            Sybase.HybridApp.Util.Json.JsonObject arguments)  
        {  
            try
```

```

        {
            double x = 0;
            double y = 0;
            x = double.Parse(arguments.GetString("x"));
            y = double.Parse(arguments.GetString("y"));
            this.DispatchCommandResult(session, new
PluginResult(PluginResult.Status.OK, x + y));
        }
        catch (System.Exception ex)
        {
            this.DispatchCommandResult(session, new
PluginResult(PluginResult.Status.ERROR, ex.Message));
        }
    }
}

```

2. Open the file `Plugins.xml`, which is located in the `HybridWebContainer` project, and add the custom plug-in:

```

<?xml version="1.0" encoding="utf-8" ?>
<plugins>
    <plugin id="showcertpicker"
        class="Sybase.Hwc.CertificationPickerPlugin"/>
    <plugin id="Calculator"
        class="Sybase.Hwc.CustomCode.Calculator"/>
</plugins>

```

3. Open the `Custom.js` file for editing and add this method:

```

function calculateSum(x, y, successCb, errorCallback){
cordova.require('cordova/exec') (
    successCb,
    errorCallback,
    "Calculator", "sum",
    { x: document.getElementById('x').value, y:
document.getElementById('y').value });
}

```

4. Call this JavaScript method somewhere else to get the result:

```

function doCalculateSum() {
    calculateSum(

        document.getElementById('x').value,

        document.getElementById('y').value,
        function (res){
            document.getElementById('res').innerHTML = res;
        },
        function (e) {

            console.log("Error occurred: " + e);

            document.getElementById('res').innerHTML = "Error
occurred: " + e;
        });
};

```

5. Generate and deploy the application and test the event in the `custom.js` file that is hooked into the new plug-in.

Removing PhoneGap From the Hybrid Web Container

If PhoneGap functionality is not required, you can make a few modifications to remove all references to the PhoneGap library that is linked to the Hybrid Web Container.

Removing PhoneGap from the Android Hybrid Web Container

Remove all references to the PhoneGap library that is linked to the Android Hybrid Web Container.

Leaving PhoneGap in place does not cause any issues, but does increase overall application size by about 70KB.

1. Open the `UiHybridAppContainer.java` file for editing and comment out this line:

```
//import org.apache.cordova.*;
```

2. Change the superclass of `UiWorkflowContainer` from `Droidgap` to `Activity`:

```
public class UiWorkflowContainer extends Activity {
```

3. Around line 91, change the `USE_PHONEGAP` variable to false, so the line of code looks like this:

```
private static final boolean USE_PHONEGAP = false;
```

4. At this point, there are 5 errors, which are caused by calling methods that were inherited from the `Droidgap` class (but do not exist in the `Activity` class); comment out the 5 lines that cause these errors :

- a) To find these lines, search for "USE_PHONEGAP."

These lines are all contained in "if (USE_PHONEGAP)" statements.

- b) Around line 110, comment out:

```
// super.init();  
// m_oWebView = this.appView;
```

- c) Around line 262, comment out:

```
// super.setStringProperty( "loadingDialog", m_sProgressText );  
// super.setIntegerProperty( "loadUrlTimeoutValue",  
300000 );  
// super.loadUrlWithData( sBaseURL, abData );
```

5. Switch to the Java perspective, right-click on the **HybridWebContainer** project, and choose **Properties**.
6. Under Java Build Path, click the **Libraries** tab.
7. Remove the PhoneGap library (`phonegap<version>.jar-HybridWebContainer/libs`).
8. Delete the `phonegap<version>.jar` file from the `HybridWebContainer\libs` folder.

Removing PhoneGap from the BlackBerry Hybrid Web Container

Remove all references to the PhoneGap library that is linked to the BlackBerry Hybrid Web Container.

Leaving PhoneGap in place does not cause any issues, but does increase overall application size by about 500KB.

1. Open the `AmpBrowserScreen.java` file for editing and comment out this line:

```
//m_browserField.addListener( new HWCBrowserFieldListener(new
HWCWidgetConfigImpl(), m_browserField) );
```

2. Right-click the **HybridWebContainer** project and choose **Properties..**
3. Under Java Build Path, click the **Libraries** tab.
4. Remove these libraries:
 - PhoneGapExtension.jar
 - WebWorksCommon.jar
 - WebWorksExtension.jar
 - WebWorksFramework.jar
5. Delete the same jar files from the `libs` folder.
6. Delete the `xml` folder, which hosts `plugins.xml`.

Removing PhoneGap from the iOS Hybrid Web Container

Remove all references to the PhoneGap library that is linked to the iOS Hybrid Web Container.

Leaving PhoneGap in place does not cause any issues, but does increase overall application size by about 400KB.

1. In Xcode, open the `HWCAppDelegate.h` file and comment out this line:


```
#define USE_PHONEGAP 1
```
2. In the Build Settings tab, for the Hybrid App project under **Other Linker Flags**, remove `libPhoneGap.a` for all build configurations.
3. Under **Warning Linker Flags** remove `libPhoneGap.a` for all build configurations.
4. In Project Navigator, remove references to these files:
 - VERSION
 - PhoneGap.plist
5. In Xcode, in Project Navigator, remove the reference to the `www` directory.
6. In Xcode, in Project Navigator, remove the reference to the `Capture.bundle` directory.
7. Clean and rebuild the project for all configurations.

Initializing the PhoneGap Library for the Windows Mobile Hybrid Web Container

You must initialize the PhoneGap library before using it.

1. Open the HTML file for the Hybrid App for editing.
2. Add this code.

