## 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 ................................................ 64
  Enabling the Datajs Library on Windows Mobile ......... 64

**Hybrid Web Container and Hybrid App JavaScript APIs** ........................................................................... 65
  anonymous namespace ........................................ 68
Assigning and Unassigning a Hybrid App to an Application Connection ........................................... 722
Activating the Hybrid App ........................................... 722
Configuring Context Variables for Hybrid App Packages ................................................................ 723
Changing Hard Coded User Credentials ........... 724
Adding a Certificate File to the Hybrid App Package ........................................................................... 725
End to End Trace and Performance ......................... 725
Enabling the Performance Agent on the Device ........................................................................... 726
Tracing Application Connections ........................................... 726

Build a Customized Hybrid Web Container Using the Provided Source Code ................................................................. 729
Building the Android Hybrid Web Container Using the Provided Source Code ........................................... 729
Building the Android Hybrid Web Container Outside of Eclipse .......................................................... 730
Building the BlackBerry Hybrid Web Container Using the Provided Source Code ........................................... 731
Supplying a Signing Key ........................................... 731
Building the iOS Hybrid Web Container Using the Provided Source Code ........................................... 732
Building the Windows Mobile Hybrid Web Container Using the Provided Source Code ........................................... 733

Install and Configure the Hybrid Web Container On the Device ........................................................................... 735
Preparing Android Devices for the Hybrid Web Container ........................................................................... 735
Installing the Hybrid Web Container on Android Devices ........................................................................... 735
Configuring the Android Emulator ........................................... 735
Preparing BlackBerry Devices for the Hybrid Web Container ........................................................................... 738
Including a Prepackaged Hybrid App in the Windows Mobile Hybrid Web Container .................860
Adding Native Device Functionality to the Hybrid Web Container .............................................................864
Supported JavaScript PhoneGap APIs .................................................864
Implementing PhoneGap ..........................................................874
PhoneGap Custom Plug-ins .................................................875
Initializing the PhonePlug Library for the Windows Mobile Hybrid Web Container ........886
PhoneGap Library Downgrade .................................................886
Using the HTTPS Proxy Exposed by the PhoneGap Plugin ..........................................................888

**Hybrid App Configuration for Data Change Notification** .................................................................893
Extending Data Change Notification to Hybrid Apps ..............................................................893
Non HTTP Authentication Hybrid App DCN Request ............................................................895
Sending Hybrid App DCN to Users Regardless of Individual Security Configurations ..................896
Hybrid App DCN Request Response ..............................................................896
Hybrid App DCN Design Approach and Sample Code ........................................................................897
Comparing Hybrid App DCN With and Without Payload ..........................................................897
Sample Java Function for Generating Hybrid App DCN .............................................................900

Index ..................................................................................................................................................903
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.
5. *Generate the JavaScript API* on page 23.
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:
   a. On the Target Server page, select the server and connect to it.
   b. 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 an 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 Apps that participate 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:

```javascript
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 server 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 635 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
<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>

    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");
    }
</head>
```

Develop Hybrid Apps Using Third-party Web Frameworks

SAP Mobile Platform
Five of the included files are from `<SMP_HOME>\MobileSDK23\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>\MobileSDK23\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.

**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 639 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
<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>
  </head>
  hwc.processDataMessage = function (incomingDataMessageValue) {
    if (incomingDataMessageValue.indexOf("<M>") != 0) {
      alert("An error occurred! " + incomingDataMessageValue);
    }
    var workflowMessage = new WorkflowMessage(incomingDataMessageValue);
    var salesOrderList = workflowMessage.getData("Sales_order");
    var salesOrderId = salesOrderList.value[0];
  }
</html>
```
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>\MobileSDK23\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\MobileSDK23\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 (.*)]]>
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:

```xml
<Notifications>
  <Notification type="onEmailTriggered" targetScreen="Salesorder"
    asyncRequestErrorScreen="" errorNotificationSubjectLine=""
    errorNotificationFromLine="" asyncRequestErrorLogs=""
    asyncRequestErrorMessage=""/>
</Notifications>
```
14. Save and close the file.
15. In the Hybrid App Packaging Tool, click Generate to create a deployable package.
16. Login into 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 645 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
   <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
   ```
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
(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_atribKey").value;
    alert("The name of the first employee is " + firstName);
}
</script>
</head>
<body>
<form>Dept Id: <input type="text" value="100" id="deptID" /></form>
<button id="findByDeptButton" onclick="findByDept()">Find</button>&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>\MobileSDK23\HybridApp\PackagingTool.

5. Enter a location for the generated ZIP file, for example, c:\patterns.


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>\MobileSDK23\HybridApp\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 processDataMesssage, 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.
  - `Camera.js` – functions for accessing the device's native camera functionality.
  - `Timezone.js` – utility functions for getting the local time.
Develop Hybrid Apps Using Third-party Web Frameworks

- **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.

```javascript
/**
 * 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 { subAttr = new Object(); }
            }
            //...
```javascript
} 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:

```javascript
/**
 * Returns Department MBO structure.
 * Used by JavaScript functions of department_create_submit, department_create_onlineRequest, department_update_submit, department_update_onlineRequest, department_delete_submit, 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_update_submit, employee_update_onlineRequest, employee_delete_submit, employee_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 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 termination_date birth_date bene_health_ins bene_life_ins bene_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.
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.

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

var employeeValues = [];

for (var employeei = 0; employeei < departmentObj.Employee.length; employeei++) {
    var employeelc = new MessageValueCollection();
    employeelc.key = guid();
    employeelc.parent = "Department_employees";
    employeelc.parentValue = department_employees
    employeelc.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( employeelc ,employeekeys , employeetypes, 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_employeeValues = [];
    }
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].emp_id);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].manager_id);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].emp_fname);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].emp_lname);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].dept_id);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].street);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].city);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].state);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].zip_code);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].phone);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].status);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].ss_number);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].salary);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].start_date);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].termination_date);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].birth_date);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].bene_health_ins);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].bene_life_ins);
lueemployeei].bene_day_care);
oldValue_employeeValues.push( departmentObj.OldValue_Employee[oldValueemployeei].sex);
createMessageValues( employeelc,oldValue_employeekkeys , oldValue_employeetypes, oldValue_employeeValues );

    break;
    
}
// end of old values --->
    employeeValues.push( employeelc);
}

    department_employees.setValue(employeeValues);
    workFlowValues.add( department_employees.getKey(), department_employees);

}  
    hwc.doSubmitWorkflow_CONT( credInfo,
    workflowMessageToSend.serializeToString(),workflowMessageToSend.getHasFileMessageValue());
} /**
  * 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
<?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="" targetscreen="" errorscreen="">
        <Methods>
          <Method type="replay" mbo="Department"
            package="apiTesttDepartmentOneToMany: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" 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"/>
                </InputBinding>
              </Value sourceType="Key">
            </InputBinding>
          </Method>
        </Methods>
      </Action>
    </Actions>
  </Triggers>
</Workflow>
```
Develop Hybrid Apps Using Third-party Web Frameworks

```xml
 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"/>
   </InputBinding>
   <InputBinding optype="none">
     <Value sourceType="Key"
 workflowKey="Employee_emp_id_attribKey" attribName="emp_id" mboType="int"/>
   <Value sourceType="Key"
 workflowKey="Employee_manager_id_attribKey" attribName="manager_id" mboType="int"/>
   </Value sourceType="Key"
```
Develop Hybrid Apps Using Third-party Web Frameworks

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"/>
    </InputBinding>
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.

```xml
<Action name="Department_findByPrimaryKey" sourcescreen="" targetscreen="" errorscreen="">
  <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_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>
      </OutputBinding>
    </Method>
  </Methods>
</Action>
```
Develop Hybrid Apps Using Third-party Web Frameworks

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" targetscreen=" ">
  
  </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" targetscreen=" ">
    <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_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" targetscreen=" ">
    <Transformation>
    
    </Transformation>
  </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:

```xml
<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_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>
</Notifications>

Develop Hybrid Apps Using Third-party Web Frameworks
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:

```javascript
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:

```javascript
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:
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
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.

<table>
<thead>
<tr>
<th>URL parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loglevel</td>
<td>Current device log level.</td>
</tr>
<tr>
<td>screenToShow</td>
<td>Name of the screen to show.</td>
</tr>
<tr>
<td>supusername</td>
<td>User name of the current Hybrid App (if available).</td>
</tr>
<tr>
<td>lang</td>
<td>Current language of the device.</td>
</tr>
<tr>
<td>isalreadyprocessed</td>
<td>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.</td>
</tr>
<tr>
<td>loadtransformdata</td>
<td>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 <em>Calling the Hybrid Web Container</em>.</td>
</tr>
</tbody>
</table>
Develop Hybrid Apps Using Third-party Web Frameworks

<table>
<thead>
<tr>
<th>URL parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignoretransformscreen</td>
<td>Indicates that the JavaScript should ignore the RequestScreen 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.</td>
</tr>
</tbody>
</table>

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 proxy properties you require. For a complete list, see Proxy Properties in SAP Control Center for SAP Mobile Platform

Note:
- To access an external service, you must configure the http.proxyHost and http.proxyPort properties during server configuration in SAP Control Center > Configuration > General > Performance > Java Virtual Machine > Properties > JVM Properties > 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.
- Ensure that enough work processes exist in both the Gateway system and in any SAP EIS systems (for example, SAP ERP or CRM) to handle the peak load. To throttle the
number of connections used by SAP Mobile Platform, use the Pool Size property for your Proxy connection pool on each SAP Mobile Server node.

- 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 ID is also present in the connection pool. This proxy URL is used for request/response.

- The "Username" and "Password" fields of a Proxy Connection Profile are only valid when Anonymous access is used: "AllowAnonymousAccess" is set to True. If set to False, the end user must provide basic authentication credentials.

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](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.

```javascript
/**
 * 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
        // 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 ||
            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.statusText);
      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);

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];
                }
            } 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);
stridex = value.indexOf(' ');
value = value.substring(0, stridex);
}
tag[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
});
SSO Token, Including SAP SSO2 and SiteMinder/Network Edge
As in basic authentication, the Datajs 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, Datajs 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
 *
 * Datajs.SSO.js
 * This file will not be regenerated, and it is expected that the user may want to
// 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
        // 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 ||
             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);

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

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

function alertText(error) {
    var txt = JSON.stringify(error);
    alert("Error:
Error:
" + txt);
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.

```javascript
/**
 * 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;

**Sending Requests Over HTTPS**

Datajs custom clients can replace the default odata.DefaultHttpClient with overrideHttpClientForDatajs to send requests to SAP Mobile Server through an HTTPS connection.

var oDataDefaultHttpClient;

function overrideHttpClientForDatajs() {
  if (HttpsConnection.supportHttpsInBrowser === true ) {
    oDataDefaultHttpClient = odata.defaultHttpClient;
    OData.defaultHttpClient = new
    {
        request: function (request, success, error)
        {
            if (request.requestURI.search("/https/i"))
            {
                // invoke HTTPs proxy api
                // return result
            }
            else
            {
                // using original http client
            }
        }
    }
  }
}
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 Windows 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:

```javascript
//Begin, This code enable datajs library on Windows 6.0 and Windows 6.1
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; }
```
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`.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwc.CallbackSet ()</td>
<td>Use for event handlers that are asynchronous.</td>
<td>Callbacks.js</td>
</tr>
<tr>
<td>hwc.CertificateStore</td>
<td>Create a user interface in HTML and JavaScript that uses X.509 certificates as the Hybrid App credentials.</td>
<td>Certificate.js</td>
</tr>
<tr>
<td>hwc.ConnectionSettings</td>
<td>The JavaScript class for the Hybrid Web Container connection settings manages the connection between applications and the server.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.CustomIcon</td>
<td>The JavaScript class for the Hybrid Web Container custom icon, lists custom icons.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.e2eTrace</td>
<td>Allows for an end to end trace of data communication from the client to the back-end.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.getExternalResource</td>
<td>Access resources on external HTTP servers.</td>
<td>ExternalResource.js</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
<td>Defined in</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>hwc.getCurrentLocale</td>
<td>The date/time functions allow you to extract and format the date and time for the Hybrid App.</td>
<td>Timezone.js</td>
</tr>
<tr>
<td>hwc.getPicture</td>
<td>Provides access to the device's default camera application or device's photo library for retrieving a picture asynchronously.</td>
<td>Camera.js</td>
</tr>
<tr>
<td>hwc.HybridApp</td>
<td>Javascript class for the Hybrid App object. Lists installed Hybrid Apps.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.LogEntry</td>
<td>Javascript class for LogEntry object.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.MediaCache</td>
<td>Used within the JavaScript to wrap the source of an image element. Fetches media content from a cache or the server using a URI.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.MenuItemCollection</td>
<td>Represents a collection of menu items.</td>
<td>hwc-comms.js</td>
</tr>
<tr>
<td>hwc.Message</td>
<td>This is the class to encapsulate an incoming message object. When a new message arrives, a notification is sent to users through custom code.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.MessageFilter</td>
<td>This is the class to encapsulate a filter for messages.</td>
<td>hwc-api.js</td>
</tr>
</tbody>
</table>
### Class Description Defined in

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwc.perf</td>
<td>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.</td>
<td>hwc-api.js</td>
</tr>
<tr>
<td>hwc.SUPStorage</td>
<td>Constructs a new storage area identified by a storage key.</td>
<td>SUPStorage.js</td>
</tr>
<tr>
<td>Resources</td>
<td>Access localized string resources.</td>
<td>Resources.js</td>
</tr>
</tbody>
</table>

### 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`.

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

Used to group anonymous objects and callback functions used as method parameters. Methods and fields in this namespace cannot be instantiated. Used for API docs generation only.

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>anonymous.AppLogErrorCallbackParameter</code> on page 70</td>
<td>Object used in <code>anonymous.getLogEntriesErrorCallback</code> on page 78 and <code>anonymous.startOrStopLogListenerErrorCallback</code> on page 83 functions.</td>
</tr>
<tr>
<td><code>anonymous.sendRequestErrorCBParameter</code> on page 70</td>
<td>Object used in <code>anonymous.sendRequestErrorCB</code> on page 82 function.</td>
</tr>
<tr>
<td><code>anonymous.sendRequestSuccessCBParameter</code> on page 71</td>
<td>Object used in <code>anonymous.sendRequestSuccessCB</code> on page 83 function.</td>
</tr>
</tbody>
</table>

Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>options</code> on page 72</td>
<td>Options object used with the <code>getExternalResource</code> function.</td>
</tr>
<tr>
<td><code>PictureOptions</code> on page 72</td>
<td>Options object that is used with the <code>hwc.getPicture</code> on page 192 method.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abort()</code> on page 72</td>
<td>JavaScript function to abort the HTTP(S) request</td>
</tr>
<tr>
<td><code>alertDialogCallbackFunction( message )</code> on page 73</td>
<td>A callback function invoked when <code>hwc.log</code> on page 208 is invoked with true for the <code>notifyUser</code> parameter.</td>
</tr>
<tr>
<td><code>AppInstallationListener( event, moduleId, version, moduleName )</code> on page 73</td>
<td>Callback function that will be invoked on hybrid app installation events.</td>
</tr>
<tr>
<td><code>AppInstallationListener( event, moduleId, version, moduleName )</code> on page 74</td>
<td>Callback function that will be invoked on hybrid app installation events.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>AppInstallationListener( notifications, event, moduleId, version, moduleName, designerVersion, containerVersion )</code> on page 74</td>
<td>Callback function that will be invoked when push notifications are available.</td>
</tr>
<tr>
<td><code>ApplicationListener( event, moduleId, version )</code> on page 75</td>
<td>Callback function that will be invoked on hybrid app events.</td>
</tr>
<tr>
<td><code>complete( resultXHR )</code> on page 76</td>
<td>Callback function used in the Options object.</td>
</tr>
<tr>
<td><code>ConnectionStateListener( event, errorCode, errorMessage )</code> on page 77</td>
<td>Callback function that will be invoked when the connection state changes.</td>
</tr>
<tr>
<td><code>errorCallbackFunction( errorMessage )</code> on page 77</td>
<td>A callback function invoked if there is an error.</td>
</tr>
<tr>
<td><code>genericCallbackFunction()</code> on page 78</td>
<td>A generic callback function that takes no parameters.</td>
</tr>
<tr>
<td><code>getLogEntriesErrorCallback( data )</code> on page 78</td>
<td>Callback function that will be invoked when <code>AppLog.getLogEntries</code> on page 91 fails.</td>
</tr>
<tr>
<td><code>getLogEntriesSuccessCallback( data )</code> on page 78</td>
<td>Callback function that will be invoked with all the entries in the app log.</td>
</tr>
<tr>
<td><code>logListener( date, event, message )</code> on page 79</td>
<td>Callback function that will be invoked when events are logged to the app log.</td>
</tr>
<tr>
<td><code>LogListener( milliseconds, event, optional-String )</code> on page 79</td>
<td>Callback function that will be invoked when events are logged to the event log.</td>
</tr>
<tr>
<td><code>MessageListener( flag, msgId )</code> on page 80</td>
<td>Callback function that will be invoked on message events.</td>
</tr>
<tr>
<td><code>onGetPictureError( err )</code> on page 81</td>
<td>Camera</td>
</tr>
<tr>
<td><code>onGetPictureSuccess( filename, response )</code> on page 82</td>
<td>User provided function that will be invoked when the <code>hwc.getPicture</code> on page 192 function is successful.</td>
</tr>
<tr>
<td><code>sendRequestErrorCB( data )</code> on page 82</td>
<td>Callback function that will be invoked <code>HttpsConnection.get() /sendRequest()</code> failed.</td>
</tr>
<tr>
<td><code>sendRequestSuccessCB( data )</code> on page 83</td>
<td>Callback function that will be invoked <code>HttpsConnection.get() /sendRequest()</code> succeeded.</td>
</tr>
<tr>
<td><code>startOrStopLogListenerErrorCallback( data )</code> on page 83</td>
<td>Callback function that will be invoked upon failure to start a log listener via <code>AppLog.startLogListener</code> on page 92, or upon failure to removing a log listener via <code>AppLog.stopLogListener</code> on page 95.</td>
</tr>
</tbody>
</table>
**startOrStopLogListenerSuccessCallback()** on page 84

Callback function that will be invoked upon successfully starting a log listener via `AppLog.startLogListener` on page 92, or upon successfully removing a log listener via `AppLog.stopLogListener` on page 95.

**Source**
*Camera.js, line 266 on page 266.*

**anonymous.AppLogErrorCallbackParameter class**

Object used in `anonymous.getLogEntriesErrorCallback` on page 78 and `anonymous.startOrStopLogListenerErrorCallback` on page 83 functions.

**Syntax**
```
new AppLogErrorCallbackParameter()
```

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorCode</td>
<td>number</td>
<td>Predefined error code</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>The description of the error</td>
</tr>
</tbody>
</table>

**Source**
*Plugins/AppLog/applog.js, line 479 on page 530.*

**anonymous.sendRequestErrorCBParameter class**

Object used in `anonymous.sendRequestErrorCB` on page 82 function.

**Syntax**
```
new sendRequestErrorCBParameter()
```

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorCode</td>
<td>number</td>
<td></td>
<td>Predefined error code</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td></td>
<td>The description of the error</td>
</tr>
</tbody>
</table>
The native error code reported from Afaria, device, etc (optional)

**Source**

*Plugins/HttpsProxy/https-proxy.js, line 506 on page 558.*

**anonymous.sendRequestSuccessCBParameter class**

Object used in `anonymous.sendRequestSuccessCB` on page 83 function.

**Syntax**

```javascript
new sendRequestSuccessCBParameter()
```

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>number</td>
<td></td>
<td>The HTTP status code</td>
</tr>
<tr>
<td>headers</td>
<td>object</td>
<td></td>
<td>An object that contains headerKey = value pairs.</td>
</tr>
<tr>
<td>responseText</td>
<td>string</td>
<td>&lt;optional&gt;</td>
<td>The text response. This parameter is present only if the response is a text response.</td>
</tr>
<tr>
<td>responseBase64</td>
<td>string</td>
<td>&lt;optional&gt;</td>
<td>Base64 encoded representation of the binary response. This parameter is included only if the response is a binary response.</td>
</tr>
<tr>
<td>clientError</td>
<td>object</td>
<td>&lt;optional&gt;</td>
<td>An optional object that contains the authentication error. It is an object of <code>anonymous.sendRequestErrorCBParameter</code> on page 70.</td>
</tr>
</tbody>
</table>

**Source**

*Plugins/HttpsProxy/https-proxy.js, line 504 on page 558.*
**options member**
Options object used with the `getExternalResource` function.

Supported options are:

- **method**: one of GET, PUT, DELETE, HEAD, OPTIONS, or POST. The default is GET.
- HTTP and HTTPS urls are supported.
- async: request should be sent asynchronously. The default is true.
- headers: request headers to be sent with request.
- data: data to be sent. If this is an array, it is converted to a query string. For a GET request, this is added to the end of the URL.
- `anonymous.complete` on page 76 is a callback function that will be invoked with the resultXHR when this method completes

**Syntax**
```javascript
<static> options
```

**Source**
`ExternalResource.js`, line 270 on page 292.

**PictureOptions member**
Options object that is used with the `hwc.getPicture` on page 192 method.

Contains 2 fields that can be specified.

- **sourceType**: One of `hwc.Picture.SourceType` values
- **destinationType**: One of `hwc.Picture.DestinationType` values

**Syntax**
```javascript
<static> PictureOptions
```

**See**
hwc.getPicture for an example.

**Source**
`Camera.js`, line 272 on page 266.

**abort() method**
JavaScript function to abort the HTTP(S) request

**Syntax**
```javascript
<static> abort()
```
**Source**  
*Plugins/HttpsProxy/https-proxy.js, line 509 on page 558.*

**alertDialogCallbackFunction( message ) method**

A callback function invoked when hwc.log on page 208 is invoked with true for the notifyUser parameter.

This callback should notify the user of the log message in an appropriate manner.

**Syntax**

```javascript
<static> alertDialogCallbackFunction( message )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>string</td>
<td>The message that the user should be notified of.</td>
</tr>
</tbody>
</table>

**Source**  
*hwc-comms.js, line 1608 on page 489.*

**AppInstallationListener( event, moduleId, version, moduleName ) method**

Callback function that will be invoked on hybrid app installation events.

App installation listeners can be added with hwc.addAppInstallationListener on page 149.

**Syntax**

```javascript
<static> AppInstallationListener( event, moduleId, version, moduleName )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>number</td>
<td>A number indicating the event (will be either hwc.INSTALLATION_BEGIN on page 132 or hwc.INSTALLATION_END on page 132).</td>
</tr>
<tr>
<td>moduleId</td>
<td>string</td>
<td>The module ID of the hybrid app the event is about.</td>
</tr>
<tr>
<td>version</td>
<td>string</td>
<td>The version of the hybrid app the event is about.</td>
</tr>
</tbody>
</table>
Develop Hybrid Apps Using Third-party Web Frameworks

<table>
<thead>
<tr>
<th>moduleName</th>
<th>string</th>
<th>The display name of the hybrid app the event is about.</th>
</tr>
</thead>
</table>

**Source**  
*hwc-api.js, line 3673 on page 428.*

**AppNameInstallationListener( event, moduleId, version, moduleName ) method**  
Callback function that will be invoked on hybrid app installation events.

**Syntax**  
<static> AppInstallationListener( event, moduleId, version, moduleName )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Integer</td>
<td>Installation flags including, BEGIN(1), END(2)</td>
</tr>
<tr>
<td>moduleId</td>
<td>String</td>
<td>Optional Module Id</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>Optional Module version</td>
</tr>
<tr>
<td>moduleName</td>
<td>String</td>
<td>Optional Module display name</td>
</tr>
</tbody>
</table>

**Source**  
*hwc-api.js, line 3676 on page 428.*

**AppNameInstallationListener( notifications, event, moduleId, version, moduleName, designerVersion, containerVersion ) method**  
Callback function that will be invoked when push notifications are available.

Push notification listeners can be added with *hwc.addPushNotificationListener* on page 157.

**Syntax**  
<static> AppInstallationListener( notifications, event, moduleId, version, moduleName, designerVersion, containerVersion ) {number}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifications</td>
<td>Array</td>
<td>An array of notifications.</td>
</tr>
</tbody>
</table>
**Returns**
A number indicating whether other push notification listeners should be called after this one. Must be either `hwc.NOTIFICATION_CANCEL` on page 137 (if no more listener callbacks should be called) or `hwc.NOTIFICATION_CONTINUE` on page 137 (if more listener callbacks should be called). Callback function that will be invoked on hybrid app installation events.

**Type:**

number

**Source**

*hwc-api.js, line 3675* on page 428.

### ApplicationListener( event, moduleId, version ) method

Callback function that will be invoked on hybrid app events.

Application listeners can be added with `hwc.addAppListener` on page 150.

**Syntax**

<static> ApplicationListener( event, moduleId, version )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Integer</td>
<td>Installation flags including, BEGIN(1), END(2), FAIL(3)</td>
</tr>
<tr>
<td>moduleId</td>
<td>String</td>
<td>Optional Module Id</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>Optional Module version</td>
</tr>
<tr>
<td>moduleName</td>
<td>String</td>
<td>Optional Module display name</td>
</tr>
<tr>
<td>designerVersion</td>
<td>String</td>
<td>Optional Version of designer used to create app</td>
</tr>
<tr>
<td>containerVersion</td>
<td>String</td>
<td>Optional Version of hybrid web container</td>
</tr>
</tbody>
</table>
Develop Hybrid Apps Using Third-party Web Frameworks

<table>
<thead>
<tr>
<th>event</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>A number indicating what event has taken place (will be one of hwc.APP_REFRESH on page 126, hwc.APP_ADDED on page 126, hwc.APP_UPDATED on page 127, hwc.APP_REMOVED on page 127).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>moduleId</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>The module id of the hybrid app the event is about.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>version</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>The version of the hybrid app the event is about.</td>
<td></td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 3678 on page 428.

**complete(resultXHR) method**
Callback function used in the Options object.

**Syntax**
<static> complete(resultXHR)

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultXHR</td>
<td>object</td>
<td>the response object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The fields/methods available on resultXHR are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. statusText</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. responseText</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. getResponseHeader(key)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. getAllResponsesHeaders()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These fields and methods are not supported for resultXHR:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• open()</td>
</tr>
</tbody>
</table>

Develop Hybrid Apps Using Third-party Web Frameworks
ConnectionStateListener( event, errorCode, errorMessage ) method
Callback function that will be invoked when the connection state changes.
Connection listeners can be added with hwc.addConnectionListener on page 151.

Syntax
<static> ConnectionStateListener( event, errorCode, errorMessage )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>number</td>
<td>A number indicating the event that occurred (will be hwc.CONNECTED on page 127 or hwc.DISCONNECTED on page 130).</td>
</tr>
<tr>
<td>errorCode</td>
<td>number</td>
<td>An error code (0 indicating success).</td>
</tr>
<tr>
<td>errorMessage</td>
<td>string</td>
<td>Text of the error message. Will be empty if there is no error.</td>
</tr>
</tbody>
</table>

errorCallbackFunction( errorMessage ) method
A callback function invoked if there is an error.

Syntax
<static> errorCallbackFunction( errorMessage )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorMessage</td>
<td>string</td>
<td>The message describing the error.</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 3669 on page 428.
**genericCallbackFunction() method**
A generic callback function that takes no parameters.
Used to execute code when a certain event occurs.

*Syntax*
<static> genericCallbackFunction()

*Source*
hwc-comms.js, line 1612 on page 489.

**getLogEntriesErrorCallback( data ) method**
Callback function that will be invoked when AppLog.getLogEntries on page 91 fails.

*Syntax*
<static> getLogEntriesErrorCallback( data )

*Parameters*

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>anonymous.AppLogErrorCallbackParameter on page 70</td>
<td>The error object.</td>
</tr>
</tbody>
</table>

*Source*
Plugins/AppLog/applog.js, line 473 on page 530.

**getLogEntriesSuccessCallback( data ) method**
Callback function that will be invoked with all the entries in the app log.

There will be one AppLog.LogEntry on page 92 object for each line in the app log. Log entries can be retrieved with AppLog.getLogEntries on page 91.

*Syntax*
<static> getLogEntriesSuccessCallback( data )

*Parameters*

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

*Source*
Plugins/AppLog/applog.js, line 471 on page 530.
**logListener( date, event, message ) method**
Callback function that will be invoked when events are logged to the app log.

Log listeners can be added with `AppLog.startLogListener` on page 92.

**Syntax**

```javascript
<static> logListener( date, event, message )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>The date of the log entry.</td>
</tr>
<tr>
<td>event</td>
<td>number</td>
<td>The event ID of the log entry (will be one of the AppLog status events, or possibly a custom value).</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>The string carrying the message of the log entry.</td>
</tr>
</tbody>
</table>

**Source**
`Plugins/AppLog/applog.js, line 481` on page 530.

**LogListener( milliseconds, event, optionalString ) method**
Callback function that will be invoked when events are logged to the event log.

Log listeners can be added with `hwc.addLogListener` on page 153.

**Syntax**

```javascript
<static> LogListener( milliseconds, event, optionalString )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>milliseconds</td>
<td>number</td>
<td>The date of the log message represented in milliseconds.</td>
</tr>
</tbody>
</table>

---

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps 79
| event | number | A number that represents which category this event falls under (It will be one of hwc.CONNECTION_ERROR on page 128, hwc.CONNECTION_OTHER on page 129, hwc.CONNECTION_CONNECTED on page 128, hwc.CONNECTION_DISCONNECTED on page 128, hwc.CONNECTION_RETRIEVED_ITEMS on page 129).
|-------|-------|
| optionalString | string | The string carrying the message of the log event.

**Source**
hwc-api.js, line 3671 on page 428.

**MessageListener( flag, msgId ) method**
Callback function that will be invoked on message events.

Message listeners can be added with `hwc.addMessageListener` on page 155.

**Syntax**

```javascript
<static> MessageListener( flag, msgId )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>number</td>
<td>A number indicating which message event occurred (will be one of hwc.MSG_ADDED on page 135, hwc.MSG_REMOVED on page 136, hwc.MSG_UPDATED on page 137, hwc.MSG_REFRESH on page 136).</td>
</tr>
<tr>
<td>msgId</td>
<td>number</td>
<td>The message id of the affected message.</td>
</tr>
</tbody>
</table>

**Source**
hwc-api.js, line 3680 on page 428.
**onGetPictureError( err ) method**

Camera

**Syntax**

<static> onGetPictureError( err )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>err</td>
<td>number</td>
<td>the error code returned. Possible values are:</td>
</tr>
</tbody>
</table>

1. PictureError.NO_ERROR = 0;
2. PictureError.NOT_SUPPORTED = -1; getPicture() not implemented, camera not present,
3. PictureError.IN_PROGRESS = -2; getPicture() has already been requested but has not yet completed.
4. PictureError.USER_REJECT = -3; the user has canceled the request.
5. PictureError.BAD_OPTIONS = -4; supplied options were not recognized.
6. PictureError.TOO_LARGE = -5; the returned image size was too large to be handled by JavaScript
7. PictureError.UNKNOWN = -6; an unknown error occurred.

**Source**

Camera.js, line 268 on page 266.
**onGetPictureSuccess( filename, response ) method**

User provided function that will be invoked when the `hwc.getPicture` on page 192 function is successful.

**Syntax**

<static> `onGetPictureSuccess( filename, response )`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>file name of the image</td>
</tr>
<tr>
<td>response</td>
<td>string</td>
<td>the response will be either a Base64-encoded JPG string or a URI depending on the options passed to the <code>hwc.getPicture</code> on page 192 function.</td>
</tr>
</tbody>
</table>

- if `options.destinationType` == `PictureOption.DestinationType.IMAGE_URI`, response is an uniform reference identifier for the image. `onGetPictureSuccess(fileName, imageURI)`
- if `options.destinationType` == `PictureOption.DestinationType.IMAGE_DATA`, response is a Base64-encoded string. `onGetPictureSuccess(fileName, imageData)`

**Source**

`Camera.js`, line 270 on page 266.

**sendRequestErrorCB( data ) method**

Callback function that will be invoked `HttpsConnection.get()/sendRequest()` failed.

**Syntax**

<static> `sendRequestErrorCB( data )`
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>data</strong></td>
<td><em>anonymous.sendRequestErrorCBParameter</em> on page 70</td>
<td>The error object.</td>
</tr>
</tbody>
</table>

**Source**

*Plugins/HttpsProxy/https-proxy.js, line 502 on page 558.*

### sendRequestSuccessCB(data) method

Callback function that will be invoked `HttpsConnection.get()/sendRequest()` succeeded.

**Syntax**

```javascript
<static> sendRequestSuccessCB( data )
```

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>data</strong></td>
<td><em>anonymous.sendRequestSuccessCBParameter</em> on page 71</td>
<td>The response data object.</td>
</tr>
</tbody>
</table>

**Source**

*Plugins/HttpsProxy/https-proxy.js, line 500 on page 558.*

### startOrStopLogListenerErrorCallback(data) method

Callback function that will be invoked upon failure to start a log listener via `AppLog.startLogListener` on page 92, or upon failure to removing a log listener via `AppLog.stopLogListener` on page 95.

**Syntax**

```javascript
<static> startOrStopLogListenerErrorCallback( data )
```

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>data</strong></td>
<td><em>anonymous.AppLogErrorCallbackParameter</em> on page 70</td>
<td>The error object.</td>
</tr>
</tbody>
</table>

**Source**

*Plugins/AppLog/applog.js, line 477 on page 530.*
**startOrStopLogListenerSuccessCallback() method**

Callback function that will be invoked upon successfully starting a log listener via `AppLog.startLogListener` on page 92, or upon successfully removing a log listener via `AppLog.stopLogListener` on page 95.

**Syntax**

<static> startOrStopLogListenerSuccessCallback()

**Source**

`Plugins/AppLog/applog.js`, line 475 on page 530.

**AppLog namespace**

The namespace for AppLog plugin

**Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ERR_UNKNOWN</code> on page 86</td>
<td>Constant indicating the operation failed with unknown error.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_CONNECTED</code> on page 86</td>
<td>Constant indicating an app log entry is associated with the client successfully connecting to the SUP server.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_DISCONNECTED</code> on page 86</td>
<td>Constant indicating an app log entry is associated with the client losing connection to the SUP server.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_DISCONNECTED_LOW_STORAGE</code> on page 87</td>
<td>Constant indicating an app log entry is associated with the client losing connection to the SUP server due to low storage.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_DISCONNECTED_ROAMING</code> on page 87</td>
<td>Constant indicating an app log entry is associated with the client losing connection to the SUP server due to roaming.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_FLIGHT_MODE</code> on page 87</td>
<td>Constant indicating an app log entry is associated with the client going into flight mode.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_NOTIFICATION_RECEIVED</code> on page 88</td>
<td>Constant indicating an app log entry is associated with the client receiving a notification.</td>
</tr>
<tr>
<td><code>STATUS_EVENT_OUT_OF_NETWORK</code> on page 88</td>
<td>Constant indicating an app log entry is associated with the client going out of network.</td>
</tr>
</tbody>
</table>
### Status Events

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STATUS_EVENT_REGISTRATION_STARTED</strong> on page 88</td>
<td>Constant indicating an app log entry is associated with the client starting registration.</td>
</tr>
<tr>
<td><strong>STATUSEVENT_RESTART</strong> on page 89</td>
<td>Constant indicating an app log entry is associated with restarting the client connection to the SUP server.</td>
</tr>
<tr>
<td><strong>STATUS_EVENT_SET_DEFAULT_ITEM</strong> on page 89</td>
<td>Constant indicating an app log entry is associated with a default app being set from the server.</td>
</tr>
<tr>
<td><strong>STATUS_EVENT_SHUTDOWN</strong> on page 89</td>
<td>Constant indicating an app log entry is associated with shutting down the client connection to the SUP server.</td>
</tr>
<tr>
<td><strong>STATUS_EVENT_STARTUP</strong> on page 90</td>
<td>Constant indicating an app log entry is associated with starting the client connection to the SUP server.</td>
</tr>
<tr>
<td><strong>STATUS_EVENT_UNKNOWN</strong> on page 90</td>
<td>Constant indicating an app log entry is associated with an unknown event.</td>
</tr>
<tr>
<td><strong>STATUS_EVENT_UNSET_DEFAULT_ITEM</strong> on page 90</td>
<td>Constant indicating an app log entry is associated with a default app being unset from the server.</td>
</tr>
<tr>
<td><strong>STATUS_EVENT_WAITING_TO_CONNECT</strong> on page 91</td>
<td>Constant indicating an app log entry is associated with the client waiting to connect to the SUP server.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getLogEntries( successCB, errorCB )</code> on page 91</td>
<td>Call this function to get an array of <code>AppLog.LogEntry</code> on page 92 objects.</td>
</tr>
<tr>
<td><code>LogEntry( logDate, event, msg )</code> on page 92</td>
<td>This object represents a log entry.</td>
</tr>
<tr>
<td><code>startLogListener( successCB, errorCB, logListener, [containingObject] )</code> on page 92</td>
<td>Registers a log listener.</td>
</tr>
<tr>
<td><code>stopLogListener( successCB, errorCB, logListener, [containingObject] )</code> on page 95</td>
<td>Removes a log listener.</td>
</tr>
</tbody>
</table>

### Source

`Plugins/AppLog/applog.js`, line 16 on page 513.
**ERR UNKNOWN member**
Constant indicating the operation failed with unknown error.

Used in `anonymous.AppLogErrorCallbackParameter` on page 70.

**Syntax**

```typescript
<static> ERR_UNKNOWN : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js, line 23` on page 513.

**STATUS_EVENT_CONNECTED member**
Constant indicating an app log entry is associated with the client successfully connecting to the SUP server.

Used in `AppLog.LogEntry` on page 92.

**Syntax**

```typescript
<static> STATUS_EVENT_CONNECTED : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js, line 53` on page 514.

**STATUS_EVENT_DISCONNECTED member**
Constant indicating an app log entry is associated with the client losing connection to the SUP server.

Used in `AppLog.LogEntry` on page 92.

**Syntax**

```typescript
<static> STATUS_EVENT_DISCONNECTED : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js, line 59` on page 515.
**STATUS_EVENT_DISCONNECTED_LOW_STORAGE** member

Constant indicating an app log entry is associated with the client losing connection to the SUP server due to low storage.

Used in *AppLog.LogEntry* on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_DISCONNECTED_LOW_STORAGE : number
```

**Type**

number

**Source**

*Plugins/AppLog/applog.js, line 89* on page 516.

---

**STATUS_EVENT_DISCONNECTED_ROAMING** member

Constant indicating an app log entry is associated with the client losing connection to the SUP server due to roaming.

Used in *AppLog.LogEntry* on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_DISCONNECTED_ROAMING : number
```

**Type**

number

**Source**

*Plugins/AppLog/applog.js, line 83* on page 516.

---

**STATUS_EVENT_FLIGHT_MODE** member

Constant indicating an app log entry is associated with the client going into flight mode.

Used in *AppLog.LogEntry* on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_FLIGHT_MODE : number
```

**Type**

number

**Source**

*Plugins/AppLog/applog.js, line 65* on page 515.
**STATUS_EVENT_NOTIFICATION_RECEIVED** member
Constant indicating an app log entry is associated with the client receiving a notification.

Used in *AppLog.LogEntry* on page 92.

**Syntax**
<static> STATUS_EVENT_NOTIFICATION_RECEIVED : number

**Type**
number

**Source**
*Plugins/AppLog/applog.js, line 101* on page 516.

**STATUS_EVENT_OUT_OF_NETWORK** member
Constant indicating an app log entry is associated with the client going out of network.

Used in *AppLog.LogEntry* on page 92.

**Syntax**
<static> STATUS_EVENT_OUT_OF_NETWORK : number

**Type**
number

**Source**
*Plugins/AppLog/applog.js, line 71* on page 515.

**STATUS_EVENT_REGISTRATION_STARTED** member
Constant indicating an app log entry is associated with the client starting registration.

Used in *AppLog.LogEntry* on page 92.

**Syntax**
<static> STATUS_EVENT_REGISTRATION_STARTED : number

**Type**
number

**Source**
*Plugins/AppLog/applog.js, line 95* on page 516.
**STATUS_EVENT_RESTART member**
Constant indicating an app log entry is associated with restarting the client connection to the SUP server.

Used in `AppLog.LogEntry` on page 92.

Syntax

<static> STATUS_EVENT_RESTART : number

Type

number

Source

`Plugins/AppLog/applog.js`, line 47 on page 514.

**STATUS_EVENT_SET_DEFAULT_ITEM member**
Constant indicating an app log entry is associated with a default app being set from the server.

Used in `AppLog.LogEntry` on page 92.

Syntax

<static> STATUS_EVENT_SET_DEFAULT_ITEM : number

Type

number

Source

`Plugins/AppLog/applog.js`, line 107 on page 516.

**STATUS_EVENT_SHUTDOWN member**
Constant indicating an app log entry is associated with shutting down the client connection to the SUP server.

Used in `AppLog.LogEntry` on page 92.

Syntax

<static> STATUS_EVENT_SHUTDOWN : number

Type

number

Source

`Plugins/AppLog/applog.js`, line 41 on page 514.
**STATUS_EVENT_STARTUP** member
Constant indicating an app log entry is associated with starting the client connection to the SUP server.

Used in `AppLog.LogEntry` on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_STARTUP : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js`, line 35 on page 514.

**STATUS_EVENT_UNKNOWN** member
Constant indicating an app log entry is associated with an unknown event.

Used in `AppLog.LogEntry` on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_UNKNOWN : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js`, line 29 on page 514.

**STATUS_EVENT_UNSET_DEFAULT_ITEM** member
Constant indicating an app log entry is associated with a default app being unset from the server.

Used in `AppLog.LogEntry` on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_UNSET_DEFAULT_ITEM : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js`, line 113 on page 517.
**STATUS_EVENT_WAITING_TO_CONNECT** member

Constant indicating an app log entry is associated with the client waiting to connect to the SUP server.

Used in `AppLog.LogEntry` on page 92.

**Syntax**

```javascript
<static> STATUS_EVENT_WAITING_TO_CONNECT : number
```

**Type**

number

**Source**

`Plugins/AppLog/applog.js`, line 77 on page 515.

**getLogEntries( successCB, errorCB )** method

Call this function to get an array of `AppLog.LogEntry` objects.

There will be one `AppLog.LogEntry` object for each line in the app log.

**Syntax**

```javascript
<static> getLogEntries( successCB, errorCB )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>successCB</td>
<td><code>anonymous.getLogEntriesSuccessCallback</code> on page 78</td>
<td>The callback function that will receive the asynchronous call-back with the log entries.</td>
</tr>
<tr>
<td>errorCB</td>
<td><code>anonymous.getLogEntriesErrorCallback</code> on page 78</td>
<td>The callback function that will be invoked on errors.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
// A global function called with the log entries.
function onLogEntriesSuccessCallback(data) {
    for ( var i = 0; i < data.length; i++ ) {
        var logEntry = data[ i ];
        alert('Log entry ' + ( i + 1 ) + ':
' + '  ' + 'Date (ms): ' + logEntry.date + '
' + '  ' + 'Status code: ' + logEntry.statusCode + '
' + '  ' + 'Message: ' + logEntry.message
    }
}
```
// A global function called if there is an error retrieving log
entries.
function onLogEntriesFailureCallback(error) {
    alert('Error retrieving log entries: ' + error);
}

// Get the log entries
AppLog.getLogEntries(onLogEntriesSuccessCallback,
onLogEntriesFailureCallback);

Source
Plugins/AppLog/applog.js, line 163 on page 518.

LogEntry( logDate, event, msg ) method
This object represents a log entry.

Syntax
<static> LogEntry( logDate, event, msg )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logDate</td>
<td>number</td>
<td>The date the log entry was recorded, in milliseconds since January 1, 1970, 00:00:00 GMT.</td>
</tr>
<tr>
<td>event</td>
<td>number</td>
<td>The event ID of the log entry (will be one of the AppLog status events, or possibly a custom value).</td>
</tr>
<tr>
<td>msg</td>
<td>string</td>
<td>The message of the log entry.</td>
</tr>
</tbody>
</table>

Source
Plugins/AppLog/applog.js, line 125 on page 517.

startLogListener( successCB, errorCB, logListener, [containingObject] ) method
Registers a log listener.
Syntax
<static> startLogListener( successCB, errorCB, logListener, [containingObject] )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>successCB</td>
<td>anonymous.startOrStopLogListenerSuccessCallback on page 84</td>
<td></td>
<td>A callback function that will be invoked if the log listener is successfully registered.</td>
</tr>
<tr>
<td>errorCB</td>
<td>anonymous.startOrStopLogListenerErrorCallback on page 83</td>
<td></td>
<td>A callback function that will be invoked if there is an error registering the log listener.</td>
</tr>
<tr>
<td>logListener</td>
<td>anonymous.logListener on page 79</td>
<td></td>
<td>The callback to register. This will be invoked when new entries are added to the log.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>Object containing the definition for logListener. If a log listener callback function references variables in its containing object, then the containing object should be passed to this function.</td>
</tr>
</tbody>
</table>

Example

// This example shows how to use this function with a globally-scoped logListener.
// A global function called by the log listener.
var doSomething = function()
{
    alert("this gets displayed when there is a new log entry.");
}

// The log listener callback function that will be passed to AppLog.startLogListener.
// This function will be invoked whenever there is a new log entry.
var logListener = function( date, statusCode, message )
{
    doSomething();
}
function onStartLogListenerSuccessCallback() {
    // Do something here after listener has been added
}

function onStartLogListenerFailureCallback(error) {
    // React to error here
}

// Add the log listener.
AppLog.startLogListener(onStartLogListenerSuccessCallback,
                        onStartLogListenerFailureCallback,
                        logListener);

// This example shows how to use this function with a logListener
// contained in an object.
// logListenerManager is an object that will contain the listener
callback as well
// as a function that will be invoked from the listener callback
// function.
var logListenerManager = {};

// This is a function that is called from the listener callback.
logListenerManager.doSomething = function() {
    alert("this gets displayed when there is a new log entry.");
}

// This is the listener callback that will be passed to
AppLog.startLogListener.
// Since a variable is referenced from the containing object, the
containing object
// will need to be passed to AppLog.startLogListener.
logListenerManager.listener = function( date, statusCode, message ) {
    this.doSomething();
}

function onStartLogListenerSuccessCallback() {
    // Do something here after listener has been added
}

function onStartLogListenerFailureCallback(error) {
    // React to error here
}

// Pass both the listener callback and the containing object.
AppLog.startLogListener(onStartLogListenerSuccessCallback,
                        onStartLogListenerFailureCallback,
                        logListenerManager.listener,
                        logListenerManager);
stopLogListener( successCB, errorCB, logListener, [containingObject] )

**method**

Removes a log listener.

This function should be called with identical parameters that were used when adding the log listener with `AppLog.startLogListener` on page 92.

**Syntax**

```
<static> stopLogListener( successCB, errorCB, logListener, [containingObject] )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>successCB</td>
<td>anonymous.startOrStopLogListenerSuccessCallback on page 84</td>
<td></td>
<td>A callback function that will be invoked if the log listener is successfully removed.</td>
</tr>
<tr>
<td>errorCB</td>
<td>anonymous.startOrStopLogListenerErrorCallback on page 83</td>
<td></td>
<td>A callback function that will be invoked if there is an error removing the log listener.</td>
</tr>
<tr>
<td>logListener</td>
<td>anonymous.logListener on page 79</td>
<td></td>
<td>The callback that was added with <code>AppLog.startLogListener</code> on page 92.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>Object containing the definition for logListener.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
// This example shows how to use this function with a globally-scoped logListener.
// A global function called by the log listener.
var doSomething = function()
{
    alert("this gets displayed when there is a new log entry.");
}

// The log listener callback function that will be passed to AppLog.startLogListener.
// This function will be invoked whenever there is a new log entry.
var logListener = function( date, statusCode, message )
{
    doSomething();
}
```
function onStartLogListenerSuccessCallback() {
    // Do something here after listener has been added
}

function onStartLogListenerFailureCallback(error) {
    // React to error here
}

function onStopLogListenerSuccessCallback() {
    // Do something here after listener has been removed
}

function onStopLogListenerFailureCallback(error) {
    // React to error here
}

// Add the log listener.
AppLog.startLogListener( onStartLogListenerSuccessCallback,
onStartLogListenerFailureCallback,
logListener);

// At some other point if we want to remove the listener, we use the following line.
AppLog.stopLogListener( onStopLogListenerSuccessCallback,
onStopLogListenerFailureCallback,
logListener);

// This example shows how to use this function with a logListener contained in an object.
// logListenerManager is an object that will contain the listener callback as well
// as a function that will be invoked from the listener callback function.
var logListenerManager = {};

// This is a function that is called from the listener callback.
logListenerManager.doSomething = function()
{
    alert("this gets displayed when there is a new log entry.");
}

// This is the listener callback that will be passed to AppLog.startLogListener.
// Since a variable is referenced from the containing object, the containing object
// will need to be passed to AppLog.startLogListener.
logListenerManager.listener = function( date, statusCode, message )
{
    this.doSomething();
}

function onStartLogListenerSuccessCallback() {
    // Do something here after listener has been added
}
function onStartLogListenerFailureCallback(error) {
    // React to error here
}

function onStopLogListenerSuccessCallback() {
    // Do something here after listener has been removed
}

function onStopLogListenerFailureCallback(error) {
    // React to error here
}

// Pass both the listener callback and the containing object.
AppLog.startLogListener(onStartLogListenerSuccessCallback,
onStartLogListenerFailureCallback,
logListenerManager.listener,
logListenerManager);

// At some other point if we want to remove the listener, we use the following line.
AppLog.stopLogListener(onStopLogListenerSuccessCallback,
onStopLogListenerFailureCallback,
logListenerManager.listener,
logListenerManager);

Source
Plugins/AppLog/applog.js, line 376 on page 526.

HttpsConnection namespace
The namespace for HTTP(S) proxy

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpsConnection.CertificateFromAfaria</td>
<td>Create certificate source description object for certificates from Afaria.</td>
</tr>
<tr>
<td>HttpsConnection.CertificateFromFile</td>
<td>Create certificate source description object for certificates from a keystore file.</td>
</tr>
<tr>
<td>HttpsConnection.CertificateFromStore</td>
<td>Create certificate source description object for certificates from system keystore (Keystore in BB, Keychain in iOS and Android).</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
Develop Hybrid Apps Using Third-party Web Frameworks

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>deleteCertificateFromStore(successCB, [errorCB], certificateKey)</code> on page 100</td>
<td>Delete cached certificate from keychain.</td>
</tr>
<tr>
<td><code>generateODataHttpClient()</code> on page 101</td>
<td>Generate an OData HttpClient object over https proxy of native platform.</td>
</tr>
<tr>
<td><code>get(url, header, successCB, [errorCB], [user], [password], [timeout], [certSource])</code> on page 102</td>
<td>Send a HTTP(S) GET request to a remote server.</td>
</tr>
<tr>
<td><code>sendRequest(method, url, header, requestBody, successCB, errorCB, [user], [password], [timeout], [certSource])</code> on page 104</td>
<td>Send a HTTP(S) request to a remote server.</td>
</tr>
</tbody>
</table>

**Source**
Plugins/HttpsProxy/datajs-https-proxy.js, line 18 on page 533.

**HttpsConnection.CertificateFromAfaria class**
Create certificate source description object for certificates from Afaria.

**Syntax**

```
new CertificateFromAfaria(CN, [ChallengeCode])
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>string</td>
<td></td>
<td>Common Name (CN) for CA/SCEP protocol. For iOS, the retrieved certificate is stored in the key store with the common name as the certificate key, the following requests for the same common name will just load the saved certificate from key store, instead of sending a new request to Afaria server.</td>
</tr>
<tr>
<td>ChallengeCode</td>
<td>string</td>
<td>(optional)</td>
<td>Challenge code for CA/SCEP protocol.</td>
</tr>
</tbody>
</table>
**HttpsConnection.CertificateFromFile class**
Create certificate source description object for certificates from a keystore file.

Not supported on Blackberry platform

**Syntax**
new CertificateFromFile( *Path*, *Password*, *CertificateKey* )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Path</em></td>
<td>string</td>
<td>Path of the keystore file. For iOS client, it first tries to load the relative file path from application's Documents folder; if it fails, then tries to load the file path from application's main bundle. In addition, before trying to load the certificate from file system, iOS client first checks whether the specified certificate key already exists in the key store, if so, it just loads the existing certificate from key store, instead of loading the certificate from file system.</td>
</tr>
<tr>
<td><em>Password</em></td>
<td>string</td>
<td>Password of the keystore.</td>
</tr>
<tr>
<td><em>CertificateKey</em></td>
<td>string</td>
<td>An unique key that will be used to locate the certificate.</td>
</tr>
</tbody>
</table>

**Source**
Plugins/HttpsProxy/https-proxy.js, line 119 on page 543.

**HttpsConnection.CertificateFromStore class**
Create certificate source description object for certificates from system keystore (Keystore in BB, Keychain in iOS and Android).

The certificateKey is not used on the BB platform. BB will prompt the user to select a certificate if a certificate was not already used for the server connection.
Syntax
new CertificateFromStore(CertificateKey)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CertificateKey</td>
<td>string</td>
<td>An unique key that will be used to locate the certificate. Not used in BB platform.</td>
</tr>
</tbody>
</table>

Source
Plugins/HttpsProxy/https-proxy.js, line 152 on page 544.

deleteCertificateFromStore( successCB, [errorCB], certificateKey ) method
Delete cached certificate from keychain.
iOS client will always try the cached certificate first if it is available before requesting the certificate from afaria server or loading the certificate from file system. In case the cached certificate is no longer valid, use this method to delete it from keychain Only supported by iOS platform

Syntax
<static> deleteCertificateFromStore( successCB, [errorCB], certificateKey )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>successCB</td>
<td>anonymous.sendRequestSuccessCB on page 83</td>
<td></td>
<td>Callback method upon success.</td>
</tr>
<tr>
<td>errorCB</td>
<td>anonymous.sendRequestErrorCB on page 82</td>
<td>(optional)</td>
<td>Callback method upon failure.</td>
</tr>
<tr>
<td>certificateKey</td>
<td>string</td>
<td></td>
<td>The key of the certificate to be deleted.</td>
</tr>
</tbody>
</table>

Source
Plugins/HttpsProxy/https-proxy.js, line 409 on page 555.
**generateODataHttpClient() method**
Generate an OData HttpClient object over https proxy of native platform.

This object will re-direct all odata request to the http proxy because even with HTTP connection, there are are some known issue by default setting since the application in device is cross server accessing the odata service. See: [http://datajs.codeplex.com/discussions/396112](http://datajs.codeplex.com/discussions/396112) for details of the issue. Call this method normally on HTML page load event to replace the default odata HTTP client.

**Syntax**
`<static> generateODataHttpClient()`

**Example**
```javascript
// Call datajs api without certificate, users could call just as normal by passing
// URL as first argument
var length = 0;
var updateUri = server + "/example.svc/Categories(1)";

OData.read(server + "/example.svc/Categories", function (data, response) {
    alert("length " + data.results.length);
    length = data.results.length;
    if ( length > 0 ) {
        var updateRequest = {
            requestUri: updateUri,
            method: "PUT",
            data:
            {
                Picture: new Date().getTime(),
                Description: "Update Record",
                CategoryName: "Updated Category",
                CategoryID: 1
            }
        };