```
<Html>
<script>
Function onLoad() {
    try
    {
        cordova.require('cordova/
channel').onDOMContentLoaded.fire();

        cordova.require('cordova/
channel').onNativeReady.fire();
        _nativeReady = true;
    }
    catch (e)
    {
        alert("Initialize
phonegap error:" + e.message);
    }
}
</script>
<body onload="onLoad();" >
</html>
```

3. Save the file.
4. Regenerate the Hybrid App package.

PhoneGap Library Downgrade

SAP Mobile Platform included PhoneGap 1.4.1 libraries embedded inside the iOS and Android Hybrid Web Containers.

SAP Mobile Platform 2.2 includes the Cordova 2.0 libraries for Android, BlackBerry, iOS, and Windows Mobile. When PhoneGap changed to the Cordova name in 1.5, interfaces to native PhoneGap plug-ins were renamed, thus, 2.1.3 Hybrid Apps that use the PhoneGap 1.4.1 will not work with 2.2 Hybrid Web Container. If you want to continue to use the PhoneGap 1.4.1 libraries with the 2.2 Android and iOS Hybrid Web Containers, you can revert from the Cordova 2.0 libraries to the PhoneGap 1.4.1 libraries.

Downgrading the PhoneGap Library Used by the Android Hybrid Web Container

Change from the Cordova 2.0 library included with the Android Hybrid Web Container to the PhoneGap 1.4.1 library.

The files referenced in this procedure are located in the *Android_PhoneGap_Downgrade.zip* file.

1. Use a diff utility tool to compare the file `UiHybridAppContainer_before.java` and `UiHybridAppContainer_after.java` files.
2. Open the `UiHybridAppContainer.java` file, which is located in `..HybridWebContainer\src\com\sybase\hwc`, and apply the changes found with the diff utility tool.

Note: Keep in mind that this change could remove bug fixes, or cause unexpected behavior of the related new features.

3. Rebuild the Hybrid Web Container project to make sure there are no compilation errors.
4. Replace the `cordova-2.0.0.jar` located in `<SMP_HOME>\UnwiredPlatform\MobileSDK22\HybridApp\API\Container\android`, with the `phonegap-1.4.1.jar` file, which is in the *Android_PhoneGap_Downgrade.zip* file.
5. In the HybridWebContainer project, remove the `res/xml/config.xml` file and add the `plugins.xml` and `phonegap.xml` files.
6. Open the `UiHybridAppContainer.java` file for editing and change the import statement from `import org.apache.cordova.DroidGap` to `import com.phonegap.DroidGap`.
7. Find the method:

```
@Override
    public void onCreate( Bundle savedInstanceState ) {
        super.setBooleanProperty("showTitle", true );
        super.onCreate( savedInstanceState );
    }
```

Remove the line: `super.setBooleanProperty("showTitle", true);`.

8. Rebuild the HybridWebContainer project.
9. (Optional) Rename the `phonegap-1.4.1.js` file to `phonegap-1.4.1.javascript`.
10. (Optional) In the Container folder of generated applications, replace the `android/cordova-2.0.0.javascript` with `phonegap-1.4.1.javascript`.

11. (Optional) In the `API.js` file, remove the string `android/cordova-2.0.0.javascript` and replace it with `android/phonegap-1.4.1.javascript`.

Downgrading the PhoneGap Library Used by the iOS Hybrid Web Container

Change from the Cordova 2.0 library included with the iOS Hybrid Web Container to the PhoneGap 1.4.1 library.

1. Unzip *PhoneGapLib.zip*.

This unzips to a directory named `PhoneGapLib`.

2. Copy `PhoneGapLib` inside the `HybridWebContainer` directory, which is located in `\SMP_HOME\UnwiredPlatform\MobileSDK\version\HybridApp\Containers\iOS\`.
3. In Xcode, in the `HybridWebContainer` directory, open the `HWC.xcodeproj`.
4. Under the `Resources` group folder, remove `VERSION`, `Capture.bundle`, `www`, and `Cordova.plist`, and replace them with equivalent files, from `HybridWebContainer/PhoneGapLib`.

Note: The `Cordova.plist` file will be replaced by the `PhoneGap.plist` file.

5. (Optional) Delete the `HybridWebContainer/CordovaLib` directory and `libCordova.a` in each of the `HybridWebContainer/libs/<configuration>` directories.
6. Open the `PhoneGapLib.xcodeproj`, which is located in `HybridWebContainer/PhoneGapLib`.
7. Build all four configurations of the `PhoneGapLib` target, including `Release-iphoneos`, `Debug-iphoneos`, `Release-iphonesimulator`, and `Debug-iphonesimulator`.
8. Copy the `libPhoneGap.a` file from each configuration build directory to the corresponding directory in `HybridWebContainer/libs/<configuration>`.
9. Close the `PhoneGapLib` Xcode project.
10. Go back to `HWC.xcodeproj`, open the project settings, and go to the `Build Settings` tab.
 - a) Under **Other Linker Flags**, change all instances of `"libCordova.a"` to `"libPhoneGap.a"`.
 - b) Under **Header Search Paths**, change all instances of `"CordovaLib"` to `"PhoneGapLib"`.
11. Perform two global search-and-replace operations in Xcode and replace:
 - `USE_CORDOVA` with `USE_PHONEGAP`
 - `CORDOVA_FRAMEWORK` with `PHONEGAP_FRAMEWORK`
12. Open the `HWCAAppDelegate.h` file, which is in the `Classes` group folder.

- a) In the `#ifdef USE_PHONEGAP` block at the top of the file, change the import statements so they are importing `PhoneGapDelegate.h` instead of `CDVViewController.h`.

The code should look like this:

```
#ifdef USE_PHONEGAP
#ifdef PHONEGAP_FRAMEWORK
#import <PhoneGap/PhoneGapDelegate.h>
#else
#import "PhoneGapDelegate.h"
#endif
```

- b) In the interface definition for `HWCAppDelegate`, change the superclass in the `#ifdef USE_PHONEGAP` block from `CDVViewController` to `PhoneGapDelegate`.

13. Open the `HWCAppDelegate.m` file, which is located in the `Classes` group folder, and in each of the following functions, add the specified code at the end of the function, just before it returns.

- a) Just before the function

`application:didFinishLaunchingWithOptions:` returns, add:

```
#ifdef USE_PHONEGAP

if ( [super
respondToSelector:@selector(application:didFinishLaunchingWithOptions:)] )

[super application:[UIApplication sharedApplication]
didFinishLaunchingWithOptions:launchOptions];

#endif
```

- b) Just before the function **`applicationDidBecomeActive:`** returns, add:

```
#ifdef USE_PHONEGAP

if ( [super
respondToSelector:@selector(applicationDidBecomeActive:)] )

[super applicationDidBecomeActive:application];

#endif
```

- c) Just before the function **`applicationWillResignActive:`** returns, add:

```
#ifdef USE_PHONEGAP

if ( [super
respondToSelector:@selector(applicationWillResignActive:)] )

[super applicationWillResignActive:application];

#endif
```

- d) Just before the function **`applicationWillTerminate:`** returns, add:

```
#ifdef USE_PHONEGAP
```

```
if ( [super  
respondsToSelector:@selector(applicationWillTerminate:)] )  
  
[super applicationWillTerminate:application];  
  
#endif
```

- 14.** In the `applicationWillEnterForeground:` function, change the call to the superclass method `onAppWillEnterForeground:` to a call to the superclass method `applicationWillEnterForeground:`.

```
if ( [super  
respondsToSelector:@selector(applicationWillEnterForeground:)] )  
  
[super applicationWillEnterForeground:application];
```

- 15.** In the `applicationDidEnterBackground:` function, change the call to the superclass method `onAppDidEnterBackGround:` to a call to the superclass method `applicationDidEnterBackground:`.

```
if ( [super  
respondsToSelector:@selector(applicationDidEnterBackground:)] )  
  
[super applicationDidEnterBackground:application];
```

- 16.** Save the file.

- 17.** Rebuild the `HWC.xcodeproj` project.

- a) From the Xcode menu, select **Product > Clean**.
- b) Select **Product > Build**.

Hybrid App Configuration for Data Change Notification

This section contains details about developing Hybrid Apps that take advantage of DCN updates.

Hybrid Apps require a server-initiated starting point and defined matching rules, which allows SAP Mobile Server to push changes to Hybrid App clients. See the topics *Starting Points* and *Adding Matching Rules* in *SAP Mobile WorkSpace - Hybrid App Package Development*.

Extending Data Change Notification to Hybrid Apps

Data change notification (WF-DCN) requests allow SAP Mobile Server to process the DCN request and send notification to the device of that data change.

Depending on the cache policy used by the affected MBO, once the application receives notification, it can retrieve data directly from the EIS or from the SAP Mobile Server cache, keeping the application synchronized. DCN messages targeted for MBOs used in applications (WF-DCN), uses similar syntax as general DCN, with these differences:

- The value of **cmd** is *wf* for WF-DCN requests, compared to *dcn* for regular DCN.
- The message contains the fields required for notification, such as the to address, from address, e-mail subject, and e-mail body.
- The WF-DCN message is captured and parsed by the server-initiated Hybrid App, which processes the WF-DCN message differently, depending on the message type: with payload or without payload.

WF-DCN format

The WF-DCN request is a JSON string consisting of these fields: engine converts MBO data and WF-DCN messages into email, and pushes it to device's inbox

1. Operation name(op) **:upsert** or **:delete**— same as regular DCN.
2. Message ID (id) of the Hybrid App – used for correlation (a **:delete** for a previously submitted request with **:upsert** is possible)
3. Username (to) – the SAP Mobile Platform user name. For the user to be recognized by WF-DCN, the device user should first have established communication using the activation mechanism in SAP Control Center.

Note: The "To" field must match the SAP Mobile Platform user name—not the user name used to register the device.

4. Subject (subject) – subject of the Hybrid App message.
5. Originator <from> – who the Hybrid App message is from.

6. Body of the Hybrid App message <body> – it can embed customized information.
7. <received> – received time of the Hybrid App message.
8. <read> – whether the Hybrid App message is read.
9. <priority> – whether the Hybrid App message has a high priority.
10. List of dcn request <data> – JSON format string.

Example DCN request in JSON format:

```
{
  "op":":upsert",
  "id":"WID123",
  "to":"SUPAdmin",
  "subject":"Trip request approval required",
  "from":"user321",
  "body":"This is a message just used to do a test",
  "received":"2009-03-29T10:07:45+05:00",
  "read":false,
  "priority":true,
  "data":
  [
    {
      "id": "1",
      "<general dcn request>"
    }
    ...
    {
      "id": "4",
      "<general dcn request>"
    }
  ]
}
```

Hybrid App DCN request flow

WF-DCN with and without payload differ slightly, but the general flow is similar for each. When the WF- DCN request is received, SAP Mobile Server gets the **wf** cmd value from the request first, and:

1. SAP Mobile Server invokes `preProcessFilter` if the DCN filter is specified.
2. SAP Mobile Server receives a raw HTTP POST body to generate and return a WF- DCN request message object.
3. The JSON format string is parsed into a WF-DCN request object.
4. The DCN request in the Hybrid App message object is parsed and those within the scope of a single transaction per DCN request object in the array are executed. Results are recorded for a report after completing the WF-DCN request.
5. From the CDB, the server looks up all users assigned to the indicated Hybrid App package in the “to” attribute of the Hybrid App message, then matches them with the receiver list. For every receiver, SAP Mobile Server generates multiple Hybrid App messages (all Hybrid App messages are created within one transaction), one per device identified (one user might have multiple devices), and then sends them to the JMS queues. The lookup of the logical id is performed by combining the username in the “to” list to the “securityProfile” specified in the HTTP POST REQUEST URL parameter list.

6. If any errors occur in step four, step five does not execute. If any errors occur in step five, step five is not committed. If any errors occur in either of those steps, an HTTP 500 error is returned.
7. SAP Mobile Server invokes the `postProcessFilter`, if specified.
8. If no errors occur, SAP Mobile Server returns success to the caller HTTP 200 with the body of the JSON string (or any opaque data returned from the `postProcessFilter`) of the WF-DCN Result. Otherwise, SAP Mobile Server returns an HTTP 500 error with the body of the JSON log records.

Non HTTP Authentication Hybrid App DCN Request

You can send Hybrid App DCN requests that are not authenticated.

The URL is:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=admin&domain=default&username=supAdmin&password=sup
Pwd&dcn_filter=aa.bb&dcn_request=<wfrequestdata>
```

where *supAdmin* represents the SAP Mobile Server Administrator, and *supPwd* represents the Administrator's password defined during SAP Mobile Platform installation.

Sending Hybrid App DCN to Users Regardless of Individual Security Configurations

You can send Hybrid App DCN requests to users in other security configurations if you belong to the default security configuration.

If the Hybrid App DCN sender is authenticated against the default admin security configuration, they are automatically authorized to push data to all users regardless of their individual security configuration. If not, the sender can only push to users within the same security configuration.

For example, in the case of a non HTTP authentication request, this request is authorized to push data to users in other security configurations since the sender *supAdmin*, belongs to the admin security configuration:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=othersecurity&domain=default
&username=supAdmin@admin&password=supPwd&dcn_filter=aa.bb&dcn_reque
st=<request>
```

And this request is denied because *supAdmin@mysecurityconfig* can only push data to users in the same security configuration:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=othersecurity&domain=default
```

```
&username=supAdmin@mysecurityconfig&password=supPwd&dcn_filter=aa.bb&dcn_request=<request>
```

Hybrid App DCN Request Response

After processing of the Hybrid App DCN request, SAP Mobile Server sends the response to notify the caller whether the request was processed successfully.