        OData.request(updateRequest, function (data, response) {
            alert("Response " + JSON.stringify(response));
        }, function (err) {
            alert("Error occurred " + err.message);
        });
    };
}, function (err) {
    alert("Error occurred " + err.message);
});
```
// However, to specify certificate source in the method call, users need to pass in
// the request object instead of URL, and add the field "certificateSource" to the request object.
var length = 0;
var updateUri = server + "\example.svc/Categories(1)";

OData.read({requestUri: server + "\example.svc/Categories",
certificateSource : cert},
    function (data, response) {
        alert("length " + data.results.length);
        length = data.results.length;
        if (length > 0)
            {
                var updateRequest = {
                    requestUri: updateUri,
                    certificateSource : cert,
                    method: "PUT",
                    data:
                    {
                        Picture: new Date().getTime(),
                        Description: "Update Record",
                        CategoryName: "Updated Category",
                        CategoryID: 1
                    }
                };

                OData.request(updateRequest,
                    function (data, response) {
                        alert("Response " + JSON.stringify(response));
                    },
                    function (err) {
                        alert("Error occurred " + err.message);
                    });
            };
    },
    function (err) {
        alert("Error occurred " + err.message);
    });

Source
Plugins/HttpsProxy/datajs-https-proxy.js, line 109 on page 536.

get( url, header, successCB, [errorCB], [user], [password], [timeout],
    [certSource] ) method
Send a HTTP(S) GET request to a remote server.

Syntax
<static> get( url, header, successCB, [errorCB], [user], [password], [timeout], [certSource] )
{anonymous.abort}
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>string</td>
<td></td>
<td>The http url with format http(s):// [user:password]@hostname[:port]/path.</td>
</tr>
<tr>
<td>header</td>
<td>Object</td>
<td></td>
<td>HTTP header to be sent to server. This is an Object. Can be null.</td>
</tr>
<tr>
<td>successCB</td>
<td>anonymous.sendRequestSuccessCB on page 83</td>
<td>(optional)</td>
<td>Callback method upon success.</td>
</tr>
<tr>
<td>errorCB</td>
<td>anonymous.sendRequestErrorCB on page 82</td>
<td>(optional)</td>
<td>Callback method upon failure.</td>
</tr>
<tr>
<td>user</td>
<td>string</td>
<td>(optional)</td>
<td>User ID for basic authentication.</td>
</tr>
<tr>
<td>password</td>
<td>string</td>
<td>(optional)</td>
<td>User password for basic authentication.</td>
</tr>
<tr>
<td>timeout</td>
<td>number</td>
<td>(optional)</td>
<td>Timeout setting in seconds.</td>
</tr>
<tr>
<td>certSource</td>
<td>Object</td>
<td>(optional)</td>
<td>Certificate description object. It can be one of $\text{HttpsConnection.CertificateFromFile}$ on page 99, $\text{HttpsConnection.CertificateFromStore}$ on page 99, or $\text{HttpsConnection.CertificateFromAfaria}$ on page 98.</td>
</tr>
</tbody>
</table>

### Returns

A JavaScript function object to cancel the operation.

Type:

`anonymous.abort` on page 72
Example

// To send a get request to server, call the method
HttpsConnection.get("http://www.google.com", null, function (data) {
    alert("Status: " + JSON.stringify(data.status));
    alert("Headers: " + JSON.stringify(data.headers));
    if (data.responseText){
        alert("Response: " +
        JSON.stringify(data.responseText));
    }
},
    function (error) {
        alert("Failed: " + JSON.stringify(error));
    });

// To send a get request to server with headers, call the method
HttpsConnection.get(url, {HeaderName : "Header value"}, successCB, errorCB);

// To send a get request to server with basic authentication, call
the method
HttpsConnection.get(url, headers, successCB, errorCB, "username", "password");

// To send a get request to server with mutual authentication, call
the method
HttpsConnection.get("https://hostname", headers, successCB, errorCB, null, null, 0,
    new CertificateFromFile("/mnt/sdcard/my.p12", "password", "mykey"));

Source
Plugins/HttpsProxy/https-proxy.js, line 395 on page 554.

sendRequest( method, url, header, requestBody, successCB, errorCB, [user],
    [password], [timeout], [certSource] ) method
Send a HTTP(S) request to a remote server.

Syntax
<static> sendRequest( method, url, header, requestBody, successCB, errorCB, [user],
    [password], [timeout], [certSource] ) (anonymous.abort)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>string</td>
<td></td>
<td>Standard HTTP request method name.</td>
</tr>
<tr>
<td><strong>url</strong></td>
<td><strong>string</strong></td>
<td>The http url with format http(s):// [user:password]@hostname[:port]/path.</td>
<td></td>
</tr>
<tr>
<td><strong>header</strong></td>
<td><strong>Object</strong></td>
<td>HTTP header to be sent to server. This is an Object. Can be null.</td>
<td></td>
</tr>
<tr>
<td><strong>requestBody</strong></td>
<td><strong>string</strong></td>
<td>Data to be sent to server with the request. It’s a string value. Can be null.</td>
<td></td>
</tr>
<tr>
<td><strong>successCB</strong></td>
<td><strong>anonymous.sendRequestSuccessCB on page 83</strong></td>
<td>Callback method upon success.</td>
<td></td>
</tr>
<tr>
<td><strong>errorCB</strong></td>
<td><strong>anonymous.sendRequestErrorCB on page 82</strong></td>
<td>Callback method upon failure.</td>
<td></td>
</tr>
<tr>
<td><strong>user</strong></td>
<td><strong>string</strong></td>
<td>(optional) User ID for basic authentication.</td>
<td></td>
</tr>
<tr>
<td><strong>password</strong></td>
<td><strong>string</strong></td>
<td>(optional) User password for basic authentication.</td>
<td></td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td><strong>number</strong></td>
<td>(optional) Timeout setting in seconds.</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**
A JavaScript function object to cancel the operation.

**Type:**
Example

// To send a post request to server, call the method
HttpsConnection.sendRequest("POST", "http://www.google.com", null,
"THIS IS THE BODY", function (data) {
  alert("Status: " + JSON.stringify(data.status));
  alert("Headers: " + JSON.stringify(data.headers));
  alert("Response: " + JSON.stringify(data.response));
}, function (data) {
  alert("Failed: " + JSON.stringify(data));});

// To send a post request to server with headers, call the method
HttpsConnection.sendRequest("POST", url, {HeaderName : "HeaderValue"},
"THIS IS THE BODY", successCB, errorCB);

// To send a post request to server with basic authentication, call the method
HttpsConnection.sendRequest("POST", url, headers, "THIS IS THE BODY",
successCB, errorCB, "username", "password");

// To send a post request to server with mutual authentication, call the method
HttpsConnection.sendRequest("POST", "https://hostname", headers,
"THIS IS THE BODY", successCB, errorCB, null,
null, 0, new CertificateFromFile("/mnt/sdcard/my.keystore",
"password", "mykey");

Source
Plugins/HttpsProxy/https-proxy.js, line 289 on page 550.

hwc namespace

The namespace for the Hybrid Web Container javascript

Classes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwc.SUPStorage on page 116</td>
<td>Storage</td>
</tr>
<tr>
<td>hwc.SUPStorageException on page 120</td>
<td>Storage</td>
</tr>
</tbody>
</table>

Namespaces

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeErrorCode on page 121</td>
<td>This object contains constants representing the different types of public native error codes.</td>
</tr>
</tbody>
</table>

Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>APP_ADDED</strong> on page 126</td>
<td>A constant indicating that a hybrid app has been added.</td>
</tr>
<tr>
<td><strong>APP_REFRESH</strong> on page 126</td>
<td>A constant indicating that the application list requires a refresh.</td>
</tr>
<tr>
<td><strong>APP_REMOVED</strong> on page 127</td>
<td>A constant indicating that a hybrid app was removed.</td>
</tr>
<tr>
<td><strong>APP_UPDATED</strong> on page 127</td>
<td>A constant indicating that a hybrid app was updated.</td>
</tr>
<tr>
<td><strong>CONNECTED</strong> on page 127</td>
<td>Constant indicating that the hwc is connected.</td>
</tr>
<tr>
<td><strong>CONNECTION_CONNECTED</strong> on page 128</td>
<td>A constant indicating that the log message is about the connection being established.</td>
</tr>
<tr>
<td><strong>CONNECTION_DISCONNECTED</strong> on page 128</td>
<td>A constant indicating that the log message is about the connection being disconnected.</td>
</tr>
<tr>
<td><strong>CONNECTION_ERROR</strong> on page 128</td>
<td>A constant indicating that the log message is about a connection error.</td>
</tr>
<tr>
<td><strong>CONNECTION_OTHER</strong> on page 129</td>
<td>A constant indicating that the log message is not about the connection.</td>
</tr>
<tr>
<td><strong>CONNECTION_RETRIEVED_ITEMS</strong> on page 129</td>
<td>A constant indicating that the log message is about retrieved items.</td>
</tr>
<tr>
<td><strong>DEFAULT_CUSTOM_ICON_INDEX</strong> on page 129</td>
<td>A constant indicating the custom icon index.</td>
</tr>
<tr>
<td><strong>DISCONNECTED</strong> on page 130</td>
<td>Constant indicating that the hwc is disconnected.</td>
</tr>
<tr>
<td><strong>e2eTrace</strong> on page 130</td>
<td>Represents an E2E Trace.</td>
</tr>
<tr>
<td><strong>INSTALLATION_BEGIN</strong> on page 132</td>
<td>A constant indicating that the application is starting to be installed.</td>
</tr>
<tr>
<td><strong>INSTALLATION_END</strong> on page 132</td>
<td>A constant indicating that the application has finished being installed.</td>
</tr>
<tr>
<td><strong>MediaCache</strong> on page 133</td>
<td>Represents a Media Cache.</td>
</tr>
<tr>
<td><strong>MSG_ADDED</strong> on page 135</td>
<td>A constant indicating that a message has been added.</td>
</tr>
<tr>
<td><strong>MSG_PRIORITY_HIGH</strong> on page 135</td>
<td>A constant indicating a message has high priority.</td>
</tr>
<tr>
<td><strong>MSG_PRIORITY_NORMAL</strong> on page 136</td>
<td>A constant indicating a message has normal priority.</td>
</tr>
<tr>
<td>Constant Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>MSG_REFRESH</strong> on page 136</td>
<td>A constant indicating that a message needs to be refreshed.</td>
</tr>
<tr>
<td><strong>MSG_REMOVED</strong> on page 136</td>
<td>A constant indicating that a message has been removed.</td>
</tr>
<tr>
<td><strong>MSG_UPDATED</strong> on page 137</td>
<td>A constant indicating that a message has been updated.</td>
</tr>
<tr>
<td><strong>NOTIFICATION_CANCEL</strong> on page 137</td>
<td>A constant indicating that no more push notification listeners should be called.</td>
</tr>
<tr>
<td><strong>NOTIFICATION_CONTINUE</strong> on page 137</td>
<td>A constant indicating that other push notification listeners should continue to be called.</td>
</tr>
<tr>
<td><strong>OPEN_APP_NOT_EXIST</strong> on page 138</td>
<td>A constant indicating that <code>hwc.openApp</code> on page 221 failed because the specified app does not exist.</td>
</tr>
<tr>
<td><strong>OPEN_APP_OTHER</strong> on page 138</td>
<td>A constant indicating that <code>hwc.openApp</code> on page 221 failed for an unspecified reason.</td>
</tr>
<tr>
<td><strong>OPEN_APP_SUCCESS</strong> on page 138</td>
<td>A constant indicating that <code>hwc.openApp</code> on page 221 completed successfully.</td>
</tr>
<tr>
<td><strong>OPEN_MSG_APP_NOT_EXIST</strong> on page 139</td>
<td>A constant indicating that a message could not be opened because there was no associated hybrid app.</td>
</tr>
<tr>
<td><strong>OPEN_MSG_NOT_EXIST</strong> on page 139</td>
<td>A constant indicating that a message could not be opened because no message with the given ID exists.</td>
</tr>
<tr>
<td><strong>OPEN_MSG_OTHER</strong> on page 139</td>
<td>A constant indicating that a message could not be opened due to an unspecified error.</td>
</tr>
<tr>
<td><strong>OPEN_MSG_SUCCESS</strong> on page 140</td>
<td>A constant indicating that a message was successfully opened.</td>
</tr>
<tr>
<td><strong>perf</strong> on page 140</td>
<td>Represents the Performance Manager.</td>
</tr>
<tr>
<td><strong>PictureError</strong> on page 142</td>
<td>An array that holds all possible error codes</td>
</tr>
<tr>
<td><strong>REG_ERR_AUTO_REG_NOT_ENABLED</strong> on page 144</td>
<td>Constant indicating that auto registration was not enabled in the template.</td>
</tr>
<tr>
<td><strong>REG_ERR_AUTO_REG_TEMPLATE_NOT_FOUND</strong> on page 145</td>
<td>Constant indicating that no MBS template was found for given appId and/or Security configuration.</td>
</tr>
</tbody>
</table>
### REGISTRATION METHODS

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG_ERR_AU-TO_REG_USER_NAME_TOO_LONG on page 145</td>
<td>Constant indicating that the user name is longer than the legal limit.</td>
</tr>
<tr>
<td>REG_ERR_AU-TO_REG_WRONG_USER_FOR_DEVICE on page 145</td>
<td>Constant indicating that the given device id is already registered for another user.</td>
</tr>
<tr>
<td>REG_ERR_COULD_NOT_REACH_MMS_SERVER on page 146</td>
<td>Constant indicating that the connection to the MMS service failed.</td>
</tr>
<tr>
<td>REG_ERR_INVALID_USER_NAME on page 146</td>
<td>Constant indicating that the user name contains invalid characters.</td>
</tr>
<tr>
<td>REG_ERR_MMS_AUTHENTICATION_FAILED on page 146</td>
<td>Constant indicating that MMS Authentication failed.</td>
</tr>
<tr>
<td>REGISTRATION_METHOD_AFARIA on page 147</td>
<td>Constant indicating that automatic registration using a certificate from Afaria is the preferred method.</td>
</tr>
<tr>
<td>REGISTRATION_METHOD_AUTOMATIC on page 147</td>
<td>Constant indicating that automatic registration using password is the preferred method.</td>
</tr>
<tr>
<td>REGISTRATION_METHOD_CERTIFICATE on page 147</td>
<td>Constant indicating that automatic registration using a local certificate is the preferred method.</td>
</tr>
<tr>
<td>REGISTRATION_METHOD_MANUAL on page 148</td>
<td>Constant indicating that manual registration is the preferred method.</td>
</tr>
<tr>
<td>REGISTRATION_METHOD_NO_PREFERENCE on page 148</td>
<td>Constant indicating no registration method preference.</td>
</tr>
<tr>
<td>SETTING_SUCCESS on page 148</td>
<td>Constant indicating hwc.saveSettings on page 237 completed successfully.</td>
</tr>
<tr>
<td>STATUS on page 149</td>
<td>This object contains constants representing the status of the hybrid app.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activationRequired() on page 149</td>
<td>This function sets the activation required state of this hybrid app to true.</td>
</tr>
<tr>
<td>addAppInstallationListener( AppInstallationListener ) on page 149</td>
<td>Register the application installation listener.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>addAppListener(ApplicationListener, [containingObject])</code> on page 150</td>
<td>Register the application listener.</td>
</tr>
<tr>
<td><code>addConnectionListener(ConnectionStateListener, [containingObject])</code> on page 151</td>
<td>Register the connection state listener.</td>
</tr>
<tr>
<td><code>addLogListener(LogListener, [containingObject])</code> on page 153</td>
<td>Register the log listener.</td>
</tr>
<tr>
<td><code>addMenuItemCollection(collection)</code> on page 154</td>
<td>This function adds a menu item collection to the menu items for the screen.</td>
</tr>
<tr>
<td><code>addMessageListener(filters, MessageListener, [containingObject])</code> on page 155</td>
<td>Registers a message listener.</td>
</tr>
<tr>
<td><code>addPushNotificationListener(PushNotificationListener, [containingObject])</code> on page 157</td>
<td>Register a push notification listener.</td>
</tr>
<tr>
<td><code>CertificateStore()</code> on page 158</td>
<td>Use these functions for X.509 credential handling.</td>
</tr>
<tr>
<td><code>clearCache()</code> on page 164</td>
<td>This function clears the contents of the on-device request result cache for the current hybrid app.</td>
</tr>
<tr>
<td><code>clearCacheItem(cachekey)</code> on page 164</td>
<td>This function clears an item from the contents of the on-device request result cache for the current hybrid app.</td>
</tr>
<tr>
<td><code>ClientVariables(clientVariablesVersion, clientVariableItems)</code> on page 165</td>
<td>Represents a ClientVariables object.</td>
</tr>
<tr>
<td><code>ClientVariablesException(errCode, errMsg)</code> on page 168</td>
<td>This exception is thrown when hwc.ClientVariables#getVariableValueByName is called with a variable name that does not exist.</td>
</tr>
<tr>
<td><code>close()</code> on page 168</td>
<td>This function closes the hybrid app.</td>
</tr>
<tr>
<td><code>ConnectionSettings(regmethod, server, port, server, user, activationcode, protocol, password, urlsuffix)</code> on page 168</td>
<td>Represents the connection settings for connecting to the SUP Server.</td>
</tr>
<tr>
<td><code>connectToServer([onNotification])</code> on page 170</td>
<td>Resumes the connection to the SUP server.</td>
</tr>
<tr>
<td><code>convertLocalTimeToUtc(date)</code> on page 171</td>
<td>Timezone</td>
</tr>
<tr>
<td><code>convertUtcToLocalTime(date)</code> on page 171</td>
<td>Timezone</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>disconnectFromServer()</code> on page 175</td>
<td>Suspends the connection to the SUP server.</td>
</tr>
<tr>
<td><code>expireCredentials()</code> on page 175</td>
<td>Allows the user to set the credentials to the expired state for the current hybrid app.</td>
</tr>
<tr>
<td><code>getAllMessages( [messageFilter], [completeList] )</code> on page 175</td>
<td>Gets received messages based on a filter and the existence of a default hybrid app.</td>
</tr>
<tr>
<td><code>getAppByID( moduleID, version )</code> on page 177</td>
<td>Gets a <code>hwc.HybridApp</code> object with the given module id and version.</td>
</tr>
<tr>
<td><code>getAppIconUrl( app, processed )</code> on page 177</td>
<td>This function gets the URL of the icon for a hybrid app depending on whether custom icons are defined.</td>
</tr>
<tr>
<td><code>getApplicationConnectionID()</code> on page 178</td>
<td>Gets the Hybrid Web Container application connection ID.</td>
</tr>
<tr>
<td><code>getBuiltInIconUrl( iconIndex, processed )</code> on page 179</td>
<td>Gets the icon URL for the built-in icon.</td>
</tr>
<tr>
<td><code>getCallbackFromNativeError( errString )</code> on page 180</td>
<td>Extract the error call back method name from a URL string.</td>
</tr>
<tr>
<td><code>getClientVariables( moduleID, version )</code> on page 180</td>
<td>Gets the client variables of the hybrid app with given module id and version.</td>
</tr>
<tr>
<td><code>getCodeFromNativeError( errString )</code> on page 181</td>
<td>Extract an error code from a URL string.</td>
</tr>
<tr>
<td><code>getCurrentApp()</code> on page 182</td>
<td>Gets the hybrid app that is currently open.</td>
</tr>
<tr>
<td><code>getCurrentLocale()</code> on page 182</td>
<td>Timezone</td>
</tr>
<tr>
<td><code>getCustomIconUrl( moduleId, moduleVersion, iconIndex, processed )</code> on page 183</td>
<td>Gets the URL to the custom icon.</td>
</tr>
<tr>
<td><code>getDstOffsetAtGivenTimeInMinutes( date )</code> on page 183</td>
<td>Timezone</td>
</tr>
<tr>
<td><code>getExternalResource( url, options )</code> on page 184</td>
<td>Makes an external cross domain request.</td>
</tr>
<tr>
<td><code>getLocalizedDate( date )</code> on page 186</td>
<td>Timezone</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| `getLocalizedDateTime(date)` | on page 186
| Timezone |
| `getLocalizedTime(date)` | on page 187
| Timezone |
| `getLogEntries()` | on page 187
| Call this function to get an array of `hwc.LogEntry` objects. |
| `getLoggingAlertDialog()` | on page 188
| This function gets the callback used by `hwc.log` when it is required to notify the user of a log item. |
| `getLoggingCurrentLevel()` | on page 188
| This function gets the logging level. |
| `getMessageByID(msgId)` | on page 189
| Gets a `hwc.Message` object with the given message ID. |
| `getMsgIconUrl(msg)` | on page 189
| This function gets the URL of the icon for a message object depending on its processed status and whether there are custom icons defined. |
| `getNativeMessageFromNativeError(errString)` | on page 190
| Extract a native message from a URL string. |
| `getOffsetFromUTC(date)` | on page 191
| Timezone |
| `getOnErrorMessageFromNativeError(errString)` | on page 191
| Extract the error message from a URL string. |
| `getPicture(onGetPictureError, onGetPictureSuccess, options)` | on page 192
| Camera |
| `getQueryVariable(variable)` | on page 193
| This function looks in the query string on the URL for the value corresponding to the given name. |
| `getServerInitiatedApps()` | on page 193
| Returns an array of `hwc.HybridApp` objects that are server initiated. |
| `getSharedStorageKey()` | on page 194
| Storage |
| `getTimezoneId()` | on page 194
| Timezone |
| `getTransformData()` | on page 195
| Returns the transform data for the hybridapp. |
| `getURLParamFromNativeError(paramName, url)` | on page 195
| Extract a parameter value from a URL string with a given parameter name. |
| `getUsesDST()` | on page 195
| Timezone |
| `getXMLHTTPRequest()` | on page 196
<p>| Reliably returns an XMLHttpRequest object regardless of what platform this code is being executed on. |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>guid()</code> on page 196</td>
<td>This function generates a GUID (globally unique identifier).</td>
</tr>
<tr>
<td><code>hideProgressDialog()</code> on page 197</td>
<td>This function hides the progress dialog displaying the spinner.</td>
</tr>
<tr>
<td><code>HybridApp( moduleId, version, displayName, iconIndex, defaultCustomIcon, customIconList )</code> on page 197</td>
<td>This object represents a hybrid app.</td>
</tr>
<tr>
<td><code>isAndroid()</code> on page 201</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isAndroid3()</code> on page 201</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isBlackBerry()</code> on page 201</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isBlackBerry5()</code> on page 202</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isBlackBerry5WithTouchScreen()</code> on page 202</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isBlackBerry6NonTouchScreen()</code> on page 202</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isBlackBerry7()</code> on page 203</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isClosed()</code> on page 203</td>
<td>This function checks if the hybrid app has been closed.</td>
</tr>
<tr>
<td><code>isDstActiveAtGivenTime( date )</code> on page 204</td>
<td>Timezone</td>
</tr>
<tr>
<td><code>isIOS()</code> on page 204</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isIOS4()</code> on page 205</td>
<td>Returns true if the hybrid app application is being run on iOS4</td>
</tr>
<tr>
<td><code>isIOS5()</code> on page 205</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isIOS6()</code> on page 205</td>
<td>Returns true if the hybrid app application is being run on iOS6</td>
</tr>
<tr>
<td><code>isIOS7()</code> on page 206</td>
<td>Returns true if the hybrid app application is being run on iOS7</td>
</tr>
<tr>
<td><code>isiPad()</code> on page 206</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isSharedStorageEnabled()</code> on page 206</td>
<td>Storage</td>
</tr>
<tr>
<td><code>isWindows()</code> on page 207</td>
<td>Platform</td>
</tr>
<tr>
<td><code>isWindowsMobile()</code> on page 207</td>
<td>Platform</td>
</tr>
<tr>
<td><code>loadSettings()</code> on page 207</td>
<td>Loads the current connection settings from the native application storage.</td>
</tr>
<tr>
<td>Method/Constructor</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>log(sMsg, eLevel, notifyUser)</code> on page 208</td>
<td>Allows the user to log a message to the device trace log which can be remotely retrieved from the server.</td>
</tr>
<tr>
<td><code>LogEntry(date, event, msg)</code> on page 209</td>
<td>This object represents a log entry.</td>
</tr>
<tr>
<td><code>markAsActivated()</code> on page 211</td>
<td>This function sets the activation required state for the current hybrid app to false.</td>
</tr>
<tr>
<td><code>markAsProcessed()</code> on page 211</td>
<td>Allows the user to set the processed state to true for the current message.</td>
</tr>
<tr>
<td><code>MenuItemCollection()</code> on page 212</td>
<td>This class represents a collection of menu items.</td>
</tr>
<tr>
<td><code>Message(msgId, date, icon, sender, isRead, processed, priority, subject, module, version)</code> on page 215</td>
<td>Represents a message received by the HWC.</td>
</tr>
<tr>
<td><code>MessageFilter([sender], [subject], [moduleId], [version], [isread], [processed])</code> on page 221</td>
<td>Represents a filter used to filter messages.</td>
</tr>
<tr>
<td><code>openApp(moduleId, version)</code> on page 221</td>
<td>Launch the hybrid app with the given module ID and version.</td>
</tr>
<tr>
<td><code>openMessage(msgId)</code> on page 222</td>
<td>Launch the server initiated hybrid app associated with a message.</td>
</tr>
<tr>
<td><code>removeAllMenuItems()</code> on page 223</td>
<td>This function removes all menu items that were added by the hybrid app.</td>
</tr>
<tr>
<td><code>removeAppInstallationListener(AppInstallationListener)</code> on page 223</td>
<td>Remove the application installation listener.</td>
</tr>
<tr>
<td><code>removeAppListener(ApplicationListener, [containingObject])</code> on page 224</td>
<td>Remove the application listener.</td>
</tr>
<tr>
<td><code>removeConnectionListener(ConnectionStateListener, [containingObject])</code> on page 226</td>
<td>Remove the connection state listener.</td>
</tr>
<tr>
<td><code>removeLogListener(LogListener, [containingObject])</code> on page 227</td>
<td>Remove the log listener.</td>
</tr>
<tr>
<td><code>removeMessage(msgId)</code> on page 229</td>
<td>Removes (deletes) a message.</td>
</tr>
<tr>
<td><code>removeMessageListener(MessageListener, [containingObject])</code> on page 229</td>
<td>Removes the message listener.</td>
</tr>
<tr>
<td><code>removePushNotificationListener(PushNotificationListener, [containingObject])</code> on page 231</td>
<td>Remove the push notification listener.</td>
</tr>
<tr>
<td>Function Call</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>sample_AppListener( event, moduleId, version )</code> on page 232</td>
<td>A sample anonymous.ApplicationListener on page 75 callback function.</td>
</tr>
<tr>
<td><code>sample_ConnectionListener( event, errorCode, errorMessage )</code> on page 233</td>
<td>A sample anonymous.ConnectionStateListener on page 77 callback function.</td>
</tr>
<tr>
<td><code>sample_InstallationAppListener( event, moduleId, version, moduleName, designerVersion, containerVersion )</code> on page 234</td>
<td>Sample application listener callback function</td>
</tr>
<tr>
<td><code>sample_LogListener( milliseconds, event, optionalString )</code> on page 234</td>
<td>Sample anonymous.LogListener on page 79 callback function.</td>
</tr>
<tr>
<td><code>sample_MessageListener( flag, msgId )</code> on page 235</td>
<td>A sample anonymous.MessageListener on page 80 callback function.</td>
</tr>
<tr>
<td><code>sample_PushNotificationListener( notifications )</code> on page 235</td>
<td>A sample implementation of a anonymous.PushNotificationListener callback function.</td>
</tr>
<tr>
<td><code>saveLoginCertificate( certificate )</code> on page 236</td>
<td>This function saves login credentials from a certificate to the credential cache.</td>
</tr>
<tr>
<td><code>saveLoginCredentials( userName, password )</code> on page 236</td>
<td>This function saves login credentials to the credential cache.</td>
</tr>
<tr>
<td><code>saveSettings( settings )</code> on page 237</td>
<td>Save the connection settings to native application storage.</td>
</tr>
<tr>
<td><code>setLoggingAlertDialog( newAlertDialogCallback )</code> on page 238</td>
<td>This function sets the callback used by hwc.log when it is required to notify the user of a log item.</td>
</tr>
<tr>
<td><code>setLoggingCurrentLevel( newLoggingLevel )</code> on page 239</td>
<td>This function sets the logging level.</td>
</tr>
<tr>
<td><code>setErrorFromNativeCallback( callbackToSet )</code> on page 239</td>
<td>This function sets the callback function called when there is a native error reported.</td>
</tr>
<tr>
<td><code>setTitle_CONT( screenTitle )</code> on page 240</td>
<td>Sets the title of the screen.</td>
</tr>
<tr>
<td><code>SharedStorage()</code> on page 240</td>
<td>Storage</td>
</tr>
<tr>
<td><code>showAlertDialog( message, [title] )</code> on page 240</td>
<td>Displays an alert dialog to the user.</td>
</tr>
<tr>
<td><code>showAttachmentContents_CONT( contents, mimeType, fileName, waitDialogCallbackString )</code> on page 241</td>
<td>Shows the given file contents in a content-appropriate way.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>showAttachmentFromCache_CONT(unique-Key, mimeType, fileName, waitDialogCallbackString)</code> on page 242</td>
<td>Shows the given file contents in a content-appropriate way.</td>
</tr>
<tr>
<td><code>showCertificatePicker()</code> on page 243</td>
<td>This function opens a form on the device that allows the user to specify the credentials for the use of certificate-based authentication.</td>
</tr>
<tr>
<td><code>showConfirmDialog(message, [title])</code> on page 243</td>
<td>Shows a confirm dialog to the user.</td>
</tr>
<tr>
<td><code>showLocalAttachment(key)</code> on page 244</td>
<td>Shows a local attachment.</td>
</tr>
<tr>
<td><code>showProgressDialog([message])</code> on page 244</td>
<td>This function shows a progress dialog with spinner.</td>
</tr>
<tr>
<td><code>showUrlInBrowser(url)</code> on page 245</td>
<td>This function opens the supplied URL in a browser.</td>
</tr>
<tr>
<td><code>shutdown()</code> on page 246</td>
<td>Shutdown the client connection to the SUP server.</td>
</tr>
<tr>
<td><code>startClient([onNotification])</code> on page 246</td>
<td>Start the client connection to the SUP server.</td>
</tr>
<tr>
<td><code>this.setIconUrl(processed)</code> on page 247</td>
<td>Gets the URL of this custom icon.</td>
</tr>
<tr>
<td><code>updateMessageProcessed(msgId, status)</code> on page 248</td>
<td>Updates the message processed status.</td>
</tr>
<tr>
<td><code>updateMessageRead(msgId, status)</code> on page 248</td>
<td>Updates the message read status.</td>
</tr>
</tbody>
</table>

Source

*Callbacks.js, line 15 on page 254.*

**hwc.SUPStorage class**

Storage

**Syntax**

```javascript
new SUPStorage(store)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>string</td>
<td>the store name</td>
</tr>
</tbody>
</table>
Example

```javascript
var store1 = new hwc.SUPStorage("one");
```

### Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BB7_MAX_STRING_STORAGE_LENGTH</code></td>
<td>A constant for the maximum length for a string being stored on BB7 BB7 cannot handle strings with length longer than 524000 This restriction applies to real devices as well as simulators.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clear()</code> on page 117</td>
<td>Storage</td>
</tr>
<tr>
<td><code>getItem(key)</code> on page 118</td>
<td>Storage</td>
</tr>
<tr>
<td><code>key(index)</code> on page 118</td>
<td>Storage</td>
</tr>
<tr>
<td><code>length()</code> on page 119</td>
<td>Storage</td>
</tr>
<tr>
<td><code>removeItem(key)</code> on page 119</td>
<td>Storage</td>
</tr>
<tr>
<td><code>setItem(key, value)</code> on page 120</td>
<td>Storage</td>
</tr>
</tbody>
</table>

### Source

`SUPStorage.js, line 40` on page 561.

**`BB7_MAX_STRING_STORAGE_LENGTH` member**

A constant for the maximum length for a string being stored on BB7 BB7 cannot handle strings with length longer than 524000 This restriction applies to real devices as well as simulators.

### Syntax

```javascript
<static> BB7_MAX_STRING_STORAGE_LENGTH
```

### Source

`SUPStorage.js, line 218` on page 567.

**`clear()` method**

Storage
**Syntax**

clear()

**Source**
*SUPStorage.js, line 302 on page 570.*

**getItem(key) method**

Storage

**Syntax**

ggetItem(key) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>String key corresponding to the requested value.</td>
</tr>
</tbody>
</table>

**Returns**

A String value corresponding to the key, or null if either the key is not known, or if the key exists but its value was set to null.

Type:

string

**Example**

// Create the SUP Storage
var store = new hwc.SUPStorage("one");
store.setItem("foo", "bar"); // add an item.
result = store.getItem("foo"); // will returns "bar".
result = store.getItem("fool"); // fool does not exists; will return null.

**Source**
*SUPStorage.js, line 170 on page 565.*

**key(index) method**

Storage

**Syntax**

key(index) {string}
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Integer</td>
<td>0-based index to the key. Must be less than the value retrieved by .length.</td>
</tr>
</tbody>
</table>

Returns
The key, or null if the index is invalid.

Type:
string

Example

```javascript
// Create the SUP Storage
var store = new hwc.SUPStorage ("one");
store.setItem ("foo", "bar"); // add an item.
var result = store.key (0); // will returns "foo".
```

Source

`SUPStorage.js`, line 97 on page 563.

`length()` method
Storage

Syntax

`length()`

Example

```javascript
// Create the SUP Storage
var store = new hwc.SUPStorage ("one");
store.setItem ("foo", "bar"); // add an item.
store.setItem ("fool", "bar"); // add an item.
store.setItem ("foo2", "bar"); // add an item.
var result = store.length; // result = 3
```

Source

`SUPStorage.js`, line 59 on page 562.

`removeItem(key)` method
Storage
Syntax
removeItem(key)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>String key to remove.</td>
</tr>
</tbody>
</table>

Example

// Create the SUP Storage
var store = new hwc.SUPStorage("one");
store.setItem("foo", "bar"); // add an item.
store.removeItem("foo");
result = store.getItem("food"); // will be null.

Source
SUPStorage.js, line 276 on page 569.

setItem(key, value) method
Storage

Syntax
setItem(key, value)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>String key corresponding to the value.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>String value to store.</td>
</tr>
</tbody>
</table>

Example

// Create the SUP Storage
var store = new hwc.SUPStorage("one");
store.setItem("foo", "bar"); // add an item.

Source
SUPStorage.js, line 233 on page 568.

hwc.SUPStorageException class
Storage
Syntax
new SUPStorageException( code, message )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Integer</td>
<td>the error code</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>the error message.</td>
</tr>
</tbody>
</table>

Source
SUPStorage.js, line 330 on page 571.

NativeErrorCodes namespace
This object contains constants representing the different types of public native error codes.

Error codes larger than 500 are reserved for server communication errors which may occur as the result of online requests and/or attachment downloads

Members

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACHMENT_NOT_DOWNLOADED on page 122</td>
<td>A constant indicating the attachment has not been downloaded.</td>
</tr>
<tr>
<td>CERTIFICATE_NOT_SELECTED on page 122</td>
<td>A constant indicating there was no certificate selected by the user.</td>
</tr>
<tr>
<td>DEVICE_NOT_CONNECTED on page 123</td>
<td>A constant indicating the device is not connected.</td>
</tr>
<tr>
<td>FAIL_TO_SAVE_CERTIFICATE on page 123</td>
<td>A constant indicating a failure to save a certificate.</td>
</tr>
<tr>
<td>FAIL_TO_SAVE_CREDENTIAL on page 123</td>
<td>A constant indicating a failure to save a credential.</td>
</tr>
<tr>
<td>FILENAME_NO_EXTENSION on page 123</td>
<td>A constant indicating there was a filename without an extension.</td>
</tr>
<tr>
<td>INVALID_COMMON_NAME on page 124</td>
<td>A constant indicating an invalid common name was passed while requesting a certificate from Afaria.</td>
</tr>
<tr>
<td>NAVIGATION_ERROR on page 124</td>
<td>A constant indicating that opening the URL failed.</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REQUIRED_PARAMETER_NOT_AVAILABLE on page 124</td>
<td>A constant indicating a required parameter was not available.</td>
</tr>
<tr>
<td>RESPONSE_TOO_LARGE on page 125</td>
<td>A constant indicating the response it too large for a javascript variable.</td>
</tr>
<tr>
<td>SSOCERT_EXCEPTION on page 125</td>
<td>A constant indicating there was an SSO certificate manager exception.</td>
</tr>
<tr>
<td>UNKNOWN_ERROR on page 125</td>
<td>A constant indicating there was an unknown error.</td>
</tr>
<tr>
<td>UNKNOWN_MIME_TYPE on page 125</td>
<td>A constant indicating there was an unknown MIME type.</td>
</tr>
<tr>
<td>UNSUPPORTED_ATTACHMENT_TYPE on page 126</td>
<td>A constant indicating the attachment type is not supported.</td>
</tr>
</tbody>
</table>

Source
hwc-comms.js, line 320 on page 444.

ATTACHMENT_NOT_DOWNLOADED member
A constant indicating the attachment has not been downloaded.

Syntax
<static> ATTACHMENT_NOT_DOWNLOADED : number

Type
number

Source
hwc-comms.js, line 338 on page 444.

CERTIFICATE_NOT_SELECTED member
A constant indicating there was no certificate selected by the user.

Syntax
<static> CERTIFICATE_NOT_SELECTED : number

Type
number

Source
hwc-comms.js, line 370 on page 445.
**DEVICE_NOT_CONNECTED member**
A constant indicating the device is not connected.

Syntax
<static> DEVICE_NOT_CONNECTED : number

Type
number

Source
hwc-comms.js, line 410 on page 447.

**FAIL_TO_SAVE_CERTIFICATE member**
A constant indicating a failure to save a certificate.

Syntax
<static> FAIL_TO_SAVE_CERTIFICATE : number

Type
number

Source
hwc-comms.js, line 402 on page 446.

**FAIL_TO_SAVE_CREDENTIAL member**
A constant indicating a failure to save a credential.

Syntax
<static> FAIL_TO_SAVE_CREDENTIAL : number

Type
number

Source
hwc-comms.js, line 394 on page 446.

**FILENAME_NO_EXTENSION member**
A constant indicating there was a filename without an extension.

Syntax
<static> FILENAME_NO_EXTENSION : number
INVALID_COMMON_NAME member
A constant indicating an invalid common name was passed while requesting a certificate from Afaria.

Syntax
<static> INVALID_COMMON_NAME : number

NAVIGATION_ERROR member
A constant indicating that opening the URL failed.

Syntax
<static> NAVIGATION_ERROR : number

REQUIRED_PARAMETER_NOT_AVAILABLE member
A constant indicating a required parameter was not available.

Syntax
<static> REQUIRED_PARAMETER_NOT_AVAILABLE : number
**Source**
hwc-comms.js, line 362 on page 445.

**RESPONSE_TOO_LARGE member**
A constant indicating the response it too large for a javascript variable.

**Syntax**
<static> RESPONSE_TOO_LARGE : number

**Type**
number

**Source**
hwc-comms.js, line 418 on page 447.

**SSOCERT_EXCEPTION member**
A constant indicating there was an SSO certificate manager exception.

**Syntax**
<static> SSOCERT_EXCEPTION : number

**Type**
number

**Source**
hwc-comms.js, line 386 on page 446.

**UNKNOWN_ERROR member**
A constant indicating there was an unknown error.

**Syntax**
<static, constant> UNKNOWN_ERROR : number

**Source**
hwc-comms.js, line 330 on page 444.

**UNKNOWN_MIME_TYPE member**
A constant indicating there was an unkown MIME type.

**Syntax**
<static> UNKNOWN_MIME_TYPE : number
**Type**

number

**Source**

*hwc-comms.js, line 346 on page 444.*

**UNSUPPORTED_ATTACHMENT_TYPE member**

A constant indicating the attachment type is not supported.

**Syntax**

<static> UNSUPPORTED_ATTACHMENT_TYPE : number

**Type**

number

**Source**

*hwc-comms.js, line 378 on page 445.*

**APP_ADDED member**

A constant indicating that a hybrid app has been added.

Used in *anonymous.ApplicationListener* on page 75 callback functions as a possible value for event.

**Syntax**

<static> APP_ADDED : number

**Type**

number

**Source**

*hwc-api.js, line 1640 on page 355.*

**APP_REFRESH member**

A constant indicating that the application list requires a refresh.

Used in *anonymous.ApplicationListener* on page 75 callback functions as a possible value for event.

**Syntax**

<static> APP_REFRESH : number
Type
number

Source
hwc-api.js, line 1634 on page 354.

**APP_REMOVED member**
A constant indicating that a hybrid app was removed.

Used in *anonymous.ApplicationListener* on page 75 callback functions as a possible value for event.

**Syntax**
<static> APP_REMOVED : number

Type
number

Source
hwc-api.js, line 1652 on page 355.

**APP_UPDATED member**
A constant indicating that a hybrid app was updated.

Used in *anonymous.ApplicationListener* on page 75 callback functions as a possible value for event.

**Syntax**
<static> APP_UPDATED : number

Type
number

Source
hwc-api.js, line 1646 on page 355.

**CONNECTED member**
Constant indicating that the hwc is connected.

Used in *anonymous.ConnectionStateListener* on page 77 callback functions.

**Syntax**
<static> CONNECTED : number
**Type**
number

**Source**
hwc-api.js, line 503 on page 313.

**CONNECTION_CONNECTED member**
A constant indicating that the log message is about the connection being established.
Used in anonymous.LogListener on page 79 callback functions.

**Syntax**
<static> CONNECTION_CONNECTED : number

**Type**
number

**Source**
hwc-api.js, line 873 on page 326.

**CONNECTION_DISCONNECTED member**
A constant indicating that the log message is about the connection being disconnected.
Used in anonymous.LogListener on page 79 callback functions.

**Syntax**
<static> CONNECTION_DISCONNECTED : number

**Type**
number

**Source**
hwc-api.js, line 878 on page 326.

**CONNECTION_ERROR member**
A constant indicating that the log message is about a connection error.
Used in anonymous.LogListener on page 79 callback functions.

**Syntax**
<static> CONNECTION_ERROR : number

**Type**
number
Source
hwc-api.js, line 863 on page 326.

**CONNECTION_OTHER** member
A constant indicating that the log message is not about the connection.
Used in *anonymous.LogListener* on page 79 callback functions.

Syntax
<static> CONNECTION_OTHER : number

Type
number

Source
hwc-api.js, line 868 on page 326.

**CONNECTION_RETRIEVED_ITEMS** member
a constant indicating that the log message is about retrieved items.
Used in *anonymous.LogListener* on page 79 callback functions.

Syntax
<static> CONNECTION_RETRIEVED_ITEMS : number

Type
number

Source
hwc-api.js, line 883 on page 327.

**DEFAULT_CUSTOM_ICON_INDEX** member
A constant indicating the custom icon index.

Syntax
<static> DEFAULT_CUSTOM_ICON_INDEX : number

Type
number

Source
hwc-api.js, line 1901 on page 364.
**DISCONNECTED** member
Constant indicating that the hwc is disconnected.

Used in `anonymous.ConnectionStateListener` on page 77 callback functions.

**Syntax**
```javascript
<static> DISCONNECTED : number
```

**Type**
number

**Source**
hwc-api.js, line 508 on page 313.

**e2eTrace** member
Represents an E2E Trace.

This object is used for debugging and analysis.

**Syntax**
```javascript
<static> e2eTrace
```

**Source**
hwc-api.js, line 3457 on page 420.

**isTraceEnabled()** method
Gets whether the e2e tracing has been requested to be started.

This function returns true between calls to `hwc.e2eTrace#startTrace` and `hwc.e2eTrace#stopTrace`.

**Syntax**
```javascript
<static> isTraceEnabled() {boolean}
```

**Returns**
True if trace is enabled, false otherwise.

**Type:**
boolean

**Source**
hwc-api.js, line 3491 on page 422.
setTraceLevel( The ) method
Sets the passport e2eTrace level.

This function must be called before hwc.e2eTrace#startTrace.

Syntax
<static> setTraceLevel( The )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>string</td>
<td>trace level. Must be one of hwc.e2eTrace.TraceLevel.LOW, hwc.e2eTrace.TraceLevel.MEDIUM, or hwc.e2eTrace.TraceLevel.HIGH.</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 3507 on page 422.

startTrace() method
Starts tracing user actions and requests.

Before this function is called, the trace level must be set with hwc.e2eTrace#setTracelevel.

Syntax
<static> startTrace()

Source
hwc-api.js, line 3520 on page 423.

stopTrace() method
Stops tracing user actions and requests.

Syntax
<static> stopTrace()

Source
hwc-api.js, line 3533 on page 423.
**uploadTrace() method**
Upload the Business Transaction XML (BTX) to the server.

To upload, the SAP Solution Manager URL must be set in SAP Control Center configuration.

**Syntax**

<static> uploadTrace() {boolean}

**Returns**
True if the upload is successful, false otherwise.

**Type:** boolean

**Source**
hwc-api.js, line 3548 on page 424.

**INSTALLATION_BEGIN member**
A constant indicating that the application is starting to be installed.

Used in anonymous.AppInstallationListener on page 73 callback functions.

**Syntax**

<static> INSTALLATION_BEGIN : number

**Type**
number

**Source**
hwc-api.js, line 1029 on page 332.

**INSTALLATION_END member**
A constant indicating that the application has finished being installed.

Used in anonymous.AppInstallationListener on page 73 callback functions.

**Syntax**

<static> INSTALLATION_END : number

**Type**
number

**Source**
hwc-api.js, line 1034 on page 332.
**MediaCache member**

 Represents a Media Cache.

This object gives the option to use the cache when accessing .

**Syntax**

```plaintext
<static> MediaCache
```

**Source**

`hwc-api.js, line 3379` on page 417.

**Policy member**

hwc.MediaCache.Policy An object containing constants representing the different caching policies.

**Syntax**

```plaintext
<static> Policy
```

**Source**

`hwc-api.js, line 3385` on page 418.

**CACHE_FIRST member**

hwc.MediaCache.Policy.CACHE_FIRST Use cache first policy: requests will be served from the cache if possible.

**Syntax**

```plaintext
<static> CACHE_FIRST : string
```

**Type**

string

**Source**

`hwc-api.js, line 3398` on page 418.

**SERVER_FIRST member**

hwc.MediaCache.Policy.SERVER_FIRST Use server first policy: requests will only be served from the cache if the server is unavailable.

**Syntax**

```plaintext
<static> SERVER_FIRST : string
```
**Type**

string

**Source**

hwc-api.js, line 3392 on page 418.

**getUrl( resourceUrl, [policy] ) method**

Creates a media cache URL for the resource.

The cache first policy will be used if no policy is specified.

**Syntax**

<static> getUrl( resourceUrl, [policy] ) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceUrl</td>
<td>string</td>
<td></td>
<td>The URL to the resource</td>
</tr>
</tbody>
</table>

**Returns**

The URL that can be used to access the resource with the specified caching policy.

**Type:**

string

**Example**

```javascript
// This line creates a url that can be used to retrieve the picture from the cache if possible, and from the server otherwise.
// The following function adds a picture to the page. Since the mediaCacheURL variable is used for the url, the picture will be retrieved from the cache if possible.
```
var addPicFromMediaCache = function() {
  // Create the image element.
  var image = document.createElement( "img" );
  // Set the source of the image to the media cache URL.
  image.setAttribute( 'src', mediaCacheURL );
  // Add the image element to the page.
  document.body.appendChild( image );
}

// This line creates a url that can be used to retrieve the picture
from the server if it is available, or the cache otherwise.
var mediaCacheURL_serverFirst = hwc.MediaCache.getUrl( "http://
yourserver.com/Pictures/pentagon.jpg",
hwc.MediaCache.Policy.SERVER_FIRST);
// The following function adds a picture to the page. Since the
mediaCacheURL_serverFirst variable is used for the url, the picture
will be gotten
// from the server if the server is available, and from the cache
otherwise.
var addPicFromMediaCache_ServerFirst = function() {
  // Create the image element.
  var image = document.createElement( "img" );
  // Set the source of the image to the media cache URL.
  image.setAttribute( 'src', mediaCacheURL_serverFirst );
  // Add the image element to the page.
  document.body.appendChild( image );
}

Source
hwc-api.js, line 3442 on page 420.

MSG_ADDED member
A constant indicating that a message has been added.
Used in anonymous.MessageListener on page 80 callback functions.

Syntax
<static> MSG_ADDED : number

Type
number

Source
hwc-api.js, line 2888 on page 399.

MSG_PRIORITY_HIGH member
A constant indicating a message has high priority.
Syntax
<static> MSG_PRIORITY_HIGH : number

Type
number

Source
hwc-api.js, line 2908 on page 400.

**MSG_PRIORITY_NORMAL** member
A constant indicating a message has normal priority.

Syntax
<static> MSG_PRIORITY_NORMAL : number

Type
number

Source
hwc-api.js, line 2903 on page 400.