The response includes two parts:

1. The result of processing the Hybrid App request.
2. The result of processing the general DCN requests.

The response is also in a JSON format string:

```
{
<wf dcn result>
"result":
[
  {
    <general dcn result>
  },
  {
    <general dcn result>
  }
]
}
```

An example response is:

```
{
  "id": "1",
  "success": false,
  "statusMessage": "there is error in processing dcn",
  "result":
  [
    {
      "id": "1",
      "success": true,
      "statusMessage": ""
    },
    {
      "id": "2",
      "success": false,
      "statusMessage": "bad msg2"
    }
  ]
}
```


Hybrid App DCN Design Approach and Sample Code

Understand the design approach for both WF-DCN with and without payload, and view samples for each approach.

Note: Samples are for illustrative purposes only and should not be used as a guide for developing your DCN requests.

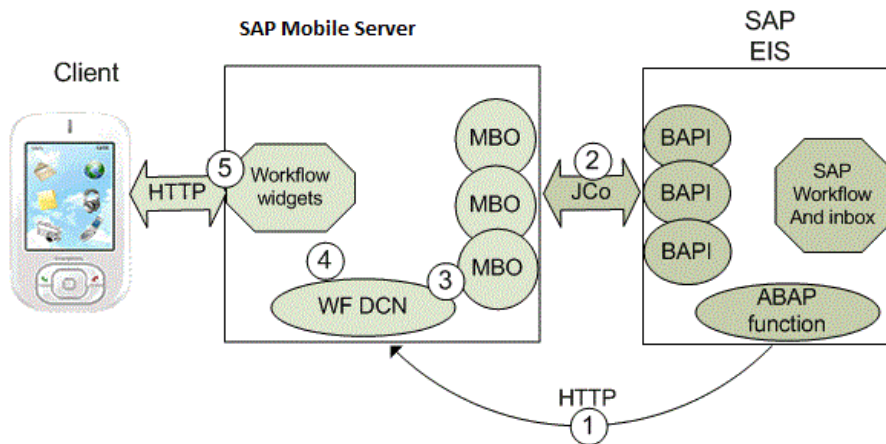
Comparing Hybrid App DCN With and Without Payload

This section compares the two types of WF-DCN and includes examples of each.

Hybrid App DCN Without Payload

Understand how to construct a Hybrid App DCN without payload message.

This example illustrates data flow of a WF-DCN without payload using an SAP® EIS:



1. The WF-DCN pushes new messages (workitems) to SAP Mobile Server, which are then delivered to the device, for example, a Hybrid App notification.
2. After the EIS sends a workitem id to SAP Mobile Server, SAP Mobile Server uses workitem MBO and workitem id to retrieve details of the workitem from the EIS.
3. After SAP Mobile Server receives the message, a matching Hybrid App server starting point parses the message and extracts data fields from the message, including data into the parameter of an MBO object query operation.

4. Since the MBO uses an online cache policy, the object query is mapped to a load operation, allowing the data to be passed into the load operation as a load argument to trigger an MBO data refresh.
5. The Hybrid App engine converts MBO data and the WF-DCN message into a notification, and pushes it to the device's mobile inbox.

MBO cache group policy

The cache group policy of MBOs used in the WF-DCN without payload must be online. The online MBO contains the `findByParameter` object query with the same parameters defined in the load operation. The query is triggered by the Hybrid App server-initiated starting point after extracting the parameter values from the WF-DCN message body.

Message format

The message format of the WF-DCN message without payload is:

```
{ "id": "", "op": "", "subject": "", "to": "", "from": "", "read":, "priority":  
  "", "body": "",  
  "data": {} }
```

For example:

```
{ "id": "", "op": ":upsert", "subject": "test", "to": "test", "from": "test",  
  "read":,  
  "priority": "", "body": "MATCH:SUP_MWF,TaskID:TS97200149, WIID:  
  1470577,  
  USER:PERF0111*#END#*", "data": {} }
```

SAP Mobile Server extracts information from the DCN message and retrieves details from the EIS.

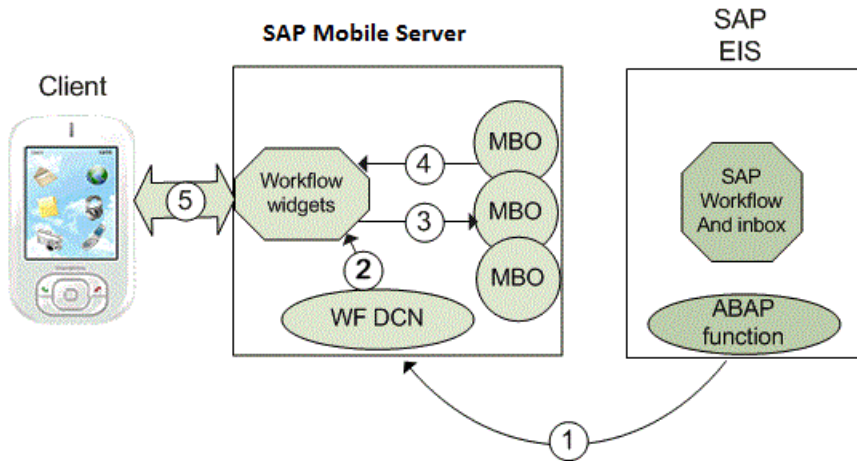
Processing the WF-DCN without payload message

After SAP Mobile Server receives the message, a matching Hybrid App server-initiated starting point parses the message and extracts data fields from the message. The server-initiated starting point sets extracted data into the parameter of an object query operation. Since the MBO used by the without payload message uses an online cache policy, the object query is mapped to a load operation. The data is passed into the load operation as a load argument to trigger MBO data refresh.