**MSG_REFRESH** member
A constant indicating that a message needs to be refreshed.
Used in *anonymous.MessageListener* on page 80 callback functions.

Syntax
<static> MSG_REFRESH : number

Type
number

Source
hwc-api.js, line 2883 on page 399.

**MSG_REMOVED** member
A constant indicating that a message has been removed.
Used in *anonymous.MessageListener* on page 80 callback functions.

Syntax
<static> MSG_REMOVED : number
**MSG_UPDATED** member

A constant indicating that a message has been updated.

Used in *anonymous.MessageListener* on page 80 callback functions.

**Syntax**

```
<static> MSG_UPDATED : number
```

**NOTIFICATION_CANCEL** member

A constant indicating that no more push notification listeners should be called.

Used as a return value for *anonymous.PushNotificationListener* functions.

**Syntax**

```
<static> NOTIFICATION_CANCEL : number
```

**NOTIFICATION_CONTINUE** member

A constant indicating that other push notification listeners should continue to be called.

Used as a return value for *anonymous.PushNotificationListener* functions.

**Syntax**

```
<static> NOTIFICATION_CONTINUE : number
```
OPEN_APP_NOT_EXIST member
A constant indicating that hwc.openApp on page 221 failed because the specified app does not exist.
This is a possible return value for hwc.openApp on page 221.

Syntax
<static> OPEN_APP_NOT_EXIST : number

Type
number

Source
hwc-api.js, line 1332 on page 344.

OPEN_APP_OTHER member
A constant indicating that hwc.openApp on page 221 failed for an unspecified reason.
This is a possible return value for hwc.openApp on page 221.

Syntax
<static> OPEN_APP_OTHER : number

Type
number

Source
hwc-api.js, line 1848 on page 362.

OPEN_APP_SUCCESS member
A constant indicating that hwc.openApp on page 221 completed successfully.
This is a possible return value for hwc.openApp on page 221.

Syntax
<static> OPEN_APP_SUCCESS : number

Type
number
OPEN_MSG_APP_NOT_EXIST member
A constant indicating that a message could not be opened because there was no associated hybrid app.
This is a possible return value for hwc.openMessage on page 222.

Syntax
<static> OPEN_MSG_APP_NOT_EXIST : number

Type
number

Source
hwc-api.js, line 1842 on page 362.

OPEN_MSG_NOT_EXIST member
A constant indicating that a message could not be opened because no message with the given ID exists.
This is a possible return value for hwc.openMessage on page 222.

Syntax
<static> OPEN_MSG_NOT_EXIST : number

Type
number

Source
hwc-api.js, line 3156 on page 409.

OPEN_MSG_OTHER member
A constant indicating that a message could not be opened due to an unspecified error.
This is a possible return value for hwc.openMessage on page 222.

Syntax
<static> OPEN_MSG_OTHER : number

Type
number

Source
hwc-api.js, line 3150 on page 409.
**OPEN_MSG_SUCCESS member**
A constant indicating that a message was successfully opened.

This is a possible return value for `hwc.openMessage` on page 222.

**Syntax**

```typescript
<static> OPEN_MSG_SUCCESS : number
```

**Type**

number

**Source**

`hwc-api.js`, line 3144 on page 409.

**perf member**

Represents the Performance Manager.

**Syntax**

```typescript
<static> perf
```

**Example**

```javascript
// Start performance collection.
if (hwc.perf.isEnabled())
{
    hwc.perf.stopInteraction();
}

hwc.perf.startInteraction('someinteraction');

hwc.perf.startInterval('IntervalName', 'CustomType'); // Start an optional interval.

// Stop performance collection. Logs will be written.
if (hwc.perf.isEnabled())
{
    hwc.perf.stopInterval('IntervalName'); // Stop an optional interval.
    hwc.perf.stopInteraction();
}
```

**Source**

`hwc-api.js`, line 3579 on page 425.
**isEnabled() method**
Gets whether the performance agent is enabled.

**Syntax**
<static> isEnabled() {boolean}

**Returns**
True if the performance agent is enabled, false otherwise.

**Type:**
boolean

**Source**
*hwc-api.js, line 3586 on page 425.*

**startInteraction(interactionName) method**
Starts the interaction.

**Syntax**
<static> startInteraction(interactionName)

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interactionName</td>
<td>string</td>
<td>The name of the interaction.</td>
</tr>
</tbody>
</table>

**Source**
*hwc-api.js, line 3600 on page 426.*

**startInterval(intervalName, intervalType) method**
Starts an interval.

**Syntax**
<static> startInterval(intervalName, intervalType)

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intervalName</td>
<td>string</td>
<td>The name of the interval.</td>
</tr>
</tbody>
</table>

| intervalType | string | The type of the interval.    |
**stopInteraction() method**
Stops the interaction.

**Syntax**

```
<static> stopInteraction()
```

**stopInterval( intervalName ) method**
Stops the interval.

**Syntax**

```
<static> stopInterval( intervalName )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intervalName</td>
<td>string</td>
<td>The name of the interval.</td>
</tr>
</tbody>
</table>

**PictureError member**
An array that holds all possible error codes

**Syntax**

```
<static> PictureError
```

**Source**

`Camera.js, line 83 on page 260.`

**BAD_OPTIONS member**
Constant indicating that the supplied options were not recognized by the `hwc.getPicture` on page 192 method
Syntax
<static> BAD_OPTIONS

Source
Camera.js, line 113 on page 261.

IN_PROGRESS member
Constant indicating that the hwc.getPicture on page 192 method has been invoked, but has not completed yet.

Syntax
<static> IN_PROGRESS

Source
Camera.js, line 101 on page 260.

NO_ERROR member
Constant indicating that the hwc.getPicture on page 192 method was successful.

Syntax
<static> NO_ERROR

Source
Camera.js, line 89 on page 260.

NOT_SUPPORTED member
Constant indicating that the hwc.getPicture on page 192 method is not implemented, camera not present, etc.

Syntax
<static> NOT_SUPPORTED

Source
Camera.js, line 95 on page 260.

TOO_LARGE member
Constant indicating that the returned image size was too large to be handled by JavaScript.
Syntax
<static> TOO_LARGE

Source
Camera.js, line 119 on page 261.

UNKNOWN member
Constant indicating that an unknown error occurred during the execution of hwc.getPicture on page 192 method.

Syntax
<static> UNKNOWN

Source
Camera.js, line 125 on page 261.

USER_REJECT member
Constant indicating that the user has cancelled the hwc.getPicture on page 192 invocation.

Syntax
<static> USER_REJECT

Source
Camera.js, line 107 on page 260.

REG_ERR_AUTO_REG_NOT_ENABLED member
Constant indicating that auto registration was not enabled in the template.
Possible return value for hwc.saveSettings on page 237.

Syntax
<static> REG_ERR_AUTO_REG_NOT_ENABLED : number

Type
number

Source
hwc-api.js, line 144 on page 299.
**REG_ERR_AUTO_REG_TEMPLATE_NOT_FOUND** member

Constant indicating that no MBS template was found for given AppId and/or Security configuration.

Possible return value for `hwc.saveSettings` on page 237.

**Syntax**

```typescript
<static> REG_ERR_AUTO_REG_TEMPLATE_NOT_FOUND : number
```

**Type**

number

**Source**

`hwc-api.js`, line 140 on page 299.

---

**REG_ERR_AUTO_REG_USER_NAME_TOO_LONG** member

Constant indicating that the user name is longer than the legal limit.

Possible return value for `hwc.saveSettings` on page 237.

**Syntax**

```typescript
<static> REG_ERR_AUTO_REG_USER_NAME_TOO_LONG : number
```

**Type**

number

**Source**

`hwc-api.js`, line 152 on page 299.

---

**REG_ERR_AUTO_REG_WRONG_USER_FOR_DEVICE** member

Constant indicating that the given device id is already registered for another user.

Possible return value for `hwc.saveSettings` on page 237.

**Syntax**

```typescript
<static> REG_ERR_AUTO_REG_WRONG_USER_FOR_DEVICE : number
```

**Type**

number

**Source**

`hwc-api.js`, line 148 on page 299.

---
**REG_ERR_COULD_NOT_REACH_MMS_SERVER** member

Constant indicating that the connection to the MMS service failed.

Possible return value for *hwc.saveSettings* on page 237.

**Syntax**

<static> REG_ERR_COULD_NOT_REACH_MMS_SERVER : number

**Type**

dnumber

**Source**

*hwc-api.js*, line 136 on page 299.

**REG_ERR_INVALID_USER_NAME** member

Constant indicating that the user name contains invalid characters.

Possible return value for *hwc.saveSettings* on page 237.

**Syntax**

<static> REG_ERR_INVALID_USER_NAME : number

**Type**

dnumber

**Source**

*hwc-api.js*, line 156 on page 300.

**REG_ERR_MMS_AUTHENTICATION_FAILED** member

Constant indicating that MMS Authentication failed.

Possible return value for *hwc.saveSettings* on page 237.

**Syntax**

<static> REG_ERR_MMS_AUTHENTICATION_FAILED : number

**Type**

dnumber

**Source**

*hwc-api.js*, line 132 on page 298.
**REGISTRATION_METHOD_AFARIA member**
Constant indicating that automatic registration using a certificate from Afaria is the preferred method.

Used in `hwc.ConnectionSettings` on page 168.

Syntax

```
<static> REGISTRATION_METHOD_AFARIA : number
```

**Type**
number

**Source**
`hwc-api.js`, line 42 on page 295.

**REGISTRATION_METHOD_AUTOMATIC member**
Constant indicating that automatic registration using password is the preferred method.

Used in `hwc.ConnectionSettings` on page 168.

Syntax

```
<static> REGISTRATION_METHOD_AUTOMATIC : number
```

**Type**
number

**Source**
`hwc-api.js`, line 34 on page 295.

**REGISTRATION_METHOD_CERTIFICATE member**
Constant indicating that automatic registration using a local certificate is the preferred method.

Used in `hwc.ConnectionSettings` on page 168.

Syntax

```
<static> REGISTRATION_METHOD_CERTIFICATE : number
```

**Type**
number

**Source**
`hwc-api.js`, line 46 on page 295.
**REGISTRATION_METHOD_MANUAL member**
Constant indicating that manual registration is the preferred method.

Used in `hwc.ConnectionSettings` on page 168.

**Syntax**

```
<static> REGISTRATION_METHOD_MANUAL : number
```

**Type**

number

**Source**

`hwc-api.js, line 38` on page 295.

**REGISTRATION_METHOD_NO_PREFERENCE member**
Constant indicating no registration method preference.

The application implementation decides the default method to use. This is handled as Manual registration by the HWC. Used in `hwc.ConnectionSettings` on page 168.

**Syntax**

```
<static> REGISTRATION_METHOD_NO_PREFERENCE : number
```

**Type**

number

**Source**

`hwc-api.js, line 30` on page 295.

**SETTING_SUCCESS member**
Constant indicating `hwc.saveSettings` on page 237 completed successfully.

Possible return value for `hwc.saveSettings` on page 237.

**Syntax**

```
<static> SETTING_SUCCESS : number
```

**Type**

number

**Source**

`hwc-api.js, line 160` on page 300.
**STATUS member**  
This object contains constants representing the status of the hybrid app.

**Syntax**  
<static> STATUS

**Source**  
`hwc-comms.js`, line 166 on page 438.

**activationRequired() method**  
This function sets the activation required state of this hybrid app to true.

After calling this function, the current hybrid app will need to be activated.

**Syntax**  
<static> activationRequired()

**Example**  
```javascript
hwc.activationRequired();
```

**Source**  
`hwc-comms.js`, line 773 on page 459.

**addAppInstallationListener( AppInstallationListener ) method**  
Register the application installation listener.

**Syntax**  
<static> addAppInstallationListener( AppInstallationListener )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppInstallationListener</td>
<td>anonymous.AppInstallationListener on page 73</td>
<td>A callback for application installation changes.</td>
</tr>
</tbody>
</table>

**Example**  
```javascript
// appInstallListener is the callback function that will be passed to hwc.addAppInstallationListener.
var appInstallListener = function( event, moduleId, version, moduleName )
{
    if( event == hwc.INSTALLATION_BEGIN )
    {
```
alert(moduleName + " has just started the installation process.");
}
else if( event == hwc.INSTALLATION_END )
{
    alert(moduleName + " has just finished the installation process.");
}
}

hwc.addAppInstallationListener( appInstallListener );

Source

hwc-api.js, line 946 on page 329.

addAppListener( ApplicationListener, [containingObject] ) method

Register the application listener.

Syntax

<static> addAppListener( ApplicationListener, [containingObject] )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationListener</td>
<td>anonymous.ApplicationListener on page 75</td>
<td></td>
<td>The callback function for application changes.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>The containing object of the listener method. This parameter is only required if the ApplicationListener references the containing object.</td>
</tr>
</tbody>
</table>

Example

// This is the callback function that will be passed to hwc.addAppListener.
var appListener = function( event, moduleId, version )
{
    if( event == hwc.APP_ADDED )
    {
        alert("A hybrid app has been added.");
    }
}

hwc.addAppListener( appListener );
// appListenerManager is an object that will contain the callback function as well as variables
// the callback function references.
var appListenerManager = {};
// doSomething is a function that is called from inside the callback function.
appListenerManager.doSomething = function( event )
{
    if( event == hwc.APP_REMOVED )
    {
        alert("A hybrid app has been removed.");
    }
}
// This is the callback function that will be passed to hwc.addAppListener. It calls doSomething,
// the definition of which is in the containing function.
appListenerManager.listener = function( event, moduleId, version )
{
    this.doSomething( event );
}
// Since the listener callback function references a variable from its containing object,
// the containing object must be passed to hwc.addAppListener.
hwc.addAppListener( appListenerManager.listener, appListenerManager );

Source
hwc-api.js, line 1539 on page 351.

addConnectionListener( ConnectionStateListener, [containingObject] ) method
Register the connection state listener.

Syntax
<static> addConnectionListener( ConnectionStateListener, [containingObject] )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionStateListener</td>
<td>anonymous.ConnectionStateListener on page 77</td>
<td></td>
<td>Callback for connection state changes.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object containing definition for ConnectionStateListener. If a connection state callback function references variables in its containing object, then the containing object should be passed to this function.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```javascript
// doSomething is a global function that gets called from the connection listener.
var doSomething = function() {
  alert("sample function that gets executed when the hwc becomes connected");
}
// connectionListener is the callback function that is given to addConnectionListener.
// When there is a connection event, connectionListener will be invoked with the details.
var connectionListener = function( event, errorCode, errorMessage ) {
  if( event == hwc.CONNECTED )
  {
    doSomething();
  }
}
hwc.addConnectionListener( connectionListener );

// connectionStateManager is an object that will contain the connection listener callback as well as 
// a variable used by the callback.
var connectionStateManager = {};
// The connectionStateManager keeps track of whether the HWC is connected or not.
connectionStateManager.connected = false;
// A function called by the listener.
connectionStateManager.doSomething = function() {
  if( this.connected )
  {
    alert("this alert gets displayed if the hwc is connected");
  }

// This is the callback function that will be passed to addConnectionListener. This callback references variables 
// from the containing object (this.connected and this.doSomething), so when we call addConnectionListener we have 
// to give the containing object as the second parameter.
```
connectionStateManager.listener = function( event, errorCode, errorMessage )
{
    if( event == hwc.CONNECTED )
    {
        this.connected = true;
    }
    else
    {
        this.connected = false;
    }
    this.doSomething();
}
// Pass both the listener and the containing object. This enables
the listener to refer to variables in the containing object when it
is invoked.
hwc.addConnectionListener( connectionStateManager.listener, connectionStateManager );

Source
hwc-api.js, line 369 on page 308.

addLogListener( LogListener, [containingObject] ) method
Register the log listener.

Syntax
<static> addLogListener( LogListener, [containingObject] )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogListener</td>
<td>anonymous.LogListener on page 79</td>
<td></td>
<td>Callback for changes to the log.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>Object containing definition for LogListener. If a log listener callback function references variables in its containing object, then the containing object should be passed to this function.</td>
</tr>
</tbody>
</table>

Example

// A global function called by the log listener.
var doSomething = function()
{

alert("this gets displays when there is a log event.");
}

// The log listener callback function that will be passed to
hwc.addLogListener.
// This function will be invoked whenever there is a log event.
var logListener = function( event, errorCode, errorMessage )
{
    doSomething();
}

// Add the log listener.
hwc.addLogListener( logListener );

// logListenerManager is an object that will contain the listener
// callback as well
// as a function that will be invoked from the listener callback
// function.
var logListenerManager = {};
// This is a function that is called from the listener callback.
logListenerManager.doSomething = function()
{
    alert("this gets displays when there is a log event.");
}
// This is the listener callback that will be passed to
hwc.addLogListener.
// Since a variable is referenced from the containing object, the
// containing object
// will need to be passed to hwc.addLogListener.
logListenerManager.listener = function( event, errorCode, errorMessage )
{
    this.doSomething();
}
// Pass both the listener callback and the containing object.
hwc.addLogListener( logListenerManager.listener,
logListenerManager );

Source
hwc-api.js, line 757 on page 322.

addMenuItemCollection( collection ) method
This function adds a menu item collection to the menu items for the screen.

Syntax
<static> addMenuItemCollection( collection )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>hwc.MenuItemCollection on page 212</td>
<td>The collection of menu items to add to the screen.</td>
</tr>
</tbody>
</table>
**Example**

```javascript
var callbackFunctionName = function() {
    alert( "Menu item clicked!" );
}
var menuItemCollection = new hwc.MenuItemCollection();
menuItemCollection.addMenuItem("menu item name", "callbackFunctionName()");
hwc.addMenuItemCollection( menuItemCollection );
```

**Source**

*hwc-comms.js*, line 669 on page 455.

**addMessageListener( filters, MessageListener, [containingObject] ) method**

Registers a message listener.

**Syntax**

<static> addMessageListener( filters, MessageListener, [containingObject] )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filters</td>
<td>hwc.MessageFilter on page 221</td>
<td></td>
<td>The message filter that message events must pass to get passed to the anonymous.Message-Listener on page 80. If no filter is desired, then null can be used for this parameter.</td>
</tr>
<tr>
<td>MessageListener</td>
<td>anonymous.MessageListener on page 80</td>
<td></td>
<td>The callback function for message changes.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>The containing object of the message listener. If a message listener callback function references variables in its containing object, then the containing object should be passed to this function.</td>
</tr>
</tbody>
</table>
Example

// soSomething is a global function called by the listener callback.
var doSomething = function()
{
    alert("New message!");
}
// messageListener is the callback function passed to
hwc.addMessageListener.
var messageListener = function( flag, messageId )
{
    if( flag == hwc.MSG_ADDED )
    {
        doSomething();
    }
}
// We do not want to filter the message events the listener will get
invoked for, so pass null for the first parameter.
hwc.addMessageListener( null, messageListener );

// someObject is an object that will contain the listener callback as
well as a variable referenced by the callback.
var someObject = {};
// doSomething is a function referenced by the callback function.
someObject.doSomething = function()
{
    alert("New message!");
}
// messageListener is the callback that will be passed to
hwc.addMessageListener.
someObject.messageListener = function( flag, messageId )
{
    if( flag == hwc.MSG_ADDED )
    {
        this.doSomething();
    }
}
// Create a filter so that not all message events will invoke our
callback function.
// Only events about messages with a subject of "Subject" will
trigger our callback function.
var filter = new hwc.MessageFilter( null, "Subject", null, null, null, null);
// The callback function references a variable in its containing
object, so we need to pass in the containing object
// in addition to the filter and the callback function.
hwc.addMessageListener( filter, someObject.messageListener, someObject );

Source

hwc-api.js, line 2781 on page 395.
addPushNotificationListener( PushNotificationListener, [containingObject] )

*method*

Register a push notification listener.

**Syntax**

<static> addPushNotificationListener( PushNotificationListener, [containingObject] )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PushNotificationLis-</td>
<td>function</td>
<td></td>
<td>The callback for push notifications.</td>
</tr>
<tr>
<td>tener</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>Object containing definition for PushNotificationListener. If the listener callback function references variables in its containing object, then the containing object should be passed to this function.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
// pushListener is the callback function that will be passed to hwc.addPushNotificationListener.
var pushListener = function( notifications )
{
  alert( "push notification:\" + JSON.stringify(notifications) );
  return hwc.NOTIFICATION_CONTINUE;
}
hwc.addPushNotificationListener( pushListener );

// pushListenerManager is an object that will contain the listener callback as well as a variable referenced from the callback.
var pushListenerManager = {};
// doSomething is a function that is called from inside the callback.
pushListenerManager.doSomething = function( notifications )
{
  alert( "push notification:\" + JSON.stringify(notifications) );
  return hwc.NOTIFICATION_CONTINUE;
}
```

---

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps
// This is the callback function.
pushListenerManager.listener = function( notifications )
{
  return this.doSomething( notifications );
}
// Since the callback function references variables in its containing
// object, the containing object
// must be passed to hwc.addPushNotificationListener as well.
hwc.addPushNotificationListener( pushListenerManager.listener,
pushListenerManager );

Source
hwc-api.js, line 1242 on page 340.

CertificateStore() method
Use these 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 Workflow 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.

Syntax
<static> CertificateStore()

Source
Certificate.js, line 45 on page 270.

certificateLabels( filterSubject, filterIssuer ) method
Certificate

Syntax
certificateLabels( filterSubject, filterIssuer ) [String[]]

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterSubject</td>
<td>String</td>
<td>filter of subject</td>
</tr>
<tr>
<td>filterIssuer</td>
<td>String</td>
<td>filter of issuer</td>
</tr>
</tbody>
</table>
**Returns**
Only filtered certificate labels

Type:

String[]

**Example**

// 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");

**Source**

*Certificate.js, line 112 on page 272.*

**getDefault() method**

Certificate

**Syntax**

<static> getDefault() {hwc.CertificateStore}

**Returns**

a certificate without the signedCertificate part set

Type:

*hwc.CertificateStore* on page 158

**Source**

*Certificate.js, line 144 on page 273.*

**getPublicCertificate( label ) method**

Certificate

**Syntax**

getPublicCertificate( label )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>String</td>
<td>label of the desired certificate</td>
</tr>
</tbody>
</table>
Returns
certificate object

Example

```javascript
// 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]);
```

Source
Certificate.js, line 164 on page 274.

getSignedCertificate( label, password ) method
Certificate

Syntax
getSignedCertificate( label, password )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>String</td>
<td>label of the desired certificate</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>Access password for the private key of the certificate. Pass null unless the platform requires a password.</td>
</tr>
</tbody>
</table>

Returns
Certificate object

Example

```javascript
// 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;
```

Source
Certificate.js, line 209 on page 276.
getSignedCertificateFromAfaria( commonName, challengeCode ) method

Certificate

Syntax

getSignedCertificateFromAfaria( commonName, challengeCode )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commonName</td>
<td>String</td>
<td>Common name used to generate the certificate by Afaria</td>
</tr>
<tr>
<td>challengeCode</td>
<td>String</td>
<td>Challenge code for the user so that CA can verify and sign it</td>
</tr>
</tbody>
</table>

Throws

- If called on a platform that is not supported.

Returns

JSON object with CertBlob in Base64 encoded format and other information about certificate

Example

// The following script gets a signed certificate from the Afaria server.
var certStore = CertificateStore.getDefault();
cert = certStore.getSignedCertificateFromAfaria("Your_CN", "CA_challenge_code");

Source

Certificate.js, line 362 on page 282.

getSignedCertificateFromFile( filePath, password ) method

Certificate

Syntax

getSignedCertificateFromFile( filePath, password )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filePath</td>
<td>String</td>
<td>The absolute path to the file.</td>
</tr>
</tbody>
</table>
password | String | The password needed to access the certificate's private data.

Example

```javascript
// 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");
```

Source
Certificate.js, line 287 on page 279.

getSignedCertificateFromServer( username, serverPassword, certPassword )

method
Certificate

Syntax

getSignedCertificateFromServer( username, serverPassword, certPassword )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>String</td>
<td>The username for the Windows user (in the form &quot;DOMAIN\username&quot;)</td>
</tr>
<tr>
<td>serverPassword</td>
<td>String</td>
<td>The password for the Windows user</td>
</tr>
<tr>
<td>certPassword</td>
<td>String</td>
<td>The password needed to access the certificate (may be the same or different from the Windows password)</td>
</tr>
</tbody>
</table>

Example

```javascript
// 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");
```
Certificate.js, line 325 on page 280.

`listAvailableCertificatesFromFileSystem(sFolder, sFileExtension) method` Certificate

**Syntax**

```javascript
listAvailableCertificatesFromFileSystem(sFolder, sFileExtension) → [String]
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sFolder</td>
<td>String</td>
<td>Folder in which to search for files. This should be a full absolute path, based on the root of the device file system. The separator may be either &quot;/&quot; or &quot;&quot;. For example, &quot;sdcard\mycerts&quot; or &quot;sdcard/mycerts&quot; is acceptable. Do not include any http prefixes, such as &quot;file: &quot;.</td>
</tr>
<tr>
<td>sFileExtension</td>
<td>String</td>
<td>File extension to which the list should be restricted. Pass the string expected after the &quot;.&quot; in the file name. For example, to match *.p12, pass &quot;p12&quot; as the argument. Pass null to return all files in the folder.</td>
</tr>
</tbody>
</table>

**Returns**

A list of Strings, each String being the full path name of a matched file in the given folder.

Type:

String[]

**Example**

```javascript
// 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");
```
**clearCache() method**
This function clears the contents of the on-device request result cache for the current hybrid app.

**Syntax**
<static> clearCache()

**Example**
```javascript
hwc.clearCache();
```

**clearCacheItem(cachekey) method**
This function clears an item from the contents of the on-device request result cache for the current hybrid app.

**Syntax**
<static> clearCacheItem(cachekey)

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cachekey</td>
<td>string</td>
<td>The key for the cache item to be removed. This is the same key that was passed to hwc.doOnlineRequest.</td>
</tr>
</tbody>
</table>

**Example**
```javascript
// The cache key is set when calling hwc.doOnlineRequest_CONT
hwc.doOnlineRequest(..., ..., ..., ..., ..., ..., cacheKey, ..., ...);
// At some later point if we want to clear the cache for the above request, we use the following code:
hwc.clearCacheItem(cacheKey);
```

**Source**
*hwc-comms.js, line 860 on page 462.*
ClientVariables( clientVariablesVersion, clientVariableItems ) method
Represents a ClientVariables object.

Syntax
<static> ClientVariables( clientVariablesVersion, clientVariableItems )

Parameters
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientVariablesVersion</td>
<td>number</td>
<td>The version of client variables.</td>
</tr>
<tr>
<td>clientVariableItems</td>
<td>Object</td>
<td>The json object that contains key/value pairs of client variable items.</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 2000 on page 368.

ITEM_NOT_FOUND member
A constant indicating that a variable does not exist in a hwc.ClientVariables on page 165 object.

Syntax
<static> ITEM_NOT_FOUND : number

Type
number

Source
hwc-api.js, line 2120 on page 372.

this.containsName( variableName ) method
Check if this hwc.ClientVariables on page 165 has a variable by the given name.

Syntax
<static> this.containsName( variableName ) {boolean}

Parameters
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableName</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>variableName</td>
<td>string</td>
<td>The name of variable to check for.</td>
</tr>
</tbody>
</table>

**Returns**
True if this *hwc.ClientVariables* on page 165 has a variable by the given name, false otherwise.

Type:
boolean

**Source**
hwc-api.js, line 2070 on page 370.

**this.getAllVariableNames() method**
Gets an array containing the names of all variables in this *hwc.ClientVariables* on page 165.

**Syntax**
<static> this.getAllVariableNames() {string[]}

**Returns**
The array holding the names of all variables contained in this *hwc.ClientVariables* on page 165.

Type:
string[]

**Source**
hwc-api.js, line 2036 on page 369.

**this.getCount() method**
Gets the number of variables this *hwc.ClientVariables* on page 165 contains.

**Syntax**
<static> this.getCount() {number}

**Returns**
The number of variables.

Type:
number
**Source**

hwc-api.js, line 2022 on page 369.

**this.getVariableValueByName( variableName ) method**

Gets the value of the variable with the given name.

If this hwc.ClientVariables on page 165 does not have a variable by the given name, a hwc.ClientVariableException on page 168 will be thrown.

**Syntax**

<static> this.getVariableValueByName( variableName ) \{string\}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variableName</td>
<td>string</td>
<td>The name of the variable to get the value of.</td>
</tr>
</tbody>
</table>

**Throws**

- This exception is thrown when there is no variable by the given name in this hwc.ClientVariables on page 165.
- Type:
  hwc.ClientVariableException

**Returns**

The value of the variable.

Type:

string

**Source**

hwc-api.js, line 2092 on page 371.

**this.getVersion() method**

Gets the version of the client variables.

**Syntax**

<static> this.getVersion() \{number\}

**Returns**

The version of the client variables.

Type:
number

Source
hwc-api.js, line 2011 on page 368.

ClientVariablesException(errCode, errMsg) method
This exception is thrown when hwc.ClientVariables#getVariableValueByName is called with a variable name that does not exist.

Syntax
<static> ClientVariablesException( errCode, errMsg )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errCode</td>
<td>number</td>
<td>The error code (will be hwc.ClientVariables.ITEM_NOT_FOUND on page 165).</td>
</tr>
<tr>
<td>errMsg</td>
<td>string</td>
<td>A message describing the error.</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 2111 on page 372.

close() method
This function closes the hybrid app.

Syntax
<static> close()

Example
hwc.close();

Source
hwc-comms.js, line 1564 on page 488.

ConnectionSettings( regmethod, server, port, server, user, activationcode, protocol, password, urlsuffix ) method
Represents the connection settings for connecting to the SUP Server.
Used in hwc.loadSettings on page 207 and hwc.saveSettings on page 237.
## Syntax

<static> `ConnectionSettings( regmethod, server, port, server, user, activationcode, protocol, password, urlsuffix )`

## Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regmethod</td>
<td>number</td>
<td>A number representing the registration method (must be one of <code>hwc.REGISTRATION_METHOD_NO_PREFERENCE</code> on page 148, <code>hwc.REGISTRATION_METHOD_MANUAL</code> on page 148, <code>hwc.REGISTRATION_METHOD_AUTOMATIC</code> on page 147, <code>hwc.REGISTRATION_METHOD_AFARIA</code> on page 147, <code>hwc.REGISTRATION_METHOD_CERTIFICATE</code> on page 147).</td>
</tr>
<tr>
<td>server</td>
<td>string</td>
<td>The SUP/Relay server name.</td>
</tr>
<tr>
<td>port</td>
<td>number</td>
<td>The SUP/Relay server port number.</td>
</tr>
<tr>
<td>server</td>
<td>string</td>
<td>The farm id.</td>
</tr>
<tr>
<td>user</td>
<td>string</td>
<td>The user name.</td>
</tr>
<tr>
<td>activationcode</td>
<td>string</td>
<td>The activation code.</td>
</tr>
</tbody>
</table>
| protocol      | string | The protocol to use. Must be "HTTP" or "HTTPS".
| password      | string | The password for automatic registration.                                                                                                   |
| urlsuffix     | string | The url suffix (used only when connecting to a relay server).                                                                 |

## Example

```javascript
// Create a new ConnectionSettings object.
var connectionSettings = new
hwc.ConnectionSettings( hwc.REGISTRATION_METHOD_MANUAL,
"999.999.999.999",
```

---

Developer Guide: Hybrid Apps
5001,
0,
"sampleUsername",
123,
"HTTP",
"samplePassword",
"/" );

// Use the ConnectionSettings object we just created to set the connection settings.
hwc.saveSettings( connectionSettings );

Source
hwc-api.js, line 79 on page 297.

**connectToServer( [onNotification] ) method**
Resumes the connection to the SUP server.

Companion function to `hwc.disconnectFromServer` on page 175. This function should only be called after the connection to the SUP server has been suspened with a call to `hwc.disconnectFromServer` on page 175.

**Syntax**
<static> connectToServer([onNotification])

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onNotification</code></td>
<td><code>anonymous.LogListener</code> on page 79</td>
<td>(optional)</td>
<td>A log listener callback function. If you are interested in the connection state it is recommended that you call <code>hwc.addConnectionListener</code> on page 151 before calling <code>hwc.connectToServer</code>.</td>
</tr>
</tbody>
</table>

**Example**

hwc.connectToServer();

// Add a log listener while calling hwc.connectToServer.
var logListener = function( time, event, message )
{
    alert(message);
}
hwc.connectToServer( logListener );
Source
hwc-api.js, line 601 on page 316.

**convertLocalTimeToUtc( date ) method**
Timezone

**Syntax**

<static> `convertLocalTimeToUtc( date ) {Date}`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date to be converted, initialized to some valid local time.</td>
</tr>
</tbody>
</table>

**Returns**

Returns the converted Date object.

Type:
Date

**Example**

```javascript
var utcDate = hwc.convertLocalTimeToUtc( date );
```

Source
Timezone.js, line 238 on page 582.

**convertUtcToLocalTime( date ) method**
Timezone

**Syntax**

<static> `convertUtcToLocalTime( date ) {Date}`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date to be converted, initialized to some valid UTC time.</td>
</tr>
</tbody>
</table>

**Returns**

Returns the converted Date object.
Type:
Date

Example
var localDate = hwc.convertUtcToLocalTime( date );

Source
Timezone.js, line 210 on page 581.

CustomIcon( width, height, type, name, path, processedPath, moduleId, moduleVersion, index ) method
Represents a CustomIcon.

Used with the hwc.HybridApp on page 197 object.

Syntax
<static> CustomIcon( width, height, type, name, path, processedPath, moduleId, moduleVersion, index )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>number</td>
<td>The width of this custom icon.</td>
</tr>
<tr>
<td>height</td>
<td>number</td>
<td>The height of this custom icon.</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>The image type of this custom icon.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of this custom icon.</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>The file path of the unprocessed icon.</td>
</tr>
<tr>
<td>processedPath</td>
<td>string</td>
<td>The file path of the processed icon.</td>
</tr>
<tr>
<td>moduleId</td>
<td>number</td>
<td>The module ID of the hybrid app this icon is for.</td>
</tr>
<tr>
<td>moduleVersion</td>
<td>number</td>
<td>The module version of the hybrid app this icon is for.</td>
</tr>
<tr>
<td>index</td>
<td>number</td>
<td>The index of this custom icon.</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 2137 on page 373.
**this.getHeight() method**  
Gets the height of this custom icon.

**Syntax**

<static> this.getHeight() {number}

**Returns**  
The height of this custom icon.

Type:  
number

**Source**  
hwc-api.js, line 2166 on page 374.

**this.getImagePath() method**  
Gets the file path of the unprocessed icon.

**Syntax**

<static> this.getImagePath() {string}

**Returns**  
The file path of the unprocessed icon.

Type:  
string

**Source**  
hwc-api.js, line 2199 on page 375.

**this.getName() method**  
Gets the name of this custom icon.

**Syntax**

<static> this.getName() {string}

**Returns**  
The name of this custom icon.

Type:  
string
`Source
hwc-api.js, line 2188 on page 375.