Hybrid App DCN With Payload

Understand how to construct a Hybrid App DCN with payload message.

This example illustrates data flow of a WF-DCN with payload using an SAP EIS:



1. When the EIS has new or modified data to push to SAP Mobile Server, it initiates an HTTP request to the WF-DCN URL. The WF-DCN message contains the new or changed data object.
2. When the WF-DCN message reaches SAP Mobile Server, the Hybrid App engine evaluates the matching rule against all registered Hybrid Apps. If a matching rule matches this message, the Hybrid App server starting point for that Hybrid App is triggered to process the message.
3. The data object included in the WF-DCN message is applied to the MBO CDB table by inserting new records or updating existing records.
4. The Hybrid App server-initiated starting point extracts parameter values from the message body and triggers the MBO object query to retrieve the newly inserted or updated record.
5. The Hybrid App engine converts the MBO data and WF-DCN message into a Hybrid App notification, then pushes it to the device mobile inbox using SAP messaging (MOCA).

MBO cache group policy

The cache group policy of MBOs used in WF-DCN with payload must be DCN.

Message format

The message format of the WF-DCN message with payload is:

```
{
  "id": "", "op": "", "subject": "", "to": "", "from": "", "read": "", "priority": "", "body": "",
  "data": [
    {
      "id": "", "pkg": "Package", "messages": [
        {
          "id": "2", "mbo": "MBO", "op": "upsert",
          "cols": {
            "attribute1": "value1", "attribute2": "value2", "attribute3": "value3"
          }
        }
      ]
    }
  ]
}
```

The message must contain e-mail information: subject, to, from, and so on, and include the MBO package name and version, MBO name, attribute name, and attribute value. The message can include multiple MBOs. For example:

```
{"id":"1137","op":":upsert","subject":"PERF0111's Leave Request",
"to":"PERF0111","from":"Leave Work
Flow","read":"false","priority":"true",
"body":"MATCH:SUP_MWF,TaskID:TS97200149, WIID:1470577,
USER:PERF0111#END#",
"data":[{"id":"dcbtest","pkg":"sup_mwf:1.2","messages":
[{"id":"2","mbo":"Workitem",
"op":":upsert","cols":
{"WORKITEM":"1470577","USERNAME":"perf0111","DESCRIPTION":"cc",
"DECISION":"test"}},{id":"6","mbo":"Alternatives","op":":upsert",
"cols":
{"WORKITEM":"1470577","USERNAME":"perf0111","PKEY":"01","PVALUE":"A
p"}]}]}
```

Sample Java Function for Generating Hybrid App DCN

This WF-DCN sample illustrates WF-DCN without payload.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;
import java.net.Authenticator;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.PasswordAuthentication;
import java.net.ProtocolException;
import java.net.URL;
import java.net.URLEncoder;

public class HttpAuth
{
    /**
     * @param args
     * @throws MalformedURLException
     */
    public static void main(String[] args) throws Exception
    {
        URL url = null;

        String wfcdn_request = "{\"id\":\"dcbtest_69\",\"op\":":
        \":upsert\",",
            + "\"subject\":\"dept_id = 1300\", \"to\":\"perf0111\",",
            + "\"from\":\"SAP Leave WorkFlow\", \"read\":false,",
        "\"priority\":true,",
            + "\"body\":\"\",TaskID:, WIID:000001468382,",
        "USER:perf0111#END#\"}";

        url = new URL("HTTP", "10.42.39.149", 8000,
```

```

        "/dcn/HttpAuthDCNServlet?
cmd=wf&security=admin&domain=default");

        HttpURLConnection con = null;

        con = (HttpURLConnection) url.openConnection();

        con.setDoOutput(true);
        con.setRequestMethod("POST");

        final String login = "supAdmin";
        final String pwd = "AdminPassword";
        Authenticator.setDefault(new Authenticator()
        {
            protected PasswordAuthentication
getPasswordAuthentication()
            {
                return new PasswordAuthentication(login,
pwd.toCharArray());
            }
        });

        StringBuffer sb = new StringBuffer();
        sb.append(wfdcn_request);
        OutputStream os = con.getOutputStream();
        os.write(sb.toString().getBytes());
        os.flush();
        os.close();

        StringBuffer xmlResponse = new StringBuffer();

        int returnCode = con.getResponseCode();
        if (returnCode != 200)
        {
            String rspErrorMsg = "Error getting response from the
server (error code "
                + returnCode + ")" + con.getResponseMessage();
            System.out.println(rspErrorMsg);
        }
        else
        {
            BufferedReader in = new BufferedReader(new
InputStreamReader(con
                .getInputStream(), "UTF-8"));
            String line;
            while ((line = in.readLine()) != null)
            {
                xmlResponse.append(line).append("\n");
            }
            System.out.println("xmlResponse: " + xmlResponse);
        }
    }
}

```


Index

.p12 certificates 223

A

ActiveSync, installing and configuring 215
 Advanced Encryption Standard 168
 AES
 See also Advanced Encryption Standard
 AES-128 170
 AES-256 166
 Afaria® Security Manager 171
 Alert Message property 230
 Alerts property 230
 Android emulator
 configuring 207
 Android Hybrid Web Container
 installing 207
 Android Hybrid Web Container customization
 setting HTTP headers 247
 ANDROID_CUSTOMIZATION_POINT_CATEG
 ORIZEDVIEWS 252
 ANDROID_CUSTOMIZATION_POINT_HYBRI
 DAPPSEARCH 266
 API functions
 credential functions 133
 general utility functions 133
 Hybrid App native device functions 133
 Hybrid App UI functions 133
 Hybrid App utility functions 133
 Hybrid App validation functions 133
 message data functions 133
 API.js 133
 APNS 227
 APNS Device Token property 230
 Apple push notification properties 230
 Apple push notification, configuring 229
 application 191
 application ID
 guidelines 192
 arbitrary metadata 80
 AttachmentViewer control
 image limitations 76

B

Badges property 230

BlackBerry 211
 configuring the simulator 211, 212
 BlackBerry Desktop Manager 210
 BlackBerry Hybrid Web Container 211
 adding a new language 275
 adding a splash screen 273
 BLACKBERRY_CUSTOMIZATION_POIN
 T_SPLASHSCREEN 273
 default behavior customization 281
 setting HTTP headers 284
 using custom colors 276

C

cached data lookup pattern
 data flow diagram 116
 overview 116
 Callbacks.js 138
 CallbackSet 138
 categorized views 252
 certificate picker 157
 Certificate.js 141
 certificates
 for context variables 195
 ClientIconIndex 80
 conditional navigation 151
 conditional start 153
 connection settings
 configuring 218
 default 304
 device 218
 Hybrid Web Container 218
 content security 166
 Android 166
 BlackBerry 168
 iOS 170
 Windows Mobile 171
 content type preference, changing 174
 context variables 80, 196
 configuring 195
 credential functions 133
 credentials, static and dynamic 156
 CredentialsCache 80
 Custom.js 125
 custom.js file
 methods 142

Index

- customAfterNavigateForward 142
- customAfterReportErrorFromNative 145
- customAfterShowScreen 142
- customAfterSubmit 142
- customAfterWorkflowLoad 142
- customBeforeMenuItemActivate 142
- customBeforeNavigateBackward 142
- customBeforeNavigateForward 142
- customBeforeReportErrorFromNative 145
- customBeforeShowScreen 142
- customBeforeSubmit 142
- customBeforeWorkflowLoad 142
- customization touch points
 - ANDROID_CUSTOMIZATION_POINT_DE
 - FAULTSETTINGS 244
- customValidateScreen 142

D

- data change notification 359
 - GET 357
 - JSON format 357
 - POST 357
 - request response 360
- Datajs 43
- datajs library 64
- Datajs library 43
- DCN 360
- debugging 184
- default locale, creating 175
- defining an MBO
 - for cached data lookup 117
 - for real-time data lookup 107
- DeleteProcessedMessages 80
- Delivery Threshold property 230
- deploy 103
- device platforms 103
- device users
 - assigning Hybrid App packages 194
- devices
 - Apple push notification properties 230
- documentation roadmap 1
- Dynamic authentication 158

E

- editing
 - locale properties file 178
- Enable property 230

- encoding type, changing 174
- encryption key length 168
- encryption policy 148

F

- file association 174
- filtering 308
- findByParameter
 - binding to a menu item 107
- findByParameter object query 111
- functions
 - resource 146
 - workflow UI 134

G

- general application properties 191
- general utility functions 133
- generated files 126, 127
- getCurrentMessageValueCollection() 136
- getDataMessage() 136
- getPicture 138

H

- hard coded credentials 196
- HTML format 79
- HWC.xcodeproj 298
- Hybrid App
 - prepackaged, BlackBerry 323
- Hybrid App client
 - using credentials 160
- Hybrid App clients
 - and static SSO2 tokens 164
 - and static X.509 certificates 161
 - using credentials in 160
 - using SSO2 tokens in 162
- Hybrid App native device functions 133
- Hybrid App package
 - generated files 126
- Hybrid App Package Generation Wizard 103