`this.getProcessedImagePath()` method
Gets the file path of the processed icon.

`Syntax
<static> this.getProcessedImagePath() {string}

`Returns
The file path of the processed icon.
Type:
string

`Source
hwc-api.js, line 2210 on page 375.

`this.getType()` method
Gets the image type of this custom icon.

`Syntax
<static> this.getType() {string}

`Returns
The file type of the image.
Type:
string

`Source
hwc-api.js, line 2177 on page 374.

`this.getWidth()` method
Gets the width of this custom icon.

`Syntax
<static> this.getWidth() {number}

`Returns
The width of this custom icon.
**disconnectFromServer() method**
Suspends the connection to the SUP server.

Companion function to hwc.connectToServer on page 170.

**Syntax**
<static> disconnectFromServer()

**Example**
```javascript
hwc.disconnectFromServer();
```

**expireCredentials() method**
Allows the user to set the credentials to the expired state for the current hybrid app.

**Syntax**
<static> expireCredentials()

**Example**
```javascript
hwc.expireCredentials();
```

**getAllMessages([messageFilter], [completeList]) method**
Gets received messages based on a filter and the existance of a default hybrid app.

**Syntax**
<static> getAllMessages([messageFilter], [completeList]) {hwc.Message[]}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>

Develop Hybrid Apps Using Third-party Web Frameworks
messageFilter | hwc.MessageFilter on page 221 | (optional) | A filter that all returned messages will pass. If you do not want to filter based on a certain attribute, use null for that attribute when creating the filter. If you do not want to filter at all, pass in null for this parameter or do not pass in this parameter at all.

completeList | boolean | (optional) | If this parameter is set to true, then all messages will be returned. If this parameter is set to false or if it is not set, then if there is a default hybrid app only the messages belonging to the default hybrid app will be returned (and if there is no default hybrid app all messages will be returned).

**Returns**
An array of hwc.Message on page 215 objects - the received messages.

Type:
hwc.Message[]

**Example**

```javascript
// get all messages that have the subject "a subject".
var filter = new hwc.MessageFilter( null, "a subject", null, null, null, null );
var messages = hwc.getAllMessages(filter);

// Get all messages without filtering, but if there is a default hybrid app only return its messages.
var messages = hwc.getAllMessages();

// Get all messages (without filtering) for all hybrid apps, even if there is a default hybrid app.
var messages = hwc.getAllMessages( null, true );
```
**Source**
*hwc-api.js, line 2945 on page 401.*

**getAppByID( moduleID, version ) method**
Gets a `hwc.HybridApp` object with the given module id and version.

**Syntax**

```javascript
<static> getAppByID( moduleID, version ) {hwc.HybridApp}
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleID</td>
<td>number</td>
<td>The module ID of the hybrid app.</td>
</tr>
<tr>
<td>version</td>
<td>number</td>
<td>The version of the hybrid app.</td>
</tr>
</tbody>
</table>

**Returns**
The hybrid app object, or null if there is no hybrid app with the given ID and version.

**Type:**

`hwc.HybridApp` on page 197

**Example**

```javascript
// Messages do not have a direct link to the hybrid app they belong to. Instead they have
// the module ID and version of the hybrid app they belong to. If you have a message and
// need to access its hybrid app, first you must call hwc.getAppByID.
var messages = hwc.getAllMessages();
if( messages.length > 0 )
{
    var app = hwc.getAppByID( messages[0].getModuleId(),
                           messages[0].getModuleVersion() );
}
```

**Source**
*hwc-api.js, line 1810 on page 361.*

**getAppIconUrl( app, processed ) method**
This function gets the URL of the icon for a hybrid app depending on whether custom icons are defined.
**Syntax**

<static> `getAppIconUrl( app, processed )` {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app</code></td>
<td><code>hwc.HybridApp</code></td>
<td>The hybrid app for which the icon URL is desired.</td>
</tr>
<tr>
<td><code>processed</code></td>
<td>boolean</td>
<td>Whether to get the URL of the processed icon (true) or the URL of the unprocessed icon (false).</td>
</tr>
</tbody>
</table>

**Returns**

The URL of the icon.

Type:

string

**Example**

```javascript
var apps = hwc.getInstalledApps();
if( apps.length > 0 )
{
  var hybridApp = apps[0];
  // Create the image element.
  var hybridAppIcon = document.createElement("img");
  // Set the source of the image to the icon URL.
  hybridAppIcon.setAttribute( 'src', hwc.getAppIconUrl( hybridApp, false ) );
  // Add the image element to the page.
  document.body.appendChild( hybridAppIcon );
}
```

**Source**

`hwc-api.js, line 2406` on page 382.

**getApplicationConnectionID() method**

Gets the Hybrid Web Container application connection ID.

**Syntax**

<static> `getApplicationConnectionID()` {string}

**Returns**

Application connection ID
Type:
string

**Example**

```javascript
var appConnectionID = hwc.getApplicationConnectionID();
```

**Source**

*hwc-api.js, line 1912 on page 365.*

**getBuiltInIconUrl( iconIndex, processed ) method**

Gets the icon URL for the built-in icon.

This function is used by *hwc.getMsgIconUrl on page 189* and *hwc.getAppIconUrl on page 177*. It is possible to call this function directly, but generally it is easier simply to call *hwc.getAppIconUrl on page 177* or *hwc.getMsgIconUrl on page 189* instead. Those functions handle both cases where there is and isn't a custom icon for the hybrid app or message.

**Syntax**

```javascript
<static> getBuiltInIconUrl( iconIndex, processed ) → {string}
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iconIndex</td>
<td>number</td>
<td>The index of the built-in icon.</td>
</tr>
<tr>
<td>processed</td>
<td>boolean</td>
<td>Whether or not to get the URL of the processed icon (true) or the unprocessed icon (false).</td>
</tr>
</tbody>
</table>

**Returns**

The URL to the icon.

Type:
string

**Example**

```javascript
// Create the image element.
var builtInIcon = document.createElement( "img" );
// Set the source of the image to the icon URL.
builtInIcon.setAttribute( 'src', hwc.getBuiltInIconUrl(56, false) );
// Add the image element to the page.
document.body.appendChild( builtInIcon );
```
**Source**  
*hwc-api.js, line 2342 on page 380.*

**getCallbackFromNativeError** *(errString)* **method**
Extract the error call back method name from a URL string.

The parameter name of the error call back method should be "onErrorCallback".

**Syntax**

<static> `getCallbackFromNativeError(errString) {String}`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errString</td>
<td>String</td>
<td>The error string URL</td>
</tr>
</tbody>
</table>

**Returns**

the error callback method name

Type:

String

**Source**  
*hwc-utils.js, line 161 on page 496.*

**getClientVariables** *(moduleId, version)* **method**

Gets the client variables of the hybrid app with given module id and version.

**Syntax**

<static> `getClientVariables(moduleID, version) {hwc.ClientVariables}`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>number</td>
<td>The module ID of the hybrid app.</td>
</tr>
<tr>
<td>version</td>
<td>number</td>
<td>The version of the hybrid app.</td>
</tr>
</tbody>
</table>

**Returns**

A `hwc.ClientVariables` on page 165 object, or null if there are no ClientVariables for the hybrid app with the given module id and version.

Type:
**Example**

```javascript
var apps = hwc.getInstalledApps();
// Loop through the apps, showing the client variables for each one.
for( var i = 0; i < apps.length; i++ )
{
    var app = apps[i];
    // Get the client variables.
    var clientVariables = hwc.getClientVariables( app.getModuleID(),
        app.getVersion() );
    if( clientVariables.getCount() > 0 )
    {
        // Get all the names of the variables for this app.
        var keys = clientVariables.getAllVariableNames();
        // Loop through all the variable for this app.
        for( var index = 0; index < keys.length; index++ )
        {
            // Get a specific variable by name.
            var variable =
                clientVariables.getVariableValueByName( keys[index] );
            alert( "variable name: " + keys[index] + "\n        variable value: " + variable );
        }
    }
}
```

**Source**

*hwc-api.js, line 1961* on page 367.

**getCodeFromNativeError(errString) method**

Extract an error code from a URL string.

The parameter name of the error code should be "errCode".

**Syntax**

<static> `getCodeFromNativeError( errString )` {String}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errString</code></td>
<td>String</td>
<td>The error string URL</td>
</tr>
</tbody>
</table>

**Returns**

error code

Type:

String
Example

TODO: CONFIRM THE RETURN DATATYPE

Source

*hwc-utils.js, line 179 on page 496.*

**getCurrentApp() method**

Gets the hybrid app that is currently open.

**Syntax**

<static> getCurrentApp() {hwc.HybridApp}

**Returns**

The hybrid app that is currently open.

**Type:**

*hwc.HybridApp on page 197*

**Example**

```javascript
var openHybridApp = hwc.getCurrentApp();
```

Source

*hwc-api.js, line 1674 on page 356.*

**getCurrentLocale() method**

**Timezone**

**Syntax**

<static> getCurrentLocale() {string}

**Returns**

Returns a string containing the current locale, or null if it is not available.

**Type:**

string

**Example**

```javascript
var sLocale = hwc.getCurrentLocale();
```

Source

*Timezone.js, line 30 on page 574.*
**getCustomIconUrl( moduleId, moduleVersion, iconIndex, processed ) method**

Gets the URL to the custom icon.

This function is used by hwc.CustomIcon#getIconUrl.

**Syntax**

<static> getCustomIconUrl( moduleId, moduleVersion, iconIndex, processed ) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>number</td>
<td>The module Id of the hybrid app the custom icon belongs to.</td>
</tr>
<tr>
<td>moduleVersion</td>
<td>number</td>
<td>The version of the hybrid app the custom icon belongs to.</td>
</tr>
<tr>
<td>iconIndex</td>
<td>number</td>
<td>The index of the custom icon.</td>
</tr>
<tr>
<td>processed</td>
<td>boolean</td>
<td>Whether to get the processed icon (true), or the unprocessed icon (false).</td>
</tr>
</tbody>
</table>

**Returns**

The URL to the target icon.

Type: string

**Source**

hwc-api.js, line 2318 on page 379.

**getDstOffsetAtGivenTimeInMinutes( date ) method**

Timezone

**Syntax**

<static> getDstOffsetAtGivenTimeInMinutes( date ) {int}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date at which to determine daylight savings offset.</td>
</tr>
</tbody>
</table>
Returns
Returns the number of minutes offset for daylight savings for the current timezone and at the given Date, or 0 if the current timezone doesn’t practice daylight savings.

Type:
int

Example
var iDstOffsetAtTime = hwc.getDstOffsetAtGivenTimeInMinutes(date);

Source
Timezone.js, line 438 on page 589.

getExternalResource( url, options ) method
Makes an external cross domain request.

Syntax
<static> getExternalResource( url, options )

Parameters
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>String</td>
<td>The url to make request to</td>
</tr>
<tr>
<td>options</td>
<td>anonymous.options on page 72</td>
<td>a set of key/value pairs that configure the underlying request.</td>
</tr>
</tbody>
</table>

Example
var options = {
    method: "GET",
    data: "data",
    async: true,
    headers: {
        "Content-Type": "text/plain; charset=UTF-8"
    },
    complete: function(response) {
        // invoked when the request completes (asynchronous mode)
        if (response.status === 200)
            alert("Update successful");
        else
            alert("Update Failed");
    }
};
getExternalResource(url, options);
getInstalledApps([completeList]) method

Returns an array of hwc.HybridApp on page 197 objects.

Syntax

<static> getInstalledApps([completeList]) → {hwc.HybridApp[]}

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>completeList</td>
<td>boolean</td>
<td>(optional)</td>
<td>If this parameter is set to true, then all apps that are user invocable or require activation will be returned. If set to false or if it is not set, then if there is a default hybrid app only the default hybrid app will be returned (and if there is no default hybrid app it will return all hybrid apps that are user invocable or require activation).</td>
</tr>
</tbody>
</table>

Returns

An array of hybrid app objects.

Type:

hwc.HybridApp[]

Example

```javascript
var apps = hwc.getInstalledApps();

var apps = hwc.getInstalledApps( true );
```

Source

hwc-api.js, line 1717 on page 357.
**getLocalizedDate( date ) method**

Timezone

**Syntax**

<static> getLocalizedDate( date ) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date to be localized, initialized to some valid time.</td>
</tr>
</tbody>
</table>

**Returns**

Returns a localized date string, or undefined if platform is unsupported.

Type: string

**Example**

var sD = hwc.getLocalizedDate( date );

**Source**

Timezone.js, line 119 on page 577.

---

**getLocalizedDateTime( date ) method**

Timezone

**Syntax**

<static> getLocalizedDateTime( date ) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date to be localized, initialized to some valid time.</td>
</tr>
</tbody>
</table>

**Returns**

Returns a localized date/time string, or undefined if platform is unsupported.

Type: string
Example

```javascript
var sDT = hwc.getLocalizedDateTime( date );
```

Source

*Timezone.js, line 75 on page 576.*

**getLocalizedTime( date ) method**

*Timezone*

**Syntax**

```javascript
<static> getLocalizedTime( date ) {string}
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date to be localized, initialized to some valid time.</td>
</tr>
</tbody>
</table>

**Returns**

Returns a localized time string, or undefined if platform is unsupported.

Type:

string

Example

```javascript
var sT = hwc.getLocalizedTime( date );
```

Source

*Timezone.js, line 163 on page 579.*

**getLogEntries() method**

Call this function to get an array of *hwc.LogEntry* on page 209 objects.

There will be one *hwc.LogEntry* on page 209 object for each line in the HWC log.

**Syntax**

```javascript
<static> getLogEntries() {hwc.LogEntry[]}
```

**Returns**

An array of *hwc.LogEntry* objects.

Type:
hwc.LogEntry[]

Example
var log = hwc.getLogEntries();

Source
hwc-api.js, line 1047 on page 333.

getLoggingAlertDialog() method
This function gets the callback used by hwc.log when it is required to notify the user of a log item.

Syntax
<static> getLoggingAlertDialog() → {anonymous.alertDialogCallbackFunction}

Returns
The alert dialog callback function.

Type:
anonymous.alertDialogCallbackFunction on page 73

Source
hwc-comms.js, line 219 on page 440.

getLoggingCurrentLevel() method
This function gets the logging level.

Syntax
<static> getLoggingCurrentLevel() → {number}

Returns
A number representing the logging level. Will be an integer in the range [1..4]. The higher numbers represent more verbose logging levels from 1 for ERROR level logging up to 4 for DEBUG level logging.

Type:
number

Example
// Get the logging level
var loggingLevel = hwc.getLoggingCurrentLevel();
getSource
hcw-comms.js, line 252 on page 441.

**getMessageByID( msgId ) method**

Gets a hwc.Message object with the given message ID.

**Syntax**

<static> getMessageByID( msgId ) → {hwc.Message}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>number</td>
<td>The message ID of the message to get.</td>
</tr>
</tbody>
</table>

**Returns**

A message object, or null if no message with given ID.

**Type:**

hwc.Message on page 215

**Example**

```javascript
// A message listener is one place that would likely need to call
hwc.getMessageByID.
var messageListener = function( flag, messageID )
{
    // Since the callback function only gets the messageID, not the
    // message itself, if we want
    // more information about the message we must call
    hwc.getMessageByID.
    var message = hwc.getMessageByID( messageID );
    if( message.getSubject() == "a special subject" )
    {
        alert( "An event occurred for a special message!" );
    }
}
hwc.addMessageListener( null, messageListener );
```

**Source**

hcw-api.js, line 3034 on page 405.

**getMsgIconUrl( msg ) method**

This function gets the URL of the icon for a message object depending on its processed status and whether there are custom icons defined.
**getMsgIconUrl**

Syntax

<static> getMsgIconUrl( *msg*) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>msg</em></td>
<td><em>hwc.Message</em> on page 215</td>
<td>The message object</td>
</tr>
</tbody>
</table>

**Returns**

The url to access the icon.

**Example**

```javascript
var messages = hwc.getAllMessages();
if( messages.length > 0 )
{
    // Create the image element.
    var messageIcon = document.createElement("img");
    // Set the source of the image to the icon URL.
    messageIcon.setAttribute( 'src',
    hwc.getMsgIconUrl( messages[0] ) );
    // Add the image element to the page.
    document.body.appendChild( messageIcon );
}
```

**Source**

*hwc-api.js*, line 2368 on page 381.

**getNativeMessageFromNativeError**

Syntax

<static> getNativeMessageFromNativeError( *errString*) {String}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>errString</em></td>
<td>String</td>
<td>The error string URL</td>
</tr>
</tbody>
</table>

**Returns**

the native message

---

Develop Hybrid Apps Using Third-party Web Frameworks

SAP Mobile Platform
Type: String

Source
hwc-utils.js, line 195 on page 497.

**getOffsetFromUTC( date ) method**

**Timezone**

**Syntax**
<static> getOffsetFromUTC( date ) → {int}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date at which time to determine offset, initialized to some valid time.</td>
</tr>
</tbody>
</table>

**Returns**

Returns the GMT offset in minutes.

Type: int

**Example**

```
var totalOffset = hwc.getOffsetFromUTC(date);
```

Source

Timezone.js, line 269 on page 583.

**getOnErrorMessageFromNativeError( errString ) method**

**Extract the error message from a URL string.**

The parameter name of the error message should be "onErrorMsg".

**Syntax**

<static> getOnErrorMessageFromNativeError( errString ) {String}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errString</td>
<td>String</td>
<td>The error string URL</td>
</tr>
</tbody>
</table>
Returns
error message
Type:
String

Source
hwc-utils.js, line 138 on page 495.

genericPictureError( onGenericPictureError, onGenericPictureSuccess, options ) method
Camera

Syntax
<static> genericPicture( onGenericPictureError, onGenericPictureSuccess, options )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onGenericPictureError</td>
<td>anonymous.onGenericPictureError on page 81</td>
<td>Function to be invoked if the attempt to get a picture fails. err will be one of the PictureError codes.</td>
</tr>
<tr>
<td>onGenericPictureSuccess</td>
<td>anonymous.onGenericPictureSuccess on page 82</td>
<td>Function to be invoked if a picture is successfully retrieved. response will either be a Base64-encoded JPG string or a URI.</td>
</tr>
<tr>
<td>options</td>
<td>anonymous.PictureOptions on page 72</td>
<td>the options to control the sourceType and destinationType.</td>
</tr>
</tbody>
</table>

Example

```javascript
// Error handler. will be invoked asynchronously.
fail = function(errorCode){
    // handle error code and take appropriate action.
}

// Success handler. will be invoked asynchronously.
success = function(fileName, content){
    // handle the content. content may be a location or base64 encoded string that is
determined by the options passed to the destinationType argument.
}

getPicture(fail,
success,
```
getQueryVariable( variable ) method
This function looks in the query string on the URL for the value corresponding to the given name.

Syntax
<static> getQueryVariable( variable ) {string}

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>string</td>
<td>The name of the variable in the URL to retrieve the value for.</td>
</tr>
</tbody>
</table>

Returns
The value corresponding to the given name.

Type:
string

Example
// Get the pageToShow variable from the URL query string
var pageToShow = hwc.getQueryVariable( "pageToShow" );

Source
hwc-comms.js, line 301 on page 443.

getAddressInitiatedApps() method
Returns an array of hwc.HybridApp on page 197 objects that are server initiated.

Syntax
<static> getServerInitiatedApps() {hwc.HybridApp[]}

Returns
An array of server initiated hybrid apps.

Type:
Develop Hybrid Apps Using Third-party Web Frameworks

hwc.HybridApp[]

Example
var serverInitiatedApps = hwc.getServerInitiatedApps();

Source
hwc-api.js, line 1763 on page 359.

getSharedStorageKey() method
Storage

Syntax
<static> getSharedStorageKey() {string}

Returns
the shared storage key.

Type:
string

Source
SUPStorage.js, line 348 on page 572.

gTimezoneld() method
Timezone

Syntax
<static> getTimezoneId() {string}

Returns
Returns a string containing the current Timezone's standard name.

Type:
string

Example
var sTzId = hwc.getTimezoneId();

Source
Timezone.js, line 523 on page 592.
**getTransformData() method**

Returns the transform data for the hybridapp.

Only a server-initiated app will have this data.

**Syntax**

```javascript
<static> getTransformData()
```

**Returns**

the transform data.

**Example**

TODO: Add an example

**Source**

`hwc-utils.js`, line 59 on page 492.

---

**getURLParamFromNativeError( paramName, url ) method**

Extract a parameter value from a URL string with a given parameter name.

**Syntax**

```javascript
<static> getURLParamFromNativeError( paramName, url ) → {String}
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>paramName</code></td>
<td>String</td>
<td>The parameter name</td>
</tr>
<tr>
<td><code>url</code></td>
<td>String</td>
<td>The containing URL of the parameter</td>
</tr>
</tbody>
</table>

**Returns**

The parameter value

Type:

String

**Source**

`hwc-utils.js`, line 212 on page 497.

---

**getUsesDST() method**

Timezone
Syntax
<static> getUsesDST() {boolean}

Returns
Returns true iff the device's current timezone practices daylight savings, irrespective of whether daylight savings is currently in effect.

Type:
boolean

Example
var isDstAware = hwc.getUsesDST();

Source
Timezone.js, line 568 on page 593.

getXMLHTTPRequest() method
Reliably returns an XMLHttpRequest object regardless of what platform this code is being executed on.

Syntax
<static> getXMLHTTPRequest() {object}

Returns
An XMLHttpRequest object.

Type:
object

Example
var request = hwc.getXMLHTTPRequest();

Source
hwc-comms.js, line 470 on page 448.

guid() method
This function generates a GUID (globally unique identifier).

Syntax
<static> guid() {string}
Returns
The generated GUID.

Type:
string

Example
var globallyUniqueName = hwc.guid();

Source
hwc-comms.js, line 457 on page 448.

hideProgressDialog() method
This function hides the progress dialog displaying the spinner.

This function should be used to hide the progress dialog after a call to
hwc.showProgressDialog on page 244. If this function is called while there is no progress
dialog, then nothing will happen.

Syntax
<static> hideProgressDialog()

Example
var showProgress = function()
{
    hwc.showProgressDialog( "a message" );
    setTimeout( hideProgress, 10000 );
}

var hideProgress = function()
{
    hwc.hideProgressDialog();
}

Source
hwc-comms.js, line 1502 on page 486.

HybridApp( moduleId, version, displayName, iconIndex, defaultCustomIcon, customIconList ) method
This object represents a hybrid app.

Syntax
<static> HybridApp( moduleId, version, displayName, iconIndex, defaultCustomIcon, customIconList )
Develop Hybrid Apps Using Third-party Web Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>number</td>
<td>The module id of this hybrid app.</td>
</tr>
<tr>
<td>version</td>
<td>number</td>
<td>The version of this hybrid app.</td>
</tr>
<tr>
<td>displayName</td>
<td>string</td>
<td>The display name of this hybrid app.</td>
</tr>
<tr>
<td>iconIndex</td>
<td>number</td>
<td>The index specifying the icon representing this Hybrid App.</td>
</tr>
<tr>
<td>defaultCustomIcon</td>
<td>hwc.CustomIcon on page 172</td>
<td>The default custom icon for this hybrid app.</td>
</tr>
<tr>
<td>customIconList</td>
<td>hwc.CustomIcon[]</td>
<td>An array of custom icon objects.</td>
</tr>
</tbody>
</table>

**Source**
*hwc-api.js, line 1363 on page 345.*

**this.getClientVariables() method**
Return a *hwc.ClientVariables* on page 165 object for the given module id and version.

**Syntax**
<static> this.getClientVariables() {hwc.ClientVariables}

**Returns**
The *hwc.ClientVariables* on page 165 object for this hybrid app.

Type:
*hwc.ClientVariables* on page 165

**Source**
hwc-api.js, line 1444 on page 347.

**this.getCustomIconList() method**
Gets the list of custom icons associated with this hybrid app.

**Syntax**
<static> this.getCustomIconList() {hwc.CustomIcon[]}
Returns
The array of custom icon objects. Null if this hybrid app has no custom icons.

Type:
hwc.CustomIcon[]

Source
hwc-api.js, line 1433 on page 347.

this.getDefaultCustomIcon() method
Gets the default custom icon object of this hybrid app.

Syntax
<static> this.getDefaultCustomIcon() {hwc.CustomIcon}

Returns
The default custom icon of this hybrid app. Null if this hybrid app does not have a custom icon.

Type:
hwc.CustomIcon on page 172

Source
hwc-api.js, line 1422 on page 347.

this.getDisplayName() method
Gets the display name for this hybrid app.

Syntax
<static> this.getDisplayName() {string}

Returns
The display name.

Type:
string

Source
hwc-api.js, line 1400 on page 346.
**this.setIconIndex() method**
Gets the icon index used in the list of built-in icons.

**Syntax**
<static> this.setIconIndex() {number}

**Returns**
The icon index

Type:
number

**Source**
hwc-api.js, line 1411 on page 346.

**this.getModuleID() method**
Gets the module ID for this hybrid app.

**Syntax**
<static> this.getModuleID() {number}

**Returns**
The module ID.

Type:
number

**Source**
hwc-api.js, line 1378 on page 345.

**this.getVersion() method**
Gets the version number for this hybrid app.

**Syntax**
<static> this.getVersion() {number}

**Returns**
The version.

Type:
number
Source
hwc-api.js, line 1389 on page 346.

**isAndroid() method**
Platform

**Syntax**
<static> isAndroid() {boolean}

**Returns**
True if the hybrid app application is being run on an Android platform.

Type:
boolean

Source
PlatformIdentification.js, line 149 on page 506.

**isAndroid3() method**
Platform

**Syntax**
<static> isAndroid3() {boolean}

**Returns**
True if the hybrid app application is being run on an Android 3.0 OS

Type:
boolean

Source
PlatformIdentification.js, line 141 on page 506.

**isBlackBerry() method**
Platform

**Syntax**
<static> isBlackBerry() {boolean}

**Returns**
True if the hybrid app application is being run on a BlackBerry platform.
Develop Hybrid Apps Using Third-party Web Frameworks

Type:
boolean

Source
PlatformIdentification.js, line 83 on page 503.

**isBlackBerry5() method**
Platform

Syntax
<static> isBlackBerry5() {boolean}

Returns
True if the hybrid app application is being run on a BlackBerry 5.0 OS

Type:
boolean

Source
PlatformIdentification.js, line 91 on page 504.

**isBlackBerry5WithTouchScreen() method**
Platform

Syntax
<static> isBlackBerry5WithTouchScreen() {boolean}

Returns
True if the hybrid app application is being run on a BlackBerry 5.0 OS with a touch screen

Type:
boolean

Source
PlatformIdentification.js, line 107 on page 504.

**isBlackBerry6NonTouchScreen() method**
Platform

Syntax
<static> isBlackBerry6NonTouchScreen() {boolean}
**Returns**
True if the hybrid app application is being run on a BlackBerry 6.0 OS without a touch screen

**Type:**
boolean

**Source**
PlatformIdentification.js, line 115 on page 505.

**isBlackBerry7() method**
Platform

**Syntax**
<static> isBlackBerry7() {boolean}

**Returns**
True if the hybrid app application is being run on a BlackBerry 7.x OS

**Type:**
boolean

**Source**
PlatformIdentification.js, line 99 on page 504.

**isClosed() method**
This function checks if the hybrid app has been closed.

**Syntax**
<static> isClosed() {boolean}

**Returns**
true if hybrid app is closed, otherwise false.

**Type:**
boolean

**Example**

```javascript
hwc.isClosed();
```

**Source**
hwc-comms.js, line 1602 on page 489.
**isDstActiveAtGivenTime( date ) method**

**Timezone**

**Syntax**

<static> isDstActiveAtGivenTime( date ) → {boolean}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date</td>
<td>Date at which to determine whether daylight savings is in effect.</td>
</tr>
</tbody>
</table>

**Returns**

Returns true iff daylight savings rules are in effect at the given time in the current timezone.

Type:

boolean

**Example**

```javascript
var isAwareAtTime = hwc.isDstActiveAtGivenTime(date);
```

**Source**

*Timezone.js, line 356 on page 586.*

**isIOS() method**

**Platform**

**Syntax**

<static> isIOS() {boolean}

**Returns**

True if the hybrid app application is being run on an iOS (e.g. iPhone, iPad) platform.

Type:

boolean

**Source**

*PlatformIdentification.js, line 34 on page 502.*
**isIOS4() method**
Returns true if the hybrid app application is being run on iOS4

**Syntax**
<static> isIOS4() {Boolean}

**Returns**
True if the hybrid app application is being run on iOS4

Type:
Boolean

**Source**
PlatformIdentification.js, line 74 on page 503.

**isIOS5() method**
Platform

**Syntax**
<static> isIOS5() {boolean}

**Returns**
True if the hybrid app application is being run on iOS5

Type:
boolean

**Source**
PlatformIdentification.js, line 50 on page 502.

**isIOS6() method**
Returns true if the hybrid app application is being run on iOS6

**Syntax**
<static> isIOS6() {boolean}

**Returns**
True if the hybrid app application is being run on iOS6

Type:
boolean
Source
PlatformIdentification.js, line 57 on page 502.

**isIOS7() method**
Returns true if the hybrid app application is being run on iOS7

**Syntax**
<static> isIOS7() {boolean}

**Returns**
True if the hybrid app application is being run on iOS7

Type:
boolean

Source
PlatformIdentification.js, line 65 on page 503.

**isIPad() method**
Platform

**Syntax**
<static> isIPad() {boolean}

**Returns**
True if the hybrid app application is being run on an iPad.

Type:
boolean

Source
PlatformIdentification.js, line 42 on page 502.

**isSharedStorageEnabled() method**
Storage

**Syntax**
<static> isSharedStorageEnabled() {boolean}

**Returns**
true if the shared storage is enabled; false otherwise.
Type:
boolean

Source
SUPStorage.js, line 362 on page 572.

isWindows() method
Platform

Syntax
<static> isWindows() {boolean}

Returns
True if the hybrid app application is being run on a Windows platform.

Type:
boolean

Source
PlatformIdentification.js, line 132 on page 505.

isWindowsMobile() method
Platform

Syntax
<static> isWindowsMobile() {boolean}

Returns
True if the hybrid app application is being run on a Windows Mobile platform.

Type:
boolean

Source
PlatformIdentification.js, line 124 on page 505.

loadSettings() method
Loads the current connection settings from the native application storage.

Syntax
<static> loadSettings() {hwc.ConnectionSettings}
Returns
The connection settings or null if there are no cached settings.

Type:
`hwc.ConnectionSettings` on page 168

Example
```javascript
// Load the connection settings.
var connectionSettings = hwc.loadSettings();
```

Source
`hwc-api.js, line 101` on page 297.

**log** (*sMsg*, *eLevel*, *notifyUser*) method
Allows the user to log a message to the device trace log which can be remotely retrieved from the server.

Whether the message actually gets logged will depend on how the log level that the administrator has selected for this device user compares with the log level of this message. The logging level and alert dialog callback can be set with `hwc.setLoggingCurrentLevel` on page 239 and `setLoggingAlertDialog`.

Syntax
```javascript
<static> log(sMsg, eLevel, notifyUser)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>sMsg</em></td>
<td>string</td>
<td>The message to be logged.</td>
</tr>
<tr>
<td><em>eLevel</em></td>
<td>string</td>
<td>The error level for this message. This parameter must be one of: &quot;ERROR&quot;, &quot;WARN&quot;, &quot;INFO&quot;, or &quot;DEBUG&quot;.</td>
</tr>
<tr>
<td><em>notifyUser</em></td>
<td>boolean</td>
<td>Whether the logging alert callback will be invoked. This parameter is independent of the logging level (the logging alert callback will always be invoked if this is true, and never if this is false).</td>
</tr>
</tbody>
</table>
Example

```javascript
var logAlert = function( message )
{
    alert( "New log message: " + message );
}

hwc.setLoggingAlertDialog( logAlert );
hwc.setLoggingCurrentLevel( 3 );
// The following will be logged, and the logging alert dialog will be invoked.

hwc.log( "info message notify", "INFO", true );
// The following will be logged, but the logging alert dialog will not be invoked.

hwc.log( "info message", "INFO", false );
// The following will not be logged, but the logging alert dialog will be invoked.

hwc.log( "debug message notify", "DEBUG", true );
// The following will not be logged, and the logging alert dialog will not be invoked.

hwc.log( "debug message", "DEBUG", false );
```

Source

*hwc-comms.js*, line 902 on page 464.

**LogEntry( date, event, msg ) method**

This object represents a log entry.

**Syntax**

<static> `LogEntry( date, event, msg )`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>date</code></td>
<td>number</td>
<td>The date the log entry was recorded, in milliseconds since January 1, 1970, 00:00:00 GMT</td>
</tr>
<tr>
<td><strong>event</strong></td>
<td><strong>number</strong></td>
<td><strong>msg</strong></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The event ID of the log entry (will be one of <code>hwc.CONNECTION_ERROR</code> on page 128, <code>hwc.CONNECTION_OTHER</code> on page 129, <code>hwc.CONNECTION_CONNECTED</code> on page 128, <code>hwc.CONNECTION_DISCONNECTED</code> on page 128, <code>hwc.CONNECTION_RETRIEVED_ITEMS</code> on page 129)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The message of the log entry.</td>
</tr>
</tbody>
</table>

**Source**

`hwc-api.js, line 1084` on page 334.

**this.getDate() method**

Gets the date of the log entry.

**Syntax**

```
<static> this.getDate() {number}
```

**Returns**

The date the log entry was created in the HWC, in milliseconds.

Type:

*number*

**Source**

`hwc-api.js, line 1096` on page 335.

**this.getEventID() method**

Gets the event ID of the log entry to see what this log entry is about.

**Syntax**

```
<static> this.getEventID() {number}
```

**Returns**

A constant indication what this log entry is about (will be one of `hwc.CONNECTION_ERROR` on page 128, `hwc.CONNECTION_OTHER` on page 129, `hwc.CONNECTION_CONNECTED` on page 128, `hwc.CONNECTION_DISCONNECTED` on page 128, `hwc.CONNECTION_RETRIEVED_ITEMS` on page 129).
Type:
number

Source
hwc-api.js, line 1108 on page 335.

`this.getMessage()` method
Gets the message text of the log entry.

Syntax
<static> this.getMessage() {string}

Returns
The message text of the log entry.

Type:
string

Source
hwc-api.js, line 1119 on page 335.

`markAsActivated()` method
This function sets the activation required state for the current hybrid app to false.

After calling this function, the current hybrid app will not need to be activated.

Syntax
<static> markAsActivated()

Example
hwc.markAsActivated();

Source
hwc-comms.js, line 790 on page 460.

`markAsProcessed()` method
Allows the user to set the processed state to true for the current message.

Syntax
<static> markAsProcessed()
Example

```javascript
hwc.markAsProcessed()
```

Source
`hwc-comms.js`, line 806 on page 460.

**MenuItemCollection() method**
This class represents a collection of menu items.

Syntax
`<static> MenuItemCollection()`

Example

```javascript
// This is the function we'll use as a callback for the first menu item.
var callback = function()
{
    alert( "You clicked the first menu item!" );
}

// This is the function we'll use as a callback for the second menu item.
var callback2 = function()
{
    alert( "You clicked the second menu item!" );
}

// This function creates and adds a menu item collection.
var addMenuItems = function()
{
    var menuItemCollection = new hwc.MenuItemCollection();
    menuItemCollection.addMenuItem("menu item 1", "callback()");
    menuItemCollection.addMenuItem("menu item 2", "callback2()");
    hwc.addMenuItemCollection( menuItemCollection );
}
```

Source
`hwc-comms.js`, line 554 on page 451.

**addMenuItem( title, callback, [isDefault] ) method**
This function adds a menu item to the collection.

Syntax
```javascript
addMenuItem( title, callback, [isDefault] )
```
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td></td>
<td>The display text for the menu item.</td>
</tr>
<tr>
<td>callback</td>
<td><code>anonymous.genericCallbackFunction</code> on page 78</td>
<td></td>
<td>The function to call when the menu item is clicked.</td>
</tr>
<tr>
<td>isDefault</td>
<td>boolean</td>
<td>(optional)</td>
<td>Determines if the menu item is selected by default on BlackBerry. If more than one menu item is added to the same collection with true for this parameter, the last menu item added with true for this parameter will be selected by default on BlackBerry.</td>
</tr>
</tbody>
</table>

#### Example

```javascript
var callbackFunctionName = function()
{
   alert( "Menu item clicked!" );
}
var menuItemCollection = new hwc.MenuItemCollection();
menuItemCollection.addMenuItem("menu item name", "callbackFunctionName()", true);
```

**Source**

`hwc-comms.js`, line 579 on page 452.

**setOKAction( callback ) method**

This function sets the OK action to use on WM.

#### Syntax

`setOKAction( callback )`

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

---

Developer Guide: Hybrid Apps
**callback**

| anonymous.genericCallback-Function on page 78 | The function to call when the OK button is pressed. |

**Example**

```javascript
var callbackFunctionName = function()
{
    alert( "Menu item clicked!" );
}
var okActionFunction = function()
{
    alert( "A OKAY!" );
}
var menuItemCollection = new hwc.MenuItemCollection();
menuItemCollection.setOKAction( "okActionFunction()" );
menuItemCollection.addMenuItem("menu item name", "callbackFunctionName()");
```

**Source**

*hwc-comms.js*, line 626 on page 454.

**setSubMenuName( name ) method**

This function sets the sub menu name to use on Windows Mobile.

**Syntax**

```javascript
setSubMenuName( name )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The sub menu name to use.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
var callbackFunctionName = function()
{
    alert( "Menu item clicked!" );
}
var menuItemCollection = new hwc.MenuItemCollection();
menuItemCollection.setSubMenuName( "Custom Menu" );
menuItemCollection.addMenuItem("menu item name", "callbackFunctionName()");
```

**Source**

*hwc-comms.js*, line 603 on page 453.
**stringify() method**

This function converts the menu item collection to a JSON string.

This function is used as a helper for `hwc.addMenuItemCollection` on page 154.

**Syntax**

```javascript
stringify() {string}
```

**Returns**

The JSON string representing this menu item collection.

Type:

string

**Example**

```javascript
var callbackFunctionName = function () {
    alert("Menu item clicked!");
}
var menuItemCollection = new hwc.MenuItemCollection();
var jsonMenuItemCollection = menuItemCollection.stringify();
```

**Source**

`hwc-comms.js`, line 645 on page 455.

---

**Message( msgId, date, icon, sender, isRead, processed, priority, subject, module, version ) method**

Represents a message received by the HWC.

**Syntax**

```javascript
<static> Message( msgId, date, icon, sender, isRead, processed, priority, subject, module, version )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>msgId</code></td>
<td>number</td>
<td>The message ID of this message.</td>
</tr>
<tr>
<td><code>date</code></td>
<td>Date</td>
<td>The date this message was received.</td>
</tr>
<tr>
<td><code>icon</code></td>
<td>number</td>
<td>The icon index for this message.</td>
</tr>
<tr>
<td><code>sender</code></td>
<td>string</td>
<td>The sender of this message.</td>
</tr>
<tr>
<td>isRead</td>
<td>boolean</td>
<td>Whether this message has been read or not.</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>processed</td>
<td>boolean</td>
<td>Whether this message has been processed or not.</td>
</tr>
<tr>
<td>priority</td>
<td>number</td>
<td>The priority of this message (must be either hwc.MSG_PRIORITY_HIGH on page 135 or hwc.MSG_PRIORITY_NORMAL on page 136).</td>
</tr>
<tr>
<td>subject</td>
<td>string</td>
<td>The subject of this message.</td>
</tr>
<tr>
<td>module</td>
<td>number</td>
<td>The module ID of the hybrid app associated with this message.</td>
</tr>
<tr>
<td>version</td>
<td>number</td>
<td>The version of the hybrid app associated with this message.</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 2441 on page 383.

**this.setIconIndex() method**
Gets the icon index of this message.

**Syntax**
<static> this.setIconIndex() {number}

**Returns**
The icon index of this message.

**Type:**
number

Source
hwc-api.js, line 2482 on page 385.

**this.getMessageId() method**
Gets the message ID of this message.

**Syntax**
<static> this.getMessageId() {number}
**Returns**
The message ID of this message.

Type:

number

**Source**
hwc-api.js, line 2460 on page 384.

**this.getModuleId() method**
Gets the module ID of the hybrid app this message belongs to.

**Syntax**
<static> this.getModuleId() {number}

**Returns**
The module ID of the hybrid app this message belongs to.

Type:

number

**Source**
hwc-api.js, line 2526 on page 386.

**this.getModuleVersion() method**
Gets the version of the hybrid app this message belongs to.

**Syntax**
<static> this.getModuleVersion() {number}

**Returns**
The version of the hybrid app this message belongs to.

Type:

number

**Source**
hwc-api.js, line 2537 on page 387.

**this.getPriority() method**
Gets the priority of the message.
### Syntax

<static> this.getPriority() {number}

### Returns

A constant indicating the priority of the message. Will be either `hwc.MSG_PRIORITY_NORMAL` on page 136 or `hwc.MSG_PRIORITY_HIGH` on page 135.

Type:

number

### Source

`hwc-api.js`, line 2563 on page 388.

#### this.getReceivedDate() method

Gets the date this message was received.

### Syntax

<static> this.getReceivedDate() {Date}

### Returns

The date this message was received.

Type:

Date

### Source

`hwc-api.js`, line 2471 on page 384.

#### this.getSender() method

Gets the sender of this message.

### Syntax

<static> this.getSender() {string}

### Returns

The sender of this message.

Type:

string
**this.getSubject() method**
Gets the subject of this message.

**Syntax**
<static> this.getSubject() {string}

**Returns**
The subject of this message.

Type:
string

**Source**
hwc-api.js, line 2493 on page 385.

**this.isProcessed() method**
Gets whether this message has been processed or not.

A message is generally marked as processed once the user submits changes from the hybrid app that was launched from the message.

**Syntax**
<static> this.isProcessed() {boolean}

**Returns**
True if this message has been processed, false otherwise.

Type:
boolean

**Source**
hwc-api.js, line 2515 on page 386.

**this.isRead() method**
Gets whether this message has been read or not.

**Syntax**
<static> this.isRead() {boolean}
**Returns**
Whether this message has been read (true) or not (false).

Type:
boolean

**Source**
hwc-api.js, line 2504 on page 386.

*this.updateProcessed( status ) method*
Updates the processed status of the message.

**Syntax**
<static> this.updateProcessed( status )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>boolean</td>
<td>The new processed status.</td>
</tr>
</tbody>
</table>

**Source**
hwc-api.js, line 2592 on page 388.

*this.updateRead( status ) method*
Updates the read status of the message.

**Syntax**
<static> this.updateRead( status )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>boolean</td>
<td>The new read status.</td>
</tr>
</tbody>
</table>

**Source**
hwc-api.js, line 2580 on page 388.
**MessageFilter( [sender], [subject], [moduleId], [version], [isread], [processed] ) method**

Represents a filter used to filter messages.

Pass in null for any parameter you do not wish to filter (or do not pass in such parameters at all).

**Syntax**

<static> MessageFilter( [sender], [subject], [moduleId], [version], [isread], [processed] )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender</td>
<td>string</td>
<td>(optional)</td>
<td>The sender of the message.</td>
</tr>
<tr>
<td>subject</td>
<td>string</td>
<td>(optional)</td>
<td>The subject of the message.</td>
</tr>
<tr>
<td>moduleId</td>
<td>number</td>
<td>(optional)</td>
<td>The associated application module ID.</td>
</tr>
<tr>
<td>version</td>
<td>number</td>
<td>(optional)</td>
<td>The associated application module version.</td>
</tr>
<tr>
<td>isread</td>
<td>boolean</td>
<td>(optional)</td>
<td>The read status.</td>
</tr>
<tr>
<td>processed</td>
<td>boolean</td>
<td>(optional)</td>
<td>The processed status.</td>
</tr>
</tbody>
</table>

Source

hwc-api.js, line 2614 on page 389.

**openApp( moduleId, version ) method**

Launch the hybrid app with the given module ID and version.

The hybrid app will be opened on top of the hybrid app that is open when hwc.openApp is called. When the hybrid app that was opened with hwc.openApp exits, it will exit to the hybrid app that was open when hwc.openApp was called. It is possible to nest open hybrid apps, but it is best not to have too many nested hybrid apps (eg: recursively opening hybrid apps) because each open hybrid app takes up device memory.

**Syntax**

<static> openApp( moduleId, version ) {number}
Develop Hybrid Apps Using Third-party Web Frameworks

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>number</td>
<td>Module id of the hybrid app.</td>
</tr>
<tr>
<td>version</td>
<td>number</td>
<td>Version of the hybrid app.</td>
</tr>
</tbody>
</table>

### Returns

A constant indicating the result of opening the hybrid app (will be one of \( hwc.\text{OPEN\_APP\_SUCCESS} \) on page 138, \( hwc.\text{OPEN\_APP\_NOT\_EXIST} \) on page 138, \( hwc.\text{OPEN\_APP\_OTHER} \) on page 138).

Type:

- number

### Example

```javascript
var apps = hwc.getInstalledApps();
if( apps.length > 0 )
{
    // Check to make sure the first app is not this app (the app that
    // is currently running),
    // since we don't want to recursively open this app until memory
    // runs out.
    if( hwc.getCurrentHybridApp.getDisplayName() !=
        apps[0].getDisplayName() )
    {
        hwc.openApp( apps[0].getModuleID(), apps[0].getVersion() );
    }
}
```

### Source

*hwc-api.js, line 1883* on page 364.

### openMessage( msgId ) method

Launch the server initiated hybrid app associated with a message.

The hybrid app will be opened on top of the hybrid app that is open when \( hwc.\text{openMessage} \) is called. When the hybrid app that was opened with \( hwc.\text{openMessage} \) exits, it will exit to the hybrid app that was open when \( hwc.\text{openMessage} \) was called. It is possible to nest open hybrid apps, but it is best not to have too many nested hybrid apps (eg: recursively opening hybrid apps) because each open hybrid app takes up device memory.

### Syntax

```javascript
<static> openMessage( msgId ) {number}
```
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>number</td>
<td>The id of message to open.</td>
</tr>
</tbody>
</table>

Returns

A number indicating the success or failure of opening the message (will be one of `hwc.OPEN_MSG_SUCCESS` on page 140, `hwc.OPEN_MSG_NOT_EXIST` on page 139, `hwc.OPEN_MSG_APP_NOT_EXIST` on page 139, `hwc.OPEN_MSG_OTHER` on page 139).

Type:

number

Example

```javascript
// get all messages, then open the first one
var messages = hwc.getAllMessages();
if( messages.length > 0 )
{
   hwc.openMessage( messages[0].getMessageId() );
}
```

Source

`hwc-api.js, line 3185` on page 410.

`removeAllMenuItems()` method

This function removes all menu items that were added by the hybrid app.

Note: This API does not support on iOS platform.

Syntax

```javascript
<static> removeAllMenuItems()
```

Example

```javascript
hwc.removeAllMenuItems();
```

Source

`hwc-comms.js, line 754` on page 458.

`removeAppInstallationListener( AppInstallationListener )` method

Remove the application installation listener.

This function should be called with identical parameters that were used to add the application installation listener with `hwc.addAppInstallationListener` on page 149.
Syntax
<static> removeAppInstallationListener( AppInstallationListener )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppInstallationListener</td>
<td>anonymous.AppInstallationListener on page 73</td>
<td>The callback for application installation changes.</td>
</tr>
</tbody>
</table>

Example

```javascript
// appInstallListener is the callback function that will be passed to hwc.addAppInstallationListener.
var appInstallListener = function( event, moduleId, version, moduleName )
{
    if( event == hwc.INSTALLATION_BEGIN )
    {
        alert(moduleName + " has just started the installation process.");
    }
    else if( event == hwc.INSTALLATION_END )
    {
        alert(moduleName + " has just finished the installation process.");
    }
}  
hwc.addAppInstallationListener( appInstallListener );
// when we want to remove this listener, we call the following line:
hwc.removeAppInstallationListener( appInstallListener );
```

Source

hwc-api.js, line 984 on page 330.

removeAppListener( ApplicationListener, [containingObject] ) method

Remove the application listener.

This function should be called with identical parameters that were used to add the application listener with hwc.addAppListener on page 150.

Syntax

<static> removeAppListener( ApplicationListener, [containingObject] )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example**

```javascript
// This is the callback function that will be passed to hwc.addAppListener.
var appListener = function( event, moduleId, version ) {
  if( event == hwc.APP_ADDED )
  {
    alert("A hybrid app has been added.");
  }
}
hwc.addAppListener( appListener );
// At some other point, if we want to remove the listener we use the following line of code:
hwc.removeAppListener( appListener );

// appListenerManager is an object that will contain the callback function as well as variables
// the callback function references.
var appListenerManager = {};
// doSomething is a function that is called from inside the callback function.
appListenerManager.doSomething = function( event ) {
  if( event == hwc.APP_REMOVED )
  {
    alert("A hybrid app has been removed.");
  }
}
// This is the callback function that will be passed to hwc.addAppListener. It calls doSomething,
// the definition of which is in the containing function.
appListenerManager.listener = function( event, moduleId, version ) {
  this.doSomething( event );
}
// Since the listener callback function references a variable from its containing object,
// the containing object must be passed to hwc.addAppListener.
hwc.addAppListener( appListenerManager.listener, appListenerManager );
// At some other point, if we want to remove the listener we use the following line of code:
hwc.removeAppListener( appListenerManager.listener, appListenerManager );
```
Source
hwc-api.js, line 1601 on page 353.

**removeConnectionListener( ConnectionStateListener, [containingObject] ) method**
Remove the connection state listener.

This function should be called with identical parameters that were used when adding the connection state listener with `hwc.addConnectionListener` on page 151.

**Syntax**
`<static> removeConnectionListener( ConnectionStateListener, [containingObject] )`

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionStateListener</td>
<td>anonymous.ConnectionStateListener on page 77</td>
<td></td>
<td>Callback function with connection state changes</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>Optional Object containing definition of ConnectionStateListener</td>
</tr>
</tbody>
</table>

**Example**

```javascript
// doSomething is a global function that gets called from the connection listener.
var doSomething = function()
{
    alert("sample function that gets executed when the hwc becomes connected");
}
// connectionListener is the callback function that is given to addConnectionListener.
// When there is a connection event, connectionListener will be invoked with the details.
var connectionListener = function( event, errorCode, errorMessage )
{
    if( event == hwc.CONNECTED )
    {
        doSomething();
    }
}
// At some other point if we want to remove the listener, we use the
```

Develop Hybrid Apps Using Third-party Web Frameworks

226 SAP Mobile Platform
following line:

```javascript
hwc.removeConnectionListener( connectionListener );
```

// connectionStateManager is an object that will contain the
// connection listener callback as well as
// a variable used by the callback.
var connectionStateManager = {};
// The connectionStateManager keeps track of whether the HWC is
// connected or not.
connectionStateManager.connected = false;
// A function called by the listener.
connectionStateManager.doSomething = function()
{
    if( this.connected )
    {
        alert("this alert gets displayed if the hwc is connected");
    }
}
// This is the callback function that will be passed to
// addConnectionListener. This callback references variables
// from the containing object (this.connected and this.doSomething),
// so when we call addConnectionListener we have
// to give the containing object as the second parameter.
connectionStateManager.listener = function( event, errorCode, errorMessage )
{
    if( event == hwc.CONNECTED )
    {
        this.connected = true;
    }
    else
    {
        this.connected = false;
    }
    this.doSomething();
}
// Pass both the listener and the containing object. This enables
// the listener to refer to variables in the containing object when it
// is invoked.
hwc.addConnectionListener( connectionStateManager.listener, connectionStateManager );
// At some other point if we want to remove the listener, we use the
// following line:
hwc.removeConnectionListener( connectionStateManager.listener, connectionStateManager );

**Source**

*hwc-api.js, line 445* on page 311.

**removeLogListener(LogListener, [containingObject]) method**

Remove the log listener.

This function should be called with identical parameters that were used when adding the log
listener with *hwc.addLogListener* on page 153.
Syntax
\(<\text{static}>\) \text{removeLogListener}( \text{LogListener, [containingObject]} )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{LogListener}</td>
<td>\text{anonymous.LogListener on page 79}</td>
<td></td>
<td>The callback function for log events.</td>
</tr>
<tr>
<td>\text{containingObject}</td>
<td>\text{Object}</td>
<td>(optional)</td>
<td>Object containing definition of ConnectionStateListener</td>
</tr>
</tbody>
</table>

Example

// A global function called by the log listener.
var doSomething = function()
{
    alert("this gets displays when there is a log event.");
}
// The log listener callback function that will be passed to hwc.addLogListener.
// This function will be invoked whenever there is a log event.
var logListener = function( event, errorCode, errorMessage )
{
    doSomething();
}
// Add the log listener.
hwc.addLogListener( logListener );
// at some other point if we want to remove the listener, we use the following line
hwc.removeLogListener( logListener );

// logListenerManager is an object that will contain the listener callback as well
// as a function that will be invoked from the listener callback function.
var logListenerManager = {};
// This is a function that is called from the listener callback.
logListenerManager.doSomething = function()
{
    alert("this gets displays when there is a log event.");
}
// This is the listener callback that will be passed to hwc.addLogListener.
// Since a variable is referenced from the containing object, the containing object
// will need to be passed to hwc.addLogListener.
logListenerManager.listener = function( event, errorCode, errorMessage )
{
    this.doSomething();
}
Source

hwc-api.js, line 818 on page 324.

**removeMessage( msgId ) method**

Removes (deletes) a message.

**Syntax**

<static> removeMessage( msgId )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>number</td>
<td>The id of the message to be removed.</td>
</tr>
</tbody>
</table>

**Example**

// remove all messages
var messages = hwc.getAllMessages();
for( var index = 0; index < messages.length; index++ )
{
    hwc.removeMessage( messages[index].getMessageId() );
}

Source

hwc-api.js, line 3132 on page 408.

**removeMessageListener( MessageListener, [containingObject] ) method**

Removes the message listener.

The two parameters passed in to this function should match exactly the corresponding parameters passed into hwc.addMessageListener on page 155 when the message listener was added.

**Syntax**

<static> removeMessageListener( MessageListener, [containingObject] )
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageListener</td>
<td>anonymous.MessageListener on page 80</td>
<td></td>
<td>The callback for message changes.</td>
</tr>
<tr>
<td>containingObject</td>
<td>Object</td>
<td>(optional)</td>
<td>If the containing object was given to <code>hwc.addMessageListener</code> on page 155 when the message listener was added, then it also must be passed into this function.</td>
</tr>
</tbody>
</table>

### Example

```javascript
// soSomething is a global function called by the listener callback.
var doSomething = function() {
    alert("New message!");
}
// messageListener is the callback function passed to `hwc.addMessageListener`. 
var messageListener = function( flag, messageId ) {
    if( flag == hwc.MSG_ADDED ) {
        doSomething();
    }
}
// We do not want to filter the message events the listener will get invoked for, so pass null for the first parameter.
hwc.addMessageListener( null, messageListener );
// If we want to remove the listener at some other point, use the following line of code:
hwc.removeMessageListener( messageListener );

// someObject is an object that will contain the listener callback as well as a variable referenced by the callback. 
var someObject = {};
// doSomething is a function referenced by the callback function. 
someObject.doSomething = function() {
    alert("New message!");
}
// messageListener is the callback that will be passed to `hwc.addMessageListener`. 
someObject.messageListener = function( flag, messageId ) {
    if( flag == hwc.MSG_ADDED )
```
Create a filter so that not all message events will invoke our callback function. Only events about messages with a subject of "SI<4>" will trigger our callback function.

```javascript
var filter = new hwc.MessageFilter( null, "SI<4>", null, null, null, null);
```

The callback function references a variable in its containing object, so we need to pass in the containing object in addition to the filter and the callback function.

```javascript
hwc.addMessageListener( filter, someObject.messageListener, someObject );
```

If we want to remove the listener at some other point, use the following line of code:

```javascript
hwc.removeMessageListener( messageListener, someObject );
```

### Source

`hwc-api.js`, line 2850 on page 398.

#### removePushNotificationListener( PushNotificationListener, [containingObject] ) method

Remove the push notification listener.

This function should be called with identical parameters that were used to add the push notification listener with `hwc.addPushNotificationListener` on page 157.

### Syntax

```javascript
<static> removePushNotificationListener( PushNotificationListener, [containingObject] )
```

### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PushNotificationListener</code></td>
<td><code>anonymous.PushNotificationListener</code></td>
<td></td>
<td>The callback for push notifications.</td>
</tr>
<tr>
<td><code>containingObject</code></td>
<td><code>Object</code></td>
<td>(optional)</td>
<td>The containing object of the listener.</td>
</tr>
</tbody>
</table>

### Example

```javascript
// pushListener is the callback function that will be passed to hwc.addPushNotificationListener.
var pushListener = function( notifications )
{
    alert( "push notification:\n" + JSON.stringify(notifications) );
    return hwc.NOTIFICATION_CONTINUE;
}
```
} hwc.addPushNotificationListener( pushListener );
// At some other point if we want to remove the push listener, we call
the following line:
// pushListenerManager is an object that will contain the listener
// callback as well as a variable
var pushListenerManager = {};
// doSomething is a function that is called from inside the callback.
pushListenerManager.doSomething = function( notifications )
{
    alert( "push notification:\n" + JSON.stringify(notifications) );
    return hwc.NOTIFICATION_CONTINUE;
}
// This is the callback function.
pushListenerManager.listener = function( notifications )
{
    return this.doSomething( notifications );
}
// Since the callback function references variables in its containing
// object, the containing object
// must be passed to hwc.addPushNotificationListener as well.
// when we want to remove the push listener, we call the following
hwc.removePushNotificationListener( pushListenerManager.listener,

Source
hwc-api.js, line 1299 on page 342.

sample_AppListener( event, moduleId, version ) method
A sample anonymous.ApplicationListener on page 75 callback function.

Syntax
<static> sample_AppListener( event, moduleId, version )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

232 SAP Mobile Platform
event 

number

A number indicating what event has taken place (will be one of `hwc.APP_REFRESH` on page 126, `hwc.APP_ADDED` on page 126, `hwc.APP_UPDATED` on page 127, `hwc.APP_REMOVED` on page 127).

moduleId

number

The module id of the hybrid app the event is about.

version

number

The version of the hybrid app the event is about.

Source

`hwc-api.js`, line 1662 on page 355.

**sample_ConnectionListener( event, errorCode, errorMessage ) method**

A sample `anonymous.ConnectionStateListener` on page 77 callback function.

**Syntax**

```javascript
<static> sample_ConnectionListener( event, errorCode, errorMessage )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>number</td>
<td>A number indicating the event that occurred (will be <code>hwc.CONNECTED</code> on page 127 or <code>hwc.DISCONNECTED</code> on page 130).</td>
</tr>
<tr>
<td>errorCode</td>
<td>number</td>
<td>An error code (0 indicating success).</td>
</tr>
<tr>
<td>errorMessage</td>
<td>string</td>
<td>Text of the error message. Will be empty if there is no error.</td>
</tr>
</tbody>
</table>

Source

`hwc-api.js`, line 480 on page 312.
sample_InstallationAppListener( event, moduleId, version, moduleName, designerVersion, containerVersion ) method
Sample application listener callback function

Syntax
<static> sample_InstallationAppListener( event, moduleId, version, moduleName, designerVersion, containerVersion )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Integer</td>
<td>Installation flags including, BEGIN(1), END(2), FAIL(3)</td>
</tr>
<tr>
<td>moduleId</td>
<td>String</td>
<td>Optional Module Id</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>Optional Module version</td>
</tr>
<tr>
<td>moduleName</td>
<td>String</td>
<td>Optional Module display name</td>
</tr>
<tr>
<td>designerVersion</td>
<td>String</td>
<td>Optional Version of designer used to create app</td>
</tr>
<tr>
<td>containerVersion</td>
<td>String</td>
<td>Optional Version of hybrid web container</td>
</tr>
</tbody>
</table>

Source
hwc-api.js, line 1021 on page 332.

sample_LogListener( milliseconds, event, optionalString ) method
Sample anonymous.LogListener on page 79 callback function.

Syntax
<static> sample_LogListener( milliseconds, event, optionalString )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>milliseconds</td>
<td>number</td>
<td>The date of the log message represented in milliseconds.</td>
</tr>
</tbody>
</table>
event  |  number  | The that represents which category this event falls under (It will be one of `hwc.CONNECTION_ERROR` on page 128, `hwc.CONNECTION_OTHER` on page 129, `hwc.CONNECTION_CONNECTED` on page 128, `hwc.CONNECTION_DISCONNECTED` on page 128, `hwc.CONNECTION_RETRIEVED_ITEMS` on page 129).

optionalString  |  string  | The string carrying the message of the log event.

Source
`hwc-api.js`, line 855 on page 326.

**sample_MessageListener( flag, msgId ) method**
A sample anonymous.MessageListener on page 80 callback function.

**Syntax**

```
<static> sample_MessageListener( flag, msgId )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>number</td>
<td>A number indicating which message event occurred (will be one of MSG_* constants).</td>
</tr>
<tr>
<td>msgId</td>
<td>number</td>
<td>The message id of the affected message.</td>
</tr>
</tbody>
</table>

Source
`hwc-api.js`, line 2916 on page 400.

**sample_PushNotificationListener( notifications ) method**
A sample implementation of a anonymous.PushNotificationListener callback function.

**Syntax**

```
<static> sample_PushNotificationListener( notifications )
```
Develop Hybrid Apps Using Third-party Web Frameworks

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifications</td>
<td>Array</td>
<td>Array of notifications.</td>
</tr>
</tbody>
</table>

Source

*hwc-api.js*, line 1345 on page 344.

**saveLoginCertificate( certificate ) method**

This function saves login credentials from a certificate to the credential cache.

The common name is used for the username and the signed certificate is used for the password.

**Syntax**

<static> saveLoginCertificate( certificate )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificate</td>
<td>object</td>
<td>The values certificate.subjectCN and certificate.signedCertificate must be defined.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
var certInfo = {};
certInfo.subjectCN = "sampleCommonName";
certInfo.signedCertificate = "samplePassword";
hwc.saveLoginCertificate( certInfo );
```

Source

*hwc-comms.js*, line 971 on page 466.

**saveLoginCredentials( userName, password ) method**

This function saves login credentials to the credential cache.

**Syntax**

<static> saveLoginCredentials( userName, password )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>string</td>
<td>The user name to save</td>
</tr>
</tbody>
</table>
Develop Hybrid Apps Using Third-party Web Frameworks

| password | string | The password to save |

**Example**

hwc.saveLoginCredentials( "sampleUserName", "samplePassword" );

**Source**

*hwc-comms.js*, line 990 on page 467.

**saveSettings( settings ) method**

Save the connection settings to native application storage.

Device registration will be attempted if and only the following conditions are both satisfied.

1. The registration method is not manual. This can be passed in the hwc.ConnectionSettings object, or if that value is null, the currently configured value will be used.
2. The password must be non-empty. This value MUST be passed in the hwc.ConnectionSettings object.

hwc.startClient() needs to be called after hwc.saveSettings() for the device to complete automatic/manual registration.

**Usage Note:** It is not mandatory to specify a value for each hwc.ConnectionSettings property. Specifying a null or undefined for a hwc.ConnectionSettings property will effectively cause this method to IGNORE the property and not change it's value. If the saveSettings() operation fails, a non-zero number will be returned. See hwc.REG_ERR_* for device registration errors. There can be other types of errors not listed here.

**Syntax**

<static> saveSettings( settings ) {number}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>settings</td>
<td>hwc.ConnectionSettings on page 168</td>
<td>The connection settings to be saved.</td>
</tr>
</tbody>
</table>

**Returns**

A status code indicating success (hwc.SETTING_SUCCESS on page 148) or an error (one of hwc.REG_ERR_AUTO_REG_NOT_ENABLED on page 144, hwc.REG_ERR_AUTO_REG_TEMPLATE_NOT_FOUND on page 145, hwc.REG_ERR_AUTO_REG_USER_NAME_TOO_LONG on page 145, hwc.REG_ERR_AUTO_REG_WRONG_USER_FOR_DEVICE on page 145, hwc.REG_ERR_COULD_NOT_REACH_MMS_SERVER on page 146,
Develop Hybrid Apps Using Third-party Web Frameworks

*hwc.REG_ERR_INVALID_USER_NAME* on page 146,  
*hwc.REG_ERR_MMS_AUTHENTICATION_FAILED* on page 146).

Type:

number

**Example**

```javascript
// Load the connection settings.
var connectionSettings = hwc.loadSettings();
// Modify the connection settings.
connectionSettings.ServerName = "999.999.999.999";
// Save the modified connection settings.
hwc.saveSettings( connectionSettings );
// Start the client to for the device to complete automatic/manual registration.
hwc.startClient();
```

**Source**

*hwc-api.js, line 196 on page 301.*

**setLoggingAlertDialog( newAlertDialogCallback ) method**

This function sets the callback used by hwc.log when it is required to notify the user of a log item.

**Syntax**

<static> setLoggingAlertDialog( new AlertDialogCallback )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newAlertDialogCallback</td>
<td>anonymous.alertDialogCallbackFunction on page 73</td>
<td>The alert dialog to use.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
customLogAlert = function( message )
{
  alert( "New log message: " + message );
}
hwc.setLoggingAlertDialog( customLogAlert );
```

**Source**

*hwc-comms.js, line 207 on page 439.*
setLoggingCurrentLevel(newLoggingLevel) method
This function sets the logging level.

The logging level set with this function only persists as long as this javascript context does. When the hybrid app is closed, the value set with this function is lost.

Syntax
<static> setLoggingCurrentLevel(newLoggingLevel)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newLoggingLevel</td>
<td>number</td>
<td>The number representing the new logging level. Must be an integer in the range [1..4]. The higher numbers represent more verbose logging levels from 1 for ERROR level logging up to 4 for DEBUG level logging.</td>
</tr>
</tbody>
</table>

Example

```javascript
// Set the logging level to debug.
hwc.setLoggingCurrentLevel( 4 );
```

Source

`hwc-comms.js`, line 236 on page 441.

setReportErrorFromNativeCallback(callbackToSet) method
This function sets the callback function called when there is a native error reported.

Calling this function will replace any callback that had been set previously.

Syntax

<static> setReportErrorFromNativeCallback(callbackToSet)

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callbackToSet</td>
<td>function</td>
<td>The callback function.</td>
</tr>
</tbody>
</table>

Example

```javascript
var errorCallback = function( errorString )
{
    alert( "There was a native error: " + errorString );
}
```
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
hwc.setReportErrorFromNativeCallback( errorCallback );
```

**Source**
*hwc-comms.js, line 274 on page 442.*

**setScreenTitle_CONT( screenTitle ) method**
Sets the title of the screen.

**Syntax**
<static> setScreenTitle_CONT( screenTitle )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>screenTitle</td>
<td>string</td>
<td>The screen title to use.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
hwc.setScreenTitle_CONT( "Custom Screen Title" );
```

**Source**
*hwc-comms.js, line 508 on page 450.*

**SharedStorage() method**
Storage

**Syntax**
<static> SharedStorage()

**Source**
*SUPStorage.js, line 379 on page 573.*

**showAlertDialog( message, [title] ) method**
Displays an alert dialog to the user.

This function blocks until it receives a response from the user.

**Syntax**
<static> showAlertDialog( message, [title] )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>title</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
message | string | The message to display  
---|---|---  
title | string | (optional) | The title doesn't actually get displayed.

**Example**
```
var openAttachmentBase64StringPng = function()
{
  // How you want get the base 64 encoding of the file is up to you. 
  // This string represents a small png image.
  var data = "iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAAAXNSR0IArs4c6QAAAA
  RnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAA

```

**Source**
*hwc-comms.js, line 1522 on page 486.*

**showAttachmentContents_CONT( contents, mimeType, fileName, waitDialogCallbackString ) method**
Shows the given file contents in a content-appropriate way.

The type of the content is supplied by either the MIME type or the filename, at least one of which must be supplied. The content itself should be presented as a base64-encoded string. Not all file types may be supported on all platforms.