- Hybrid App packages
 - assigning device users 194
 - configuring notification mailbox 193
 - deploying 101
- Hybrid App UI functions 133
- Hybrid App utility functions 133
- Hybrid App validation functions 133

Hybrid Web Container

- Android 244
- ANDROID_CUSTOMIZATION_POINT_DE
FAULTSETTINGS 244
- architecture 3
- building using source code 204
- customization 3, 244
- default values for settings screen 244
- installing from App Store 213
- management 3
- offline capabilities 3
- removing 231
- settings screen 244
- settings screen, default values 244

hybridapp_Custom.html 127

hybridapp_Custom.xml 100

hybridapp_JQM.html 127

hybridapp_JQM.xml 100

hybridapp.html 128

hybridapp.html generated file 127

HybridApp.js 24

HybridWebContainer.cod 211, 212

I

image

- limitations in Hybrid App messages 76

installing 213

internationalization

- Hybrid App Designer 179
- on the device 181

InvokeOnClient 80

iOS 212

iOS Hybrid Web Container

- customizations 300
- settings screen 306

iOS Hybrid Web Container customization 302

filtering 308

setting HTTP headers 310

sorting 308

iOS push notification properties 230

IOS_CUSTOMIZATION_POINT 298

iPad

hiding the listview 312

ISO-8859-1 encoding 174

iTunes 214

J

jquery.mobile-1.1.0.css 128

L

load arguments 107

locale

- editing 178
- properties file 175, 178

localization 125, 172

creating a new locale 175

Hybrid App package 173

limitations 173

task flow 173

updating the current locale 178

look and feel 100

look and feel files 128

M

manage 189

manifest.xml 80

MarkProcessedMessages 80

master.css 128

matching rules

specifying 113

message data functions 133

Microsoft ActiveSync, installing and configuring
215

ModuleDesc 80

ModuleDisplayName 80

ModuleName 80

ModuleVersion 80

N

native device functions 135

network edge authentication 159

non HTTP authentication request 359

non-ASCII encoding 174

notification mailbox 193

notifications

creating 39

null value support 72

O

object queries

binding to a menu item 118

object query parameters

defining a control that passes 119

Index

- OData 43
- offline capabilities 3
- Optimize for appearance look and feel 127
- Optimize for performance look and feel 127, 131
- OTA 211
- over the air 211

P

- performance agent 197, 198
- PersistAppDomain 80
- PersistentContent 168
- PersistentContentListener 168
- PersistentStore 168
- PhoneGap 328
 - initializing 352
 - removing 350
 - supported APIs 328
- PhoneGap plugin
 - testing 342
- PIN screens
 - CreatePasswordViewController.xib 303
 - customizing 303
 - EnterPasswordViewController.xib 303
 - iOS 303
- preferences
 - appearance 174
 - content types 174
 - general 174
- ProcessUpdates 80
- properties
 - push notification for iOS 230
- propagate to attributes 107
- PurchaseOrderSample 175
- push notification properties for iOS 230

Q

- query types
 - addallmenuitems 73
 - addMenuItem 73
 - alert 73
 - clearrequestcache 73
 - clearrequestcacheitem 73
 - close 73
 - downloadattachment 73
 - formredirect 73
 - loadtransformdata 73
 - logtoworkflow 73

- removeallmenuitems 73
- rmi 73
- setscreentitle 73
- showattachment 73
- showcertpicker 73
- showInBrowser 73
- showlocalattachment 73
- submit 73

R

- real-time lookup pattern
 - data flow diagram 106
 - overview 106
- RequiresActivation 80
- resource functions 146
- rmi.xml 184
- RSA algorithm 223

S

- SAP passport 197
- send a notification 183
- sending server notification to a device 115
- server notification pattern 111
 - creating an MBO for 111
 - data flow diagram 111
 - overview 111
- server-driven notification
 - creating 113
- server-initiated 39
- shared storage 150
- SharedStorage 150
- showErrorFromNative 145
- signing key 203
- single sign-on 159
 - using credentials 160
 - using SSO2 tokens 162
 - using static SSO2 tokens 164
 - using static X.509 certificates 161
- single sign-on task flow 357
- sorting 308
- Sounds property 230
- SQLite Encryption Extensions (AES-128) 170
- static authentication 157
- strings.xml 239
- stylesheet.css 128
- SupCertificateIssuer 197
- SupCertificateNotAfter 197

- SupCertificateNotBefore 197
- SupCertificateSubject 197
- SUPMessaging_Pro.cab 216
- SUPMobileHybridApp.replaceHybridAppCertificate() 165
- SupPassword 196
 - for context variables 195
- SUPStorage 148
- SUPStorage.js 147
- SupUser 196, 197
 - for context variables 195
- synchronization software 215

T

- task flow 7
- testing
 - X.509 certificates 223, 227
- touch point 298
- trace 197

U

- URL parameters 42

- UTF-8 encoding 174

V

- variables, context
 - configuring 195
- viewing Hybrid App messages
 - Android 183
 - BlackBerry 183
 - iOS 183
 - Windows Mobile 183

W

- WorkflowClient.xml 39, 89

X

- X.509 certificate 226