**Syntax**
```
<static> showAttachmentContents_CONT( contents, mimeType, fileName, waitDialogCallbackString )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
<td>string</td>
<td>The base-64 encoded version of the binary content of the attachment to be displayed.</td>
</tr>
<tr>
<td>mimeType</td>
<td>string</td>
<td>The MIME type of the file.</td>
</tr>
<tr>
<td>fileName</td>
<td>string</td>
<td>The name of the file.</td>
</tr>
<tr>
<td>waitDialogCallbackString</td>
<td>anonymous.genericCallback-Function on page 78</td>
<td>The callback function used to close a wait dialog once the attachment is done opening.</td>
</tr>
</tbody>
</table>

**Example**
```
var openAttachmentBase64StringPng = function()
{
  // How you want get the base 64 encoding of the file is up to you. 
  // This string represents a small png image.
  var data = "iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAAAXNSR0IArs4c6QAAAA
  RnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAA
```

Developer Guide: Hybrid Apps 241
var openAttachmentBase64StringTxt = function() {
    // How you want get the base 64 encoding of the file is up to you. This string represents a short text file.
    var data = "VGhpcyBpcyBwYXJ0IG9mIGEgaHlicmlkIGFwcC4=";
    // Don't have to pass the MIME type because we are passing the filename.
    hwc.showAttachmentContents_CONT( data, null, "attach.txt" );
}

Source
hwc-comms.js, line 1070 on page 470.

showAttachmentFromCache_CONT( uniqueKey, mimeType, fileName, waitDialogCallbackString ) method
Shows the given file contents in a content-appropriate way.

The type of the content is supplied by either the MIME type or the filename, at least one of which must be supplied. The content itself will be a unique key supplied earlier to a call to doAttachmentDownload.

Syntax
<static> showAttachmentFromCache_CONT( uniqueKey, mimeType, fileName, waitDialogCallbackString )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniqueKey</td>
<td>string</td>
<td>The unique key for the attachment.</td>
</tr>
<tr>
<td>mimeType</td>
<td>string</td>
<td>The MIME type of the file.</td>
</tr>
<tr>
<td>fileName</td>
<td>string</td>
<td>The name of the file.</td>
</tr>
<tr>
<td>waitDialogCallbackString</td>
<td>string</td>
<td>string with the value for the 'callback=' parameter.</td>
</tr>
</tbody>
</table>

Source
hwc-comms.js, line 1106 on page 471.
**showCertificatePicker() method**
This function opens a form on the device that allows the user to specify the credentials for the use of certificate-based authentication.

If the user picks a certificate, then that certificate is saved in the credentials cache.

**Syntax**
<static> showCertificatePicker()

**Example**
```
hwc.showCertificatePicker();
```

**Source**
hwc-comms.js, line 948 on page 465.

**showConfirmDialog( message, [title] ) method**
Shows a confirm dialog to the user.

This function blocks until it receives a response from the user.

**Syntax**
<static> showConfirmDialog( message, [title] ) → {boolean}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>string</td>
<td></td>
<td>The message to display in the dialog.</td>
</tr>
<tr>
<td>title</td>
<td>string</td>
<td>(optional)</td>
<td>The title doesn't actually get displayed.</td>
</tr>
</tbody>
</table>

**Returns**
The user's choice from the confirm dialog.

Type:
boolean

**Example**
```
var userConfirm = hwc.showConfirmDialog( "Are you sure you want to see an alert message?", "Confirm Alert" );
if( userConfirm )
{
    alert( "This is what you wanted." );
}
```
**showLocalAttachment( key ) method**

Shows a local attachment.

**Syntax**

<static> showLocalAttachment( key )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>The key of the attachment. This is the path to the file, with the root being the folder that manifest.xml is located.</td>
</tr>
</tbody>
</table>

**Example**

```
        hwc.showLocalAttachment( "html/images/samplePic.gif" );
```

**showProgressDialog( [message] ) method**

This function shows a progress dialog with spinner.

The dialog created by this function will block all user input until `hwc.hideProgressDialog` on page 197 is called. It is important to be sure that `hwc.hideProgressDialog` on page 197 will be called after a call to this function.

**Syntax**

<static> showProgressDialog( [message] )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**message**

string

(optional)

The message to show on the progress dialog. This message is displayed on Android platforms only - other platforms show only a spinner.

**Example**

```javascript
var showProgress = function()
{
    hwc.showProgressDialog( "a message" );
    setTimeout( hideProgress, 10000 );
}

var hideProgress = function()
{
    hwc.hideProgressDialog();
}
```

**Source**

*hwc-comms.js, line 1475 on page 485.*

**showUrlInBrowser( url ) method**

This function opens the supplied URL in a browser.

The browser opens on top of the hybrid app - the context of the hybrid app is undisturbed.

**Syntax**

<static> showUrlInBrowser( url )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>string</td>
<td>The URL to be shown in a browser.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
hwc.showUrlInBrowser( "http://www.google.com" );
```

**Source**

*hwc-comms.js, line 1017 on page 468.*
**shutdown() method**

Shut down the client connection to the SUP server.

Companion function to `hwc.startClient` on page 246. If a hybrid app is running in the context of the Hybrid Web Container, then it will probably never have to call this function. If you want to temporarily stop the connection, then call `hwc.disconnectFromServer` on page 175 instead.

**Syntax**

```javascript
<static> shutdown()
```

**Example**

```javascript
hwc.shutdown();
```

**Source**

`hwc-api.js, line 564` on page 315.

---

**startClient([onNotification]) method**

Start the client connection to the SUP server.

Companion function to `hwc.shutdown` on page 246. If a hybrid app is running in the context of the Hybrid Web Container then it will probably never have to call this function unless `hwc.shutdown` on page 246 client was called first.

**Syntax**

```javascript
<static> startClient([onNotification])
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onNotification</code></td>
<td><code>anonymous.LogListener</code> on page 79</td>
<td>(optional)</td>
<td>A log listener callback function. If you are interested in the connection state it is recommended that you call <code>hwc.addConnectionListener</code> on page 151 before calling <code>hwc.startClient</code>.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
hwc.startClient();

// Add a log listener while calling hwc.startClient.
var logListener = function( time, event, message )
{
```
alert(message);
}

hwc.startClient( logListener );

**Source**
*hwc-api.js, line 531 on page 314.*

**this.getIconUrl( processed ) method**

Gets the URL of this custom icon.

It is possible to call this function directly, but generally it is easier simply to call
*hwc.getAppIconUrl* on page 177 or *hwc.getMsgIconUrl* on page 189. Those functions handle both cases where there is and isn't a custom icon for the hybrid app or message.

**Syntax**

<static> this.getIconUrl( processed ) {string}

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processed</td>
<td>boolean</td>
<td>When set to true, the URL of the processed icon will be returned. When set to false, the URL of the unprocessed icon will be returned.</td>
</tr>
</tbody>
</table>

**Returns**
The URL to the target icon.

Type:

string

**Example**

```javascript
var apps = hwc.getInstalledApps();
var app = apps[0];
// If app doesn't have a custom icon, then customIcon will be null.
var customIcon = app.getDefaultCustomIcon();
if( customIcon != null )
{
    // Create the image element.
    var image = document.createElement( "img" );
    // Set the source of the image to the icon URL.
    image.setAttribute( 'src', customIcon.getIconUrl() );
    // Add the image element to the page.
    document.body.appendChild( image );
}
```
updateMessageProcessed( msgId, status ) method
Updates the message processed status.

Syntax
<static> updateMessageProcessed( msgId, status )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>number</td>
<td>The id of message to update the processed status for.</td>
</tr>
<tr>
<td>status</td>
<td>boolean</td>
<td>Whether the message will be set to processed (true) or unprocessed (false).</td>
</tr>
</tbody>
</table>

Example
// set all messages as processed
var messages = hwc.getAllMessages();
for( var index = 0; index < messages.length; index++ )
{
    hwc.updateMessageProcessed( messages[index].getMessageId(), true );
}

updateMessageRead( msgId, status ) method
Updates the message read status.

Syntax
<static> updateMessageRead( msgId, status )

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msgId</td>
<td>number</td>
<td>The id of message to update the read status for.</td>
</tr>
<tr>
<td>status</td>
<td>boolean</td>
<td>Whether the message will be set to read (true) or unread (false).</td>
</tr>
</tbody>
</table>

**Example**

```javascript
// set all messages as read
var messages = hwc.getAllMessages();
for( var index = 0; index < messages.length; index++ )
{
    hwc.updateMessageRead( messages[index].getMessageId(), true );
}
```

**Source**

*hwc-api.js, line 3075 on page 406.*

**activationRequired() method**

Deprecated: Deprecated since version 2.2 - use hwc.activationRequired()

**Syntax**

`activationRequired()`

**Source**

*hwc-comms.js, line 77 on page 435.*

**clearCache() method**

Deprecated: Deprecated since version 2.2 - use hwc.clearCache()

**Syntax**

`clearCache()`

**Source**

*hwc-comms.js, line 52 on page 434.*

**clearCachetItem() method**

Deprecated: Deprecated since version 2.2 - use hwc.clearCacheItem( cachekey )

**Syntax**

`clearCacheItem()`

**Source**

*hwc-comms.js, line 47 on page 434.*
closeWorkflow() method
Deprecated: Deprecated since version 2.2 - use hwc.close()

Syntax
closeWorkflow()

Source
hwc-comms.js, line 42 on page 433.

expireCredentials() method
Deprecated: Deprecated since version 2.2 - use hwc.close()

Syntax
expireCredentials()

Source
hwc-comms.js, line 57 on page 434.

getXMLHTTPRequest() method
Deprecated: Deprecated since version 2.2 - use hwc.getXMLHTTPRequest()

Syntax
getXMLHTTPRequest()

Source
hwc-comms.js, line 32 on page 433.

guid() method
Deprecated: Deprecated since version 2.2 - use hwc.guid()

Syntax
guid()

Source
hwc-comms.js, line 27 on page 433.

logToWorkflow() method
Deprecated: Deprecated since version 2.2 - use hwc.log(sMsg, eLevel, notifyUser)
Syntax
logToWorkflow()

Source
hwc-comms.js, line 37 on page 433.

**markAsActivated() method**
Deprecated: Deprecated since version 2.2 - use hwc.markAsActivated()

Syntax
markAsActivated()

Source
hwc-comms.js, line 92 on page 435.

**markAsProcessed() method**
Deprecated:Deprecated since version 2.2 - use hwc.markAsProcessed()

Syntax
markAsProcessed()

Source
hwc-comms.js, line 87 on page 435.

**processDataMessage( incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType ) method**
Delegate for data message processing details.

In the custom case, the user is expected to provide their own implementation. In the default SUP HybridApp case, this updates values then sets the next screen to navigate to.

Syntax
processDataMessage( incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType )

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incomingDataMessageValue</td>
<td>string</td>
<td>The XML formatted string for the incoming message</td>
</tr>
</tbody>
</table>
Develop Hybrid Apps Using Third-party Web Frameworks

<table>
<thead>
<tr>
<th>noUI</th>
<th>boolean</th>
<th>true if this has no UI</th>
</tr>
</thead>
<tbody>
<tr>
<td>loading</td>
<td>boolean</td>
<td>If true, this is being called while the application is loading</td>
</tr>
<tr>
<td>fromActivationFlow</td>
<td>boolean</td>
<td>If true, this is being called from within an activation flow</td>
</tr>
<tr>
<td>dataType</td>
<td>string</td>
<td>If supplied, the data type of the value display on target screen</td>
</tr>
</tbody>
</table>

Source
hwc-comms.js, line 104 on page 436.

**processWorkflowMessage() method**

Deprecated: Deprecated since version 2.2 - use hwc.processDataMessage(incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType)

**Syntax**

```
processWorkflowMessage()
```

Source
hwc-comms.js, line 118 on page 436.

**saveLoginCertificate() method**

Deprecated: Deprecated since version 2.2 - use hwc.saveLoginCertificate(certificate)

**Syntax**

```
saveLoginCertificate()
```

Source
hwc-comms.js, line 67 on page 434.

**saveLoginCredentials() method**

Deprecated: Deprecated since version 2.2 - use hwc.saveLoginCredentials(userName, password)

**Syntax**

```
saveLoginCredentials()
```
showCertificatePicker() method
Deprecation: Deprecated since version 2.2 - use hwc.showCertificatePicker()

Syntax
showCertificatePicker()

Source
hwc-comms.js, line 72 on page 435.

showUrlInBrowser() method
Deprecation: Deprecated since version 2.2 - use hwc.showUrlInBrowser(url)

Syntax
showUrlInBrowser()

Source
hwc-comms.js, line 62 on page 434.

Source code

```javascript
Callbacks.js
1    /*
2    * Sybase Hybrid App version 2.3.4
3    *
4    * Callbacks.js
5    * This file will not be regenerated, so it is possible to modify it, but it
6    * is not recommended.
7    *
8    * Copyright (c) 2012 Sybase Inc. All rights reserved.
9    */
10
11    /**
```
The namespace for the Hybrid Web Container javascript

```
15     hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc;
       // SUP 'namespace'
18     (function(hwc, window, undefined) {
19
22          * Constructs CallbackSet object. This object is not meant
23          * for general use.
24          * @private
25          * @constructor
26          * @memberOf hwc
27          */
26     hwc.CallbackSet = function() {
27          hwc.CallbackSet.setCount++;
28     this.setId = hwc.CallbackSet.setCount;
29     };
31
36     hwc.CallbackSet.setCount = 0;
38
39     */
```
hwc.CallbackSet.callbacks = {};

/**
 * Registers a callback to be handled from container
 * @memberOf hwc.CallbackSet
 * @private
 * @param {string} methodName The name of the callback.
 * @param {function} callback The function pointer to the callback
 * @returns {string} callbackId that can be used by the container
 */

hwc.CallbackSet.prototype.registerCallback = function (methodName, callback) {
    if (!hwc.CallbackSet.callbacks[this.setId]) {
        hwc.CallbackSet.callbacks[this.setId] = {};
    }

    hwc.CallbackSet.callbacks[this.setId][methodName] = callback;
    return this.setId + ':' + methodName;
};

/**
 * Invoked asynchronously to handle callback from container
 * @memberOf hwc.CallbackSet
 * @static
 * @private
 * @param {string} callbackId The id of the callback. Format is "setid:methodname"
 * @param {boolean} removeSet True if the callback set should be removed
 * @param {array} args The arguments to be passed to the registered callback
 */

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps 255
```javascript
hwc.CallbackSet.callbackHandler = function(callbackId, removeSet, args) {
    var callbackSet, c, callback;
    c = callbackId.split(':', 2);

    if (c && c.length === 2) {
        callbackSet = hwc.CallbackSet.callbacks[c[0]];

        if (callbackSet) {
            callback = callbackSet[c[1]];

            if (removeSet) {
                delete hwc.CallbackSet.callbacks[c[0]];
            }

            if (callback) {
                callback.apply(callback, args);
            }
        }
    }

    window.CallbackSet = [];
    window.CallbackSet.callbackHandler = hwc.CallbackSet.callbackHandler;

})(hwc, window);
```
Camera.js

1 /*
2   * Sybase Hybrid App version 2.3.4
3   *
4   * Camera.js
5   * This file will not be regenerated, so it is possible to
6   * modify it, but it
7   * is not recommended.
8   *
9   * Copyright (c) 2012 Sybase Inc. All rights reserved.
10 */

/* The feature comment is necessary at the class level for
the custom template to work. */

/**
 * The namespace for the Hybrid Web Container javascript
 * @namespace
 * @desc Camera
 */

hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc;  // SUP 'namespace'

(function(hwc, window, undefined) {

/**
* An array that holds all possible option codes for use
with getPicture()
* @private
*/

hwc.PictureOption = [];

/**
*/
hwc.PictureOptionSourceType = {  
    CAMERa: 1,  // Specifies the built-in camera as the image source where image content is not persisted by the device
    PHOTOLIBRARY: 2,  // Specifies the photo library as the image source where image content is already persisted on the device
    BOTH: 3  // Specifies the built-in camera as the image source where image content is persisted by the device
};

/**  
* Use this constant to specify that base64 encoded image data be returned by the {@link hwc.getPicture} method.
*/

hwc.PictureOptionDestinationType = {  
    **
};

/**  
* Use this constant to specify that base64 encoded image data be returned by the {@link hwc.getPicture} method.
*/

Develop Hybrid Apps Using Third-party Web Frameworks
* @memberOf hwc.PictureOption.DestinationType
* @deprecated
*
* IMAGE_DATA: 0, // Returns base64 encoded string (deprecated)
*/

* Use this constant to specify that the image URI be returned by the {@link hwc.getPicture} method.
* @memberOf hwc.PictureOption.DestinationType
*/

* IMAGE_URI: 1 // Returns uniform reference identifier for the image
*/

/**
 * Open a platform-specific application allowing the user to capture an image
 * using the built-in camera.
 * @deprecated
 */

hwc.PictureOption.CAMERA = hwc.PictureOption.SourceType.CAMERA;

/**
 * Open a platform-specific application allowing the user to select an existing picture from a gallery.
 * @deprecated
 */

hwc.PictureOption.PHOTOLIBRARY = hwc.PictureOption.SourceType.PHOTOLIBRARY;

/**
 * An array that holds all possible error codes
 */
83          hwc.PictureError = [];
84
85          /**
86           * Constant indicating that the {@link hwc.getPicture} method was successful.
87           * @memberOf hwc
88           */
89          hwc.PictureError.NO_ERROR      =  0;
90
91          /**
92           * Constant indicating that the {@link hwc.getPicture} method is not implemented, camera not present, etc.
93           * @memberOf hwc
94           */
95          hwc.PictureError.NOT_SUPPORTED = -1;
96
97          /**
98           * Constant indicating that the {@link hwc.getPicture} method has been invoked, but has not completed yet.
99           * @memberOf hwc
100          */
101         hwc.PictureError.IN_PROGRESS   = -2;
102
103          /**
104           * Constant indicating that the user has cancelled the {@link hwc.getPicture} invocation.
105           * @memberOf hwc
106           */
107         hwc.PictureError.USER_REJECT   = -3;
108
109          /**
110           * Constant indicating that the supplied options were not recognized by the {@link hwc.getPicture} method
111           * @memberOf hwc
hwc.PictureError.BAD_OPTIONS = -4;

/**
 * Constant indicating that the returned image size was too large to be handled by JavaScript.
 *
 * @memberOf hwc
 */

hwc.PictureError.TOO_LARGE = -5;

/**
 * Constant indicating that an unknown error occurred during the execution of \{@link hwc.getPicture\} method.
 *
 * @memberOf hwc
 */

hwc.PictureError.UNKNOWN = -6;

/**
 * A namespace for our private use
 *
 * @private
 */

var _Picture = new function() {};

// private object '_Picture' within 'hwc'

/**
 * Requests retrieval of a picture asynchronously.
 *
 * @param {anonymous.onGetPictureError} onGetPictureError
 * Function to be invoked if the attempt to get a picture fails. err will be one of the PictureError codes.
 *
 * @param {anonymous.onGetPictureSuccess} onGetPictureSuccess
 * Function to be invoked if a picture is successfully retrieved. response will either be a Base64-encoded JPG string or a URI.
 */
Develop Hybrid Apps Using Third-party Web Frameworks

```
140    * @param {anonymous.PictureOptions} options the options
to control the sourceType and destinationType.
141    * @desc Camera
142    * @memberOf hwc
143    * @public
144    * @example
145    * // Error handler. will be invoked asynchronously.
146    * fail = function(errorCode){
147    *   // handle error code and take appropriate
148    *   action.
149    * }
150    * // Success handler. will be invoked asynchronously.
151    * success = function(fileName, content){
152    *   // handle the content. content may be a location or
153    *   base64 encoded string that is
154    *   // determined by the options passed to the
155    *   destinationType argument.
156    * }
157    * getPicture(fail,
158    *    success,
159    *    { sourceType:
160    *      PictureOption.SourceType.CAMERA,
161    *       destinationType:
162    *      PictureOption.DestinationType.IMAGE_URI
163    *    });
164    */
165    hwc.getPicture = function(onGetPictureError,
166    onGetPictureSuccess, options)
167    {
168      hwc.traceEnteringMethod("hwc.getPicture");
169      try {
170        // Return if callback functions are not
171        if (typeof onGetPictureError !== 'function' ||

typeof onGetPictureSuccess !== 'function')
{
    return;
}

if ("_onGetPictureSuccess" in _Picture &&
    _Picture._onGetPictureSuccess !== null)
{
    // Already requested but not yet complete
    onGetPictureError(hwc.PictureError.IN_PROGRESS);
    return;
}

_Picture._onGetPictureError =
onGetPictureError;

_Picture._onGetPictureSuccess =
onGetPictureSuccess;

// Convert options parameter to object notation if number type and return image data to preserve behavior
// of previous release
if (typeof options === 'number') {
    options = { destinationType: hwc.PictureOption.DestinationType.IMAGE_DATA,
    sourceType: options
};

// Convert options object to serialized JSON text in preparation for submission to the container
options = JSON.stringify(options);

if (hwc.isWindowsMobile())
{

hwc.getDataFromContainer("getPicture", "PictureOptions=" + encodeURIComponent(options));

else if (hwc.isIOS())
{
    // Only difference between iOS and WindowsMobile
    // above is the leading '&'
    hwc.getDataFromContainer("getPicture", 
        
    }
}
else
{
    _HWC.getPicture(options);
}
} finally {
    hwc.traceLeavingMethod("hwc.getPicture");
}

/**
 * (Internal) Invoked asynchronously when the image arrives.
 *
 * @private
 * @param result The PictureError code, or PictureError.NO_ERROR for
 *     success.
 * @param {string} filename Filename corresponding to the image.
 * @param {string} imageData Base64-encoded String containing the image data. Undefined
 *     if the result parameter indicates an error or the image URI was requested.
 * @param {string} imageUri Uniform resource indicator of the image resource. Undefined
if the result parameter indicates an error or the image data was requested.

*/

__Picture._getPictureComplete = function(result, fileName, imageData, imageUri) {

    var response, successFunc, errorFunc;

    try {
        successFunc = __Picture._onGetPictureSuccess;
        errorFunc = __Picture._onGetPictureError;

        __Picture._onGetPictureSuccess = null;
        __Picture._onGetPictureError = null;

        if (result === hwc.PictureError.NO_ERROR) {
            if (imageData) {
                // For WM client, the picture data is too big to be passed from url, so only
                // the unique key is sent from container to JavaScript. JavaScript needs to send
                // another xmlhttprequest to fetch the actual data
                if (hwc.isWindowsMobile()) {
                    response = hwc.getDataFromContainer("getpicturedata", "pictureid=" + imageData);
                    successFunc(fileName, response);
                } else {
                    successFunc(fileName, imageData);
                }
            } else if (imageUri) {
                successFunc(fileName, imageUri);
            } else {
                successFunc(fileName, imageUri);
            }
        } else {
            errorFunc(fileName, result);
        }
    } finally {
        __Picture._onGetPictureSuccess = null;
        __Picture._onGetPictureError = null;
    }
}

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps
errorFunc(hwc.PictureError.UNknown);

} else {
  errorFunc(result);
}

} finally {
  hwc.traceLeavingMethod("_Picture._getPictureComplete");

};

window._Picture = _Picture;
})(hwc, window);

/**
 * Used to group anonymous objects and callback functions used as method parameters. Methods and fields in this
 * namespace cannot be instantiated. Used for API docs generation only.
 *
 * @namespace
 * @*/

anonymous = (typeof anonymous === "undefined" || !anonymous) ? {} : anonymous; // SUP 'namespace'

/**
 * User provided function that is invoked when the {link hwc.getPicture} function fails.
 *
 * @anonymous.onGetPictureError
 * @param {number} err the error code returned. Possible values are
 * <ol>
 * <li>PictureError.NO_ERROR = 0;</li>
 * </ol>
 */
* PictureError.NOT_SUPPORTED = -1; getPicture() not implemented, camera not present,<li>
* PictureError.IN_PROGRESS = -2; getPicture() has already been requested but has not yet completed.</li>
* PictureError.USER_REJECT = -3; the user has canceled the request.</li>
* PictureError.BAD_OPTIONS = -4; supplied options were not recognized.</li>
* PictureError.TOO_LARGE = -5; the returned image size was too large to be handled by JavaScript</li>
* PictureError.UNKNOWN = -6; an unknown error occurred.</li>

* User provided function that will be invoked when the {link hwc.getPicture} function is successful.

* @name anonymous.onGetPictureSuccess

* @param {string} filename file name of the image

* @param {string} response the response will be either a Base64-encoded JPG string or a URI depending on the options passed to

* the {link hwc.getPicture} function.
Options object that is used with the \{link hwc.getPicture\} method. Contains 2 fields that can be specified.

* <ul>
  * sourceType: One of \{link hwc.Picture.SourceType\} values
  * destinationType: One of \{link hwc.Picture.DestinationType\} values

* </ul>

* @name anonymous.PictureOptions
* @see hwc.getPicture for an example.

Certificate.js

Sybase Hybrid App version 2.3.4

This file will not be regenerated, so it is possible to modify it, but it is not recommended.

* Last Updated: 2011/6/29

* Copyright (c) 2012 Sybase Inc. All rights reserved.

* Note a certificate object will have the following fields
  - issuerCN - The common name (CN) from the certificate issuer's distinguished name.
  - issuerDN - The certificate issuer's distinguished name, in string form.
  - notAfter - End time for certificate's validity period, with date/time fields as they would appear in UTC.
- notBefore - Start time for the certificate's validity period, with date/time fields as they would appear in UTC.

- signedCertificate - The digitally signed certificate in Base64 format

- subjectCN - The common name (CN) from the certificate subject's distinguished name.

- subjectDN - The certificate subject's distinguished name, in string form.

*/

/**
 * This class represents an X.509 public certificate store.
 */

/**
 * The namespace for the Hybrid Web Container javascript
 * @namespace
 */

hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc; // SUP 'namespace'

(function(hwc, window, undefined) {

/**
 * Use these functions for X.509 credential handling.
 * <p>
 * Use these functions to create a user interface in HTML and JavaScript, that uses X.509 certificates as the Workflow credentials.
 * </p>
 * 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.
 */
```javascript
hwc.CertificateStore = function() {
};

(function() {

    /**
     * Private function
     * Convert string type date to JavaScript Date
     * Format: 2014-05-24T20:00:12Z -> Sat May 24 2014 16:00:12 GMT-0400 (Eastern Daylight Time)
     */
    function parseCertDate(value) {
        var a = /^\d{4}-(\d{2})-(\d{2})T(\d{2}):(\d{2}):(\d{2}(?:\d*)?)Z$/i.exec(value);
        return new Date(Date.UTC(+a[1], +a[2] - 1, +a[3], +a[4], +a[5], +a[6]));
    }

    /**
     * Private function
     * Create certificate object
     */
    function createCertObject(value) {
        var JSONCertObj = JSON.parse(value);
        return {
            subjectDN: JSONCertObj.subjectDN,
            notBefore: JSONCertObj.notBefore,
            notAfter: JSONCertObj.notAfter
        };
    }

})
```
* "subjectCN":"android",
* "signedCertificate":"base64 encoded string here",
* "issuerDN":"CN=teva, CN=sybase.com, OU=Unwired Enterprise, O=Sybase Inc., L=Dublin, ST=California, C=US",
* "issuerCN":"teva"}

* @returns Certificate object

*/

function createCert(value) {
    var cert;
    if (value === null || typeof value === 'undefined' || value.length === 0) {
        return null;
    }
    cert = JSON.parse(value);
    if (cert.notAfter) {
        cert.notAfter = new Date(parseCertDate(cert.notAfter));
    }
    if (cert.notBefore) {
        cert.notBefore = new Date(parseCertDate(cert.notBefore));
    }
    return cert;
}

* Returns a list of all the certificate labels in this store (can be empty). Each certificate in this store has a unique label.
Supported Platforms: Windows Mobile and BlackBerry.

@desc Certificate
@public
@memberOf hwc.CertificateStore
@param {String} filterSubject filter of subject
@param {String} filterIssuer filter of issuer
@returns {String[]} Only filtered certificate labels
@example

// 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");

hwc.CertificateStore.prototype.certificateLabels = function(filterSubject, filterIssuer) {
    var response = "";
    hwc.traceEnteringMethod("hwc.CertificateStore.certificateLabels");
    try {
        filterSubject = filterSubject ? filterSubject : "";
        filterIssuer = filterIssuer ? filterIssuer : "";

        if (hwc.isWindowsMobile()) {
            response = hwc.getDataFromContainer("certificatestore", "&command=certificateLabels" +
                
            )
        } else if (hwc.isBlackBerry()) {

        }
    }
response = _HWC.getCertificateLabels(filterSubject, filterIssuer);

else {
    throw "Not supported on this platform";
}

return eval('(' + response + '));

hwc.traceLeavingMethod("hwc.CertificateStore.certificateLabels");

/**
 * Returns a certificate without the signedCertificate part set.
 *
 * @desc Certificate
 * @public
 * @memberOf hwc.CertificateStore
 * @returns {hwc.CertificateStore} a certificate without the signedCertificate part set
 */

hwc.CertificateStore.getDefault = function() {
    return new hwc.CertificateStore();
};

/**
 * Returns a certificate without the signedCertificate part set.
 *
 * <b> Supported Platforms </b>: Windows Mobile and BlackBerry.
 *
 * @desc Certificate

153         * @public
154         * @memberOf hwc.CertificateStore
155         * @param {String} label label of the desired certificate
156         * @returns certificate object
157         * @example
158         * // The following script gets the certificate data for the first certificate to match the provided subject and issuer
159         * // certificate to match the provided subject and issuer
160         * var certStore = CertificateStore.getDefault();
161         * var labels = certStore.certificateLabels("MyUser", "mydomain.com");
162         * var cert = certStore.getPublicCertificate(labels[0]);
163         */
164         hwc.CertificateStore.prototype.getPublicCertificate = function(label) {
165             var response = "";
166
168             try {
169                 if (hwc.isWindowsMobile()) {
170                     response = hwc.getDataFromContainer("certificatestore", 
171                         
172             encodeURIComponent("&command=getPublicCertificate" +
173                         
174             
175             }
176             else if (hwc.isBlackBerry()) {
177                 response = _HWC.getPublicCertificate(label);
178             }
179             else {
180                 throw "Not supported on this platform";
181             }
return createCert(response);

} finally {
  
  hwc.traceLeavingMethod("hwc.CertificateStore.getPublicCertificate");

  }
}

/**
  * Returns the certificate with the specified label, and
decrypts it if necessary using the specified password,
or returns null if the certificate is encrypted and the
password is incorrect.
  *
  * <b>Supported Platforms</b>: Windows Mobile and BlackBerry

  * @desc Certificate

  * @public

  * @memberOf hwc.CertificateStore

  * @param {String} label label of the desired
certificate

  * @param {String} password Access password for the private
key of the certificate. Pass null unless the platform requires a
password.

  * @returns Certificate object

  * @example

  * // 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]);
hwc.CertificateStore.prototype.getSignedCertificate = function(label, password) {
    var response = "";

    hwc.traceEnteringMethod("hwc.CertificateStore.getSignedCertificate");

    try {
        if (hwc.isWindowsMobile()) {
            response = hwc.getDataFromContainer("certificatestore", 
                "&command=getSignedCertificate" +
                "&label=" +
                    encodeURIComponent(label));
        } else if (hwc.isBlackBerry()) {
            response = _HWC.getSignedCertificate(label);
        } else {
            throw "Not supported on this platform";
        }
    } finally {

        hwc.traceLeavingMethod("hwc.CertificateStore.getSignedCertificate") ;

    }

    return createCert(response);

}

/**
 * Returns a list of full path names for the certificate files found in the

* file system for import.

* <b>Supported Platforms</b>: Android

* @desc Certificate

* @memberOf hwc.CertificateStore

* @public

* @param {String} sFolder Folder in which to search for files. This should be a full absolute path, based on the root of the device file system. The separator may be either "/" or "\". For example, "\sdcard\mycerts"

* @param {String} sFileExtension File extension to which the list should be restricted. Pass the string expected after the "." in the file name. For example, to match *.p12, pass "p12" as the argument.

* @returns {String[]} A list of Strings, each String being the full path name of a matched file in the given folder.

* @example

  // 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");

  /*
  * hwc.CertificateStore.prototype.listAvailableCertificatesFromFileSystem = function(sFolder, sFileExtension) {
```javascript
var response = "";

hwc.traceEnteringMethod("hwc.CertificateStore.listAvailableCertificatesFromFileSystem");

try {
    if (hwc.isAndroid()) {
        response = _HWC.listAvailableCertificatesFromFileSystem(sFolder, sFileExtension);
    } else {
        throw "Not supported on this platform";
    }

    return eval('(' + response + ')');
} finally {
    hwc.traceLeavingMethod("hwc.CertificateStore.listAvailableCertificatesFromFileSystem");
}

/**
* Gets a certificate from a file.
* <b>Supported Platforms</b>: Android
* @desc Certificate
* @public
* @memberOf hwc.CertificateStore
* @param {String} filePath The absolute path to the file.
* @param {String} password The password needed to access the certificate's private data.
* @example
* // 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");
*/

hwc.CertificateStore.prototype.getSignedCertificateFromFile =
function(filePath, password) {
    var response = "";

    hwc.traceEnteringMethod("hwc.CertificateStore.getSignedCertificateFromFile");

    try {
        if (hwc.isAndroid()) {
            response = _HWC.getSignedCertificateFromFile(filePath, password);
        } else if (hwc.isIOS()) {
            response =
                hwc.getDataFromContainer("certificatestore",
                
                
                
                
                
                } else {
                throw "Not supported on this platform";

            return createCert(response);
        } finally {
    
    hwc.traceLeavingMethod("hwc.CertificateStore.getSignedCertificateFromFile");

}
hwc.CertificateStore.prototype.getSignedCertificateFromServer =
function(username, serverPassword, certPassword) {
    var response = "";
    
    hwc.traceEnteringMethod("hwc.CertificateStore.getSignedCertificateF
    romServer");
    try {
        if (hwc.isIOS()) {
            
            } else {
            
            
        } else {
            
            
        }
    } catch (error) {
        
    }
}
response = hwc.getDataFromContainer("certificatestore", 
"&command=getSignedCertificateFromServer" +
 "&username=" +
 encodeURIComponent(username) + "&serverPassword=" +
 encodeURIComponent(serverPassword) +
 "&certPassword=" +
 encodeURIComponent(certPassword));

} else {

    throw "Not supported on this platform";

}

}

return eval('(' + response + ')');

} finally {

hwc.traceLeavingMethod("hwc.CertificateStore.getSignedCertificateFromServer");

}

/**
 * Gets a certificate from the Afaria server.
 * To retrieve an x509 certificate from Afaria, you must get a CertificateStore and then call getSignedCertificateFromAfaria. If Afaria is installed and configured on the device, this gets the Afaria seeding file from the Afaria server.
 * If the seeding file is retrieved from the Afaria server, the user is prompted to update user specific information in the Settings screen.
 * 
 * <b>Supported Platforms</b>: iOS, Android & BlackBerry
 * @desc Certificate
 * @public
 * @memberOf hwc.CertificateStore
 * @param {String} commonName Common name used to generate the certificate by Afaria
 * @param {String} challengeCode Challenge code for the user so that CA can verify and sign it
* @returns JSON object with CertBlob in Base64 encoded format and other information about certificate

* @throws If called on a platform that is not supported.

* @example

  // The following script gets a signed certificate from the Afaria server.

  var certStore = CertificateStore.getDefault();

  var cert = certStore.getSignedCertificateFromAfaria("Your_CN", "CA_challenge_code");

  */

  hwc.CertificateStore.prototype.getSignedCertificateFromAfaria = function(commonName, challengeCode) {

    var response = "";

    hwc.traceEnteringMethod("hwc.CertificateStore.getSignedCertificateFromAfaria");

    try {
      if (hwc.isiOS()) {
        response = hwc.getDataFromContainer("certificatestore", "&command=getSignedCertificateFromAfaria" +
        "&commonname=" + encodeURIComponent(commonName) + "&challengecode=" + encodeURIComponent(challengeCode));
      } else if (hwc.isAndroid() || hwc.isBlackBerry()) {
        response = _HWC.getSignedCertificateFromAfaria(commonName, challengeCode);
      } else {
        throw "Not supported on this platform";
      }

    return eval('(' + response + ')');
378    } finally {
379
hwc.traceLeavingMethod("hwc.CertificateStore.getSignedCertificateFromAfaria");
380
}
381
};
382
})(hwc, window);
383
384 })(hwc, window);
385

**ExternalResource.js**

1     /*
2      * Sybase Hybrid App version 2.3.4
3      *
4      * ExternalResource.js
5      *
6      * This file will not be regenerated, so it is possible to modify it, but it
7      * is not recommended.
8      *
9      * Copyright (c) 2012 Sybase Inc. All rights reserved.
10     *
11     */
12    /**
13    * The namespace for the Hybrid Web Container javascript
14    * @namespace
15    */
16    hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc;    // SUP 'namespace'
17
18    (function() {
19    /**
20    */
* Makes an external cross domain request.

* @public

* @memberOf hwc

* @param {String} url The url to make request to

* @param {anonymous.options} options a set of key/value pairs that configure the underlying request.

* @example

var options = {
  method: "GET",
  data: "data",
  async: true,
  headers: {
    "Content-Type": "text/plain;charset=UTF-8"
  },
  complete: function(response) {
    // invoked when the request completes
    if (response.status === 200)
      alert("Update successful");
    else
      alert("Update Failed");
  }
};

global.getExternalResource(url, options);

hwc.getExternalResource = function(url, options) {"}
try {
    // Default options
    _options = {
        method: "GET",
        async: true
    //headers: {},
    //data: '',
    //complete: function() {}
    
    };  

    // Fill in options
    options = options || {};  

    for (key in options) {
        _options[key] = options[key];
    
    }

    options = _options;
    options.method = options.method.toUpperCase();

    if (typeof (options.data) === 'string') {
        params.push(options.data);
    }

    else if (Object.prototype.toString.call(options.data) === '[object Array]') {
        params = options.data;
    }

    else {
        for (key in options.data) {

params.push(encodeURIComponent(key) + "=" + encodeURIComponent(options.data[key]));

if (queryString) {
    if (options.method === "GET") {
        url = url + (url.indexOf("?") === -1 ? '?' : '&') + queryString;
        options.data = "";
    }
else {
    options.data = queryString;
}
}

if (hwc.isBlackBerry()) {
    request = hwc.getXMLHTTPRequest();
    request.open(options.method, url, options.async);
    if (options.headers) {
        for (key in options.headers) {
            request.setRequestHeader(key, options.headers[key]);
        }
    }
    request.onreadystatechange = function() {
        if (request.readyState === 4) {
            // Handle response
        }
    }
handleResponse(options, request);
}
;

request.send(options.data);
}

else if (hwc.isAndroid()){

if (options.async) {

   // Setup callbacks

   callbackSet = new hwc.CallbackSet();

   options.callback =
   callbackSet.registerCallback("callback", function(response)
   { handleResponse(options, response); });

}

// Create a json string for options

jsonOptions = JSON.stringify(options);

jsonText = _HWC.makeExternalRequest(url,
jsonOptions) + "";

if (!options.async && jsonText) {

   handleResponse(options,
JSON.parse(jsonText));

}

else if (hwc.isWindowsMobile() || hwc.isWindows())
{

   // Create a json string for options

   jsonOptions = JSON.stringify(options);

   try {

   //make xmlhttp request to load the rmi response from server

}
xmlhttp = hwc/XMLHTTPRequest();

//container always sends the request as
synchronized, javascript sends the request based on
//caller's choice
xmlhttp.open("POST", "/sup.amp?
querytype=externalresource&" + hwc.versionURLParam, options.async);

xmlhttp.onreadystatechange = function()
{
    if (xmlhttp.readyState == 4) {
        // Success
        if (xmlhttp.status == 200) {
            handleResponse(options,
            JSON.parse(xmlhttp.responseText));
        }
    }
}

xmlhttp.send("url=" +
encodeURIComponent(url) + "&options=" +
encodeURIComponent(jsonOptions));

catch (ex) {
    alert(ex);
}

else if (hwc.isIOS()) {
    // Create a json string for options
    jsonOptions = JSON.stringify(options);
    try {
        // make xmlhttp request to load the rmi
        response from server
        xmlhttp = hwc/XMLHTTPRequest();
    }
//container always sends the request as synced, javascript sends the request based on caller's choice
xmlhttp.open("GET", "http://localhost/sup.amp?querytype=externalresource&" + hwc.versionURLParam + "&url=" + encodeURIComponent(url) + "&options=" + encodeURIComponent(jsonOptions), options.async);

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState === 4) {
        // Success
        handleResponse(options, JSON.parse(xmlhttp.responseText));
    }
};

xmlhttp.send("");

catch (err) {
    alert(err);
}

} finally {
    hwc.traceLeavingMethod("hwc.getExternalResource");
}

/**
 * Internal method to wrap response in a fake xhr
 * @private
 * @param {anonymous.options} options The options provided for the request
function handleResponse(options, response) {
    hwc.traceEnteringMethod("handleResponse");
    try {
        var fakeXHR = {
            "status": response.status,
            "statusText": response.statusText,
            "responseText": response.responseText,
            "getResponseHeader": function(key) {
                var headerValue, header;
                hwc.traceEnteringMethod("fakeXHR.getResponseHeader");
                try {
                    if (response.getResponseHeader) {
                        headerValue = response.getResponseHeader(key);
                    }
                    else if (response.headers) {
                        for (header in response.headers) {
                            if (key.toLowerCase() === header.toLowerCase()) {
                                headerValue = response.headers[header];
                                break;
                            }
                        }
                    }
                } catch (error) {
                    console.log(error);
                }
            }
        }
    }
}
return headerValue === undefined ? null : headerValue;

} finally {

hwc.traceLeavingMethod("fakeXHR.getResponseHeader");

}

"getAllResponseHeaders": function() {

var allHeaders, key;

hwc.traceEnteringMethod("fakeXHR.getAllResponseHeaders");

try {

if (response.getAllResponseHeaders)
{
    return
    response.getAllResponseHeaders();
}

if (response.headers) {

    for (key in response.headers) {

        if (allHeaders) {

            allHeaders += "\r\n";
        }

    }

    allHeaders += (key + ":" + response.headers[key]);

}

return allHeaders;

}

return null;

} finally {

hwc.traceLeavingMethod("fakeXHR.getAllResponseHeaders");

}
if (options.complete) {
  options.complete(fakeXHR);
}
} finally {
  hwc.traceLeavingMethod("handleResponse");
}

/**
 * Used to group anonymous objects and callback functions used as method
 * parameters only for purposes of API docs generation only.
 * Methods and fields in this namespace cannot be instantiated.
 *
 * @namespace
 */

anonymous = (typeof anonymous === "undefined" || !anonymous) ? {} : anonymous;  // SUP 'namespace'

/**
 * Options object used with the {@link getExternalResource} function.
 *
 * Supported options are:
 * <ul>
 *   <li> method: one of GET, PUT, DELETE, HEAD, OPTIONS, or POST. The default is GET.</li>
 *   <li> HTTP and HTTPS urls are supported. </li>
 *   <li> async: request should be sent asynchronously. The default is true. </li>
 */
headers: request headers to be sent with request.

* data: data to be sent. If this is an array, it is converted to a query string. For a GET request, this is added to the end of the URL.

* {@link anonymous.complete} is a callback function that will be invoked with the resultXHR when this method completes

* @name anonymous.options

*/

Callback function used in the {@link Options} object.

* @name anonymous.complete

* @param {object} resultXHR the response object.

The fields/methods available on resultXHR are

* status

* statusText

* responseText

* getResponseHeader(key)

* getAllResponsesHeaders()

These fields and methods are not supported for resultXHR:

* open()
Develop Hybrid Apps Using Third-party Web Frameworks

**hwc-api.js**

1 /*
2 * Sybase Hybrid App version 2.3.4
3 *
4 * hwc-api.js
5 * This file will not be regenerated, so it is possible to modify it, but it
6 * is not recommended.
7 *
8 * Copyright (c) 2011,2012 Sybase Inc. All rights reserved.
9 */
10 /**
11 * Holds all the Hybrid Web Container javascript
12 * @namespace
13 */
14 hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc; // SUP 'namespace'
15
16
17 /**
18 * Container API
19 */
20 (function(hwc, undefined) {
21
22 /**
23 * Constant definitions for registration methods
24 */
25 /**
26 * Constant indicating no registration method preference. The application implementation decides the default method to use.
27 * This is handled as Manual registration by the HWC.
28 * Used in {@link hwc.ConnectionSettings}.}
* @type number */

hwc.REGISTRATION_METHOD_NO_PREFERENCE = 0;

/**
 * Constant indicating that automatic registration using password is the preferred method. Used in {@link hwc.ConnectionSettings}.
 */

* @type number */

hwc.REGISTRATION_METHOD_AUTOMATIC = 1;

/**
 * Constant indicating that manual registration is the preferred method. Used in {@link hwc.ConnectionSettings}.
 */

* @type number */

hwc.REGISTRATION_METHOD_MANUAL = 2;

/**
 * Constant indicating that automatic registration using a certificate from Afaria is the preferred method. Used in {@link hwc.ConnectionSettings}.
 */

* @type number */

hwc.REGISTRATION_METHOD_AFARIA = 3;

/**
 * Constant indicating that automatic registration using a local certificate is the preferred method. Used in {@link hwc.ConnectionSettings}.
 */

* @type number */

hwc.REGISTRATION_METHOD_CERTIFICATE = 4;

/**
 * Represents the connection settings for connecting to the SUP Server. Used in {@link hwc.loadSettings} and {@link hwc.saveSettings}.
 */

* @classdesc

* @memberOf hwc

* @public

* @param {number} regmethod A number representing the registration method (must be one of {@link
hwc.REGISTRATION_METHOD_NO_PREFERENCE}, {@link hwc.REGISTRATION_METHOD_MANUAL},

55          * {@link hwc.REGISTRATION_METHOD_AUTOMATIC}, {@link hwc.REGISTRATION_METHOD_AFARIA}, {@link hwc.REGISTRATION_METHOD_CERTIFICATE}).

56          * @param {string} server The SUP/Relay server name.
57          * @param {number} port The SUP/Relay server port number.
58          * @param {string} server The farm id.
59          * @param {string} user The user name.
60          * @param {string} activationcode The activation code.
61          * @param {string} protocol The protocol to use. Must be
62          * "HTTP" or "HTTPS".
63          * @param {string} password The password for automatic
64          * registration.
65          * @param {string} urlsuffix The url suffix (used only when
66          * connecting to a relay server).

67          * @example
68          * // Create a new ConnectionSettings object.
69          * var connectionSettings = new
hwc.ConnectionSettings( hwc.REGISTRATION_METHOD_MANUAL,

70          * "999.999.999.999",
71          * 5001,
72          * 0,
73          * "sampleUsername",
74          * 123,
75          * "HTTP",
76          * "samplePassword",
77          * "/" );
78          * // Use the ConnectionSettings object we just created to
set the connection settings.
79          * hwc.saveSettings( connectionSettings );
80          * */
hwc.ConnectionSettings = function (regmethod, server, port, farm, user, activatecode, protocol, password, urlsuffix)
{
    this.RegistrationMethod = regmethod;
    this.ServerName = server;
    this.Port = port;
    this.FarmID = farm;
    this.UserName = user;
    this.ActivationCode = activatecode;
    this.Protocol = protocol;
    this.Password = password;
    this.UrlSuffix = urlsuffix;
}

/**
 * Loads the current connection settings from the native application storage.
 * @memberOf hwc
 * @public
 * @returns {hwc.ConnectionSettings} The connection settings or null if there are no cached settings.
 * @example
 * // Load the connection settings.
 * var connectionSettings = hwc.loadSettings();
 */

hwc.loadSettings = function () {
    var settings, response, jsonobj;
    settings = null;

    hwc.traceEnteringMethod("hwc.loadSettings");

    try {
        response = hwc.getDataFromContainer("loadsettings");
        jsonobj = JSON.parse(response);
    
if (jsonobj !== null && jsonobj !== undefined) {
    settings = new hwc.ConnectionSettings(jsonobj.enableautoregistration, 
        jsonobj.servername, 
        jsonobj.port, jsonobj.farmid, jsonobj.username, 
        jsonobj.activationcode, jsonobj.protocol, jsonobj.password, 
        jsonobj.urlsuffix);
}

} catch (ex) {
    hwc.log("loadSettings error:" + ex.message, "ERROR", false);
}

} finally {
    hwc.traceLeavingMethod("hwc.loadSettings");
}

return settings;

/**
 * Constant definitions for device management in add device registration.
 * Some other error numbers may apply for technical support.
 */

/**
 * Constant indicating that MMS Authentication failed. Possible return value for {@link hwc.saveSettings}.
 * @type number *
 */

hwc.REG_ERR_MMS_AUTHENTICATION_FAILED = 14814;

/**
* Constant indicating that the connection to the MMS service failed. Possible return value for `hwc.saveSettings`.

```java
hwc.REG_ERR_COULD_NOT_REACH_MMS_SERVER = 14813;
```

* Constant indicating that no MBS template was found for given AppId and/or Security configuration. Possible return value for `hwc.saveSettings`.

```java
hwc.REG_ERR_AUTO_REG_TEMPLATE_NOT_FOUND = 14850;
```

* Constant indicating that auto registration was not enabled in the template. Possible return value for `hwc.saveSettings`.

```java
hwc.REG_ERR_AUTO_REG_NOT_ENABLED = 14851;
```

* Constant indicating that the given device id is already registered for another user. Possible return value for `hwc.saveSettings`.

```java
hwc.REG_ERR_AUTO_REG_WRONG_USER_FOR_DEVICE = 14853;
```

* Constant indicating that the user name is longer than the legal limit. Possible return value for `hwc.saveSettings`.

```java
hwc.REG_ERR_AUTO_REG_USER_NAME_TOO_LONG = 14854;
```

* Constant indicating that the user name contains invalid characters. Possible return value for `hwc.saveSettings`.

```java
hwc.REG_ERR_AUTO_REG_USER_NAME_TOO_LONG = 14854;
```
hwc.REG_ERR_INVALID_USER_NAME = 14856;

/**
 * Constant indicating {@link hwc.saveSettings} completed successfully. Possible return value for {@link hwc.saveSettings}.
 * @type number */

hwc.SETTING_SUCCESS = 0;

/**
 * Save the connection settings to native application storage.
 * Device registration will be attempted if and only the following conditions are both satisfied.
 * <ol>
 * <li>The registration method is not manual. This can be passed in the hwc.ConnectionSettings object, or if that value is null, the currently configured value will be used. </li>
 * <li>The password must be non-empty. This value MUST be passed in the hwc.ConnectionSettings object. </li>
 * </ol>
 * <p>hwc.startClient() needs to be called after hwc.saveSettings() for the device to complete automatic/manual registration. </p>
 * <p>Usage Note: It is not mandatory to specify a value for each {@link hwc.ConnectionSettings} property. Specifying a null or undefined for a {@link hwc.ConnectionSettings} property will effectively cause this method to IGNORE the property and not change it's value. </p>
 * <p>If the saveSettings() operation fails, a non-zero number will be returned. See hwc.REG_ERR_* for device registration errors. There can be other types of errors not listed here. </p>
 * @public
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
180         * @memberOf hwc
181         * @param {hwc.ConnectionSettings} settings The connection
settings to be saved.
182         *
183         * @returns {number} A status code indicating success
((@link hwc.SETTING_SUCCESS)) or an error (one of (@link
hwc.REG_ERR_AUTO_REG_NOT_ENABLED),
184         * {@link hwc.REG_ERR_AUTO_REG_TEMPLATE_NOT_FOUND}, {@link
hwc.REG_ERR_AUTO_REG_USER_NAME_TOO_LONG}, {@link
hwc.REG_ERR_AUTO_REG_WRONG_USER_FOR_DEVICE},
185         * {@link hwc.REG_ERR_COULD_NOT_REACH_MMS_SERVER}, {@link
hwc.REG_ERR_INVALID_USER_NAME}, {@link
hwc.REG_ERR_MMS_AUTHENTICATION_FAILED})).
186         * @example
187         * // Load the connection settings.
188         * var connectionSettings = hwc.loadSettings();
189         * // Modify the connection settings.
190         * connectionSettings.ServerName = "999.999.999.999";
191         * // Save the modified connection settings.
192         * hwc.saveSettings( connectionSettings );
193         * // Start the client to for the device to complete
automatic/manual registration.
194         * hwc.startClient();
195         */
196        hwc.saveSettings = function (settings) {
197           hwc.traceEnteringMethod("hwc.saveSettings");
198           try {
199              // First compose the URL argument string
200              var argumentString, ret;
201              argumentString = "";
202              ret = "";
203
204              if (settings.RegistrationMethod !== null &&
settings.RegistrationMethod !== undefined)
205              {
```
argumentString = "&enableautoregistration=" + settings.RegistrationMethod;

if (settings.ServerName !== null && settings.ServerName !== undefined) {
    argumentString = argumentString + "&servername=" + encodeURIComponent(settings.ServerName);
}

if (settings.Port !== null && settings.Port !== undefined) {
    argumentString = argumentString + "&port=" + settings.Port;
}

if (settings.FarmID !== null && settings.FarmID !== undefined) {
    argumentString = argumentString + "&farmid=" + encodeURIComponent(settings.FarmID);
}

if (settings.UserName !== null && settings.UserName !== undefined) {
    argumentString = argumentString + "&username=" + encodeURIComponent(settings.UserName);
}

if (settings.ActivationCode !== null && settings.ActivationCode !== undefined) {
    argumentString = argumentString + "&activationcode=" + encodeURIComponent(settings.ActivationCode);
}

    argumentString = argumentString + "&protocol=" + encodeURIComponent(settings.Protocol);
}
function saveSettings(settings) {

    var argumentString = ""

        argumentString = argumentString + "&protocol=" +
        encodeURIComponent(settings.Protocol);
    }

    if (settings.Password !== null && settings.Password !== undefined) {
        argumentString = argumentString + "&password=" +
        encodeURIComponent(settings.Password);
    }

    if (settings.UrlSuffix !== null && settings.UrlSuffix !== undefined) {
        argumentString = argumentString + "&urlsuffix=" +
        encodeURIComponent(settings.UrlSuffix);
    }

    // Only invoke the native function if we're saving at
    // least one setting
    if (argumentString !== ")
    {
        ret = hwc.getDataFromContainer("savesettings", argumentString);
        return parseInt(ret, 10);
    }

    else {
        return hwc.SETTING_SUCCESS;
    }

    } catch (ex) {
        hwc.log("saveSettings error:" + ex.message, "ERROR", false);
        throw ex;
    }

    } finally {
        hwc.traceLeavingMethod("hwc.saveSettings");
    }
}
257        
258        
259        /**
260        * Start of connection state listener callback functions
261        */
262        /**
263        */
264        * An array of {@link anonymous.ConnectionStateListener} callback functions.
265        * @type Array
266        * @private
267        */
268        hwc._connectionListeners = [];
269        /**
270        */
271        * An array of objects containing {@link anonymous.ConnectionStateListener} callback functions.
272        * The containing objects need to be kept track of since the callback functions may reference
273        * variables in the containing object.
274        * @type Array
275        * @private
276        */
277        hwc._connectionListenerContainingObjects = [];
278        /**
279        */
280        * This is the main entry of connection event notification. The native code
281        * calls this function internally
282        * @param {number} event A flag indicating the current connection state (will be either {@link hwc.CONNECTED} or {@link hwc.DISCONNECTED}).
283        * @param {number} errorCode An error code. Will be 0 if there is no error.
@param {string} errorMessage Text of an error message. Will be the empty string if there is no error.

```
function (event, errorCode, errorMessage)
```
/**
 * Register the connection state listener.
 *
 * @public
 * @memberOf hwc
 * @param {anonymous.ConnectionStateListener} ConnectionStateListener Callback for connection state changes.
 * @param {Object} [containingObject] Object containing definition for ConnectionStateListener. If a connection state callback function references variables in its containing object, then the containing object should be passed to this function.
 *
 * @example
 * // doSomething is a global function that gets called from the connection listener.
 * var doSomething = function()
 * {
 *    alert("sample function that gets executed when the hwc becomes connected");
 * }
 * // connectionListener is the callback function that is given to addConnectionListener.
 * var connectionListener = function( event, errorCode, errorMessage )
 * {
 *    if( event == hwc.CONNECTED )
 *    {
 *       doSomething();
 *    }
 * }
 * hwc.addConnectionListener( connectionListener );
 */
* @example

* // connectionStateManager is an object that will contain the connection listener callback as well as
* // a variable used by the callback.
* var connectionStateManager = {};
* // The connectionStateManager keeps track of whether the HWC is connected or not.
* connectionStateManager.connected = false;
* // A function called by the listener.
* connectionStateManager.doSomething = function()
* {
*    if( this.connected )
*    {
*       alert("this alert gets displayed if the hwc is connected");
*    }
*    
*    this.doSomething();
* }
* // This is the callback function that will be passed to addConnectionListener. This callback references variables
* // from the containing object (this.connected and this.doSomething), so when we call addConnectionListener we have
* // to give the containing object as the second parameter.
* connectionStateManager.listener = function( event, errorCode, errorMessage )
* {
*    if( event == hwc.CONNECTED )
*    {
*       this.connected = true;
*    }
*    else
*    {
*       this.connected = false;
*    }
*    this.doSomething();
hwc.addConnectionListener = function (ConnectionStateListener, containingObject) {
    hwc.traceEnteringMethod("hwc.addConnectionListener");
    try {
        hwc._connectionListeners.push(ConnectionStateListener);
        hwc._connectionListenerContainingObjects.push(containingObject);
        if (hwc._connectionListeners.length === 1)
        {
            hwc.getDataFromContainer("startconnectionlistener");
        }
    } finally {
        hwc.traceLeavingMethod("hwc.addConnectionListener");
    }
};

/**
 * Remove the connection state listener. This function should be called with identical parameters
 * that were used when adding the connection state listener with {@link hwc.addConnectionListener}.
 */

* @public
* @memberof hwc
* @param {anonymous.ConnectionStateListener} ConnectionStateListener Callback function with connection state changes

* @param {Object} [containingObject] Optional Object containing definition of ConnectionStateListener

* @example

* // doSomething is a global function that gets called from the connection listener.

* var doSomething = function()

* {

*   alert("sample function that gets executed when the hwc becomes connected");

* }

* // connectionListener is the callback function that is given to addConnectionListener.

* // When there is a connection event, connectionListener will be invoked with the details.

* var connectionListener = function( event, errorCode, errorMessage )

* {

*   if( event == hwc.CONNECTED )

*     { doSomething();

*   }

* }

* hwc.addConnectionListener( connectionListener );

* // At some other point if we want to remove the listener, we use the following line:

* hwc.removeConnectionListener( connectionListener );

* 

* @example

* // connectionStateManager is an object that will contain the connection listener callback as well as

* // a variable used by the callback.

* var connectionStateManager = {};

* // The connectionStateManager keeps track of whether the HWC is connected or not.
* connectionStateManager.connected = false;

* // A function called by the listener.
* connectionStateManager.doSomething = function()
* {
*   if( this.connected )
*   {
*      alert("this alert gets displayed if the hwc is connected");
*   }
* }

* // This is the callback function that will be passed to addConnectionListener. This callback references variables
* // from the containing object (this.connected and this.doSomething), so when we call addConnectionListener we have
* // to give the containing object as the second parameter.
* connectionStateManager.listener = function( event, errorCode, errorMessage )
* {
*   if( event == hwc.CONNECTED )
*   {
*      this.connected = true;
*   }
*   else
*   {
*      this.connected = false;
*   }
*   this.doSomething();
* }

* // Pass both the listener and the containing object. This enables the listener to refer to variables in the containing
* object when it is invoked.
* hwc.addConnectionListener( connectionStateManager.listener, connectionStateManager );
* // At some other point if we want to remove the listener, we use the following line:

```javascript
hwc.removeConnectionListener( connectionStateManager.listener, connectionStateManager );
```

```javascript
hwc.removeConnectionListener = function (ConnectionStateListener, containingObject)
{
    var i;

    hwc.traceEnteringMethod("hwc.removeConnectionListener");

    try {
        if (hwc._connectionListeners.length === 0) {
            return;
        }

        for (i = 0; i < hwc._connectionListeners.length; i++) {
            if (hwc._connectionListeners[i] === ConnectionStateListener &&
                hwc._connectionListenerContainingObjects[i] === containingObject)
                hwc._connectionListeners[i].splice(i, 1);
        }

        if (hwc._connectionListeners.length === 0)
        {
            hwc.getDataFromContainer("stopconnectionlistener");
        }

        return;
    }
```

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps
        } finally {

            hwc.traceLeavingMethod("hwc.removeConnectionListener");

        }

    

    /**
     * A sample {link anonymous.ConnectionStateListener} callback function.
     *
     * @param {number} event A number indicating the event that occurred (will be {link hwc.CONNECTED} or {link hwc.DISCONNECTED}).
     *
     * @param {number} errorCode An error code (0 indicating success).
     *
     * @param {string} errorMessage Text of the error message. Will be empty of there is no error.
     *
     */

    hwc.sample_ConnectionListener = function (event, errorCode, errorMessage) {

        switch (event) {

            case hwc.CONNECTED:
                alert('Connected event');
                break;

            case hwc.DISCONNECTED:
                alert('Disconnected event');
                break;

        }

        if (errorCode !== null && errorMessage !== null) {
            alert('Connection error
' +
      'Code: ' + errorCode + '
' +
      'Message: ' + errorMessage);

        }
hwc.CONNECTED = 1;

/**
 * Constant indicating that the hwc is disconnected. Used in
 * {@link anonymous.ConnectionStateListener} callback functions.
 * @type number
 */

hwc.DISCONNECTED = 2;

/**
 * Start the client connection to the SUP server.
 * Companion function to {@link hwc.shutdown}.
 * If a hybrid app is running in the context of the Hybrid
 * Web Container then it will probably never have to call this function
 * unless {@link hwc.shutdown} client was called first.
 * @public
 * @memberOf hwc
 * @param {anonymous.LogListener} [onNotification] A log
 * listener callback function. If you are interested in
 * the connection state it is recommended that you call
 * {@link hwc.addConnectionListener} before calling hwc.startClient.
 * @example
 * hwc.startClient();
 * @example
 */
* // Add a log listener while calling hwc.startClient.
* var logListener = function( time, event, message )
* {
*    alert(message);
* }
* hwc.startClient( logListener );
*
* hwc.startClient = function (onNotification) {
      hwc.traceEnteringMethod("hwc.startClient");
      try {
        if (hwc._defaultLogListener !== null &&
            hwc._defaultLogListener !== undefined)
        {
          hwc.removeLogListener(hwc._defaultLogListener, null);
          hwc._defaultLogListener = null;
        }
        if (onNotification !== null && onNotification !== undefined)
        {
          hwc.addLogListener( onNotification, null );
          hwc._defaultLogListener = onNotification;
        }
      }
      finally {
        hwc.traceLeavingMethod("hwc.startClient");
      }
    hwc.getDataFromContainer( "startclient" );
    return 0;
    } catch (ex){
      hwc.log("startClient error:" + ex.message, "ERROR", false);
    } finally {
      hwc.traceLeavingMethod("hwc.startClient");
    }
```javascript
hwc.shutdown = function () {
    hwc.traceEnteringMethod("hwc.shutdown");
    try {
        hwc.getDataFromContainer("shutdownclient");
        if (hwc._defaultLogListener !== null &&
            hwc._defaultLogListener !== undefined)
        {
            hwc.removeLogListener(hwc._defaultLogListener, null);
            hwc._defaultLogListener = null;
        }
    } catch (ex) {
        hwc.log("shutdown error:" + ex.message, "ERROR", false);
    } finally {
        hwc.traceLeavingMethod("hwc.shutdown");
    }
};
```

/**
* Shutdown the client connection to the SUP server. Companion function to {@link hwc.startClient}.
* If a hybrid app is running in the context of the Hybrid Web Container, then it will probably never have to call
* this function. If you want to temporarily stop the connection, then call {@link hwc.disconnectFromServer} instead.
* @public
* @memberOf hwc
* @example
* hwc.shutdown();
*/
* Resumes the connection to the SUP server. Companion function to {@link hwc.disconnectFromServer}. This function should only be called after the connection to the SUP server has been suspended with a call to {@link hwc.disconnectFromServer}.

* @public

* @memberOf hwc

* @param {anonymous.LogListener} [onNotification] A log listener callback function. If you are interested in the connection state it is recommended that you call {@link hwc.addConnectionListener} before calling hwc.connectToServer.

* @example

    * hwc.connectToServer();

* @example

    * // Add a log listener while calling hwc.connectToServer.
    * var logListener = function( time, event, message )
    * {
    *     alert(message);
    * }
    * hwc.connectToServer( logListener );

hwc.connectToServer = function (onNotification) {
    hwc.traceEnteringMethod("hwc.connectToServer");
    try {
        if (hwc._defaultLogListener !== null && hwc._defaultLogListener !== undefined)
        {
            hwc.removeLogListener(hwc._defaultLogListener, null);
            hwc._defaultLogListener = null;
        }
    }
if (onNotification !== null && onNotification !== undefined) {
    hwc.addLogListener(onNotification, null);
    hwc._defaultLogListener = onNotification;
}

hwc.getDataFromContainer("connecttoserver");

return 0;
} catch (ex) {
    hwc.log("connectToServer error:" + ex.message, "ERROR", false);
    throw ex;
} finally {
    hwc.traceLeavingMethod("hwc.connectToServer");
}

/**
 * This is the default one to keep the listener added in the connectionToServer call.
 * @private
 */

hwc._defaultLogListener = null;

/**
 * Suspends the connection to the SUP server. Companion function to {@link hwc.connectToServer}.
 * @public
 * @memberOf hwc
 * @example
 * hwc.disconnectFromServer();
 */
hwc.disconnectFromServer = function () {
  hwc.traceEnteringMethod("hwc.disconnectFromServer");
  try {
    hwc.getDataFromContainer("disconnectfromserver");

    if (hwc._defaultLogListener !== null && hwc._defaultLogListener !== undefined) {
      hwc.removeLogListener(hwc._defaultLogListener, null);
      hwc._defaultLogListener = null;
    }
  } catch (ex) {
    hwc.log("disconnectFromServer error:" + ex.message, "ERROR", false);
  } finally {
    hwc.traceLeavingMethod("hwc.disconnectFromServer");
  }
};

/**
 * Start of connection functions
 */

/**
 * An array of log listener callback functions.
 */

hwc._logListeners = [];
An array of objects containing log listener callback functions. The containing objects need to be kept track of because the callback functions may reference variables in the containing object.

@type Array

hwc._logListenerContainingObjects = [];

/**
 * This is the main entry of log event notification. The native code calls this function internally.

 * @param {number} milliseconds The date of the log message represented in milliseconds.
 * @param {number} event The that represents which category this event falls under (It will be one of hwc.CONNECTION_* constants).
 * @param {string} optionalString The string carrying the message of the log event.

 * @private

hwc._logListenerNotification = function (milliseconds, event, optionalString)
{
  var date, i, containingObject;

  hwc.traceEnteringMethod("hwc._logListenerNotification");

  try {
    if (hwc._logListeners.length === 0) {
      return;
    }
  }
}
// The incoming date is number of millisecond, we need to change it to real JavaScript Date type.

date = new Date(milliseconds);

for (i = 0; i < hwc._logListeners.length; i++)
{
    containingObject = hwc._logListenerContainingObjects[i];

    if (containingObject !== null && containingObject !== undefined)
    {
        hwc._logListeners[i].call(containingObject, date, event, optionalString);
    }
    else
    {
        hwc._logListeners[i](date, event, optionalString);
    }
} finally {
    hwc.traceLeavingMethod("hwc._logListenerNotification");
}

/**
 * Register the log listener.
 * @public
 * @memberOf hwc
 * @param {anonymous.LogListener} LogListener Callback for changes to the log.
 * @param {Object} [containingObject] Object containing definition for LogListener. If a log listener callback function references variables in its containing object, then the containing object should be passed to this function.
* A global function called by the log listener.

```javascript
var doSomething = function()
{
    alert("this gets displays when there is a log event.");
}
```

* The log listener callback function that will be passed to hwc.addLogListener.

```javascript
var logListener = function( event, errorCode, errorMessage )
{
    doSomething();
}
```

* This function will be invoked whenever there is a log event.

```javascript
var logListenerManager = {};
```

* as a function that will be invoked from the listener callback function.

```javascript
var logListenerManager = {};
```

* This is a function that is called from the listener callback.

```javascript
logListenerManager.doSomething = function()
{
    alert("this gets displays when there is a log event.");
}
```

* This is the listener callback that will be passed to hwc.addLogListener.
* // Since a variable is referenced from the containing object, the containing object
* // will need to be passed to hwc.addLogListener.

* logListenerManager.listener = function( event, errorCode, errorMessage )
* {
*    this.doSomething();
* }
* // Pass both the listener callback and the containing object.

* hwc.addLogListener( logListenerManager.listener, logListenerManager );
*/

hwc.addLogListener = function ( LogListener, containingObject)
{
    hwc.traceEnteringMethod("hwc.addLogListener");
    try {
        hwc._logListeners.push(LogListener);
        hwc._logListenerContainingObjects.push(containingObject);
        if (hwc._logListeners.length === 1)
        {
            hwc.getDataFromContainer("startloglistener");
        }
    } finally {
        hwc.traceLeavingMethod("hwc.addLogListener");
    }
}

/**
* Remove the log listener. This function should be called with identical parameters that were used
* when adding the log listener with [@link hwc.addLogListener].

@public
@memberOf hwc
@param {anonymous.LogListener} LogListener The callback function for log events.
@param {Object} [containingObject] Object containing definition of ConnectionStateListener
@example
// A global function called by the log listener.
var doSomething = function()
{
    alert("this gets displays when there is a log event.");
}

// The log listener callback function that will be passed to hwc.addLogListener.
// This function will be invoked whenever there is a log event.
var logListener = function( event, errorCode, errorMessage )
{
doS omething();
}

// Add the log listener.
hwc.addLogListener( logListener );

// at some other point if we want to remove the listener, we use the following line
hwc.removeLogListener( logListener );

/* @example
/* logListenerManager is an object that will contain the listener callback as well
/* as a function that will be invoked from the listener callback function.
var logListenerManager = {};
*/
Develop Hybrid Apps Using Third-party Web Frameworks

801* // This is a function that is called from the listener callback.
802* logListenerManager.doSomething = function()
803* {
804*   alert("this gets displays when there is a log event.");
805* }
806* // This is the listener callback that will be passed to hwc.addLogListener.
807* // Since a variable is referenced from the containing object, the containing object
808* // will need to be passed to hwc.addLogListener.
809* logListenerManager.listener = function(event, errorCode, errorMessage)
810* {
811*   this.doSomething();
812* }
813* // Pass both the listener callback and the containing object.
814* hwc.addLogListener(logListenerManager.listener, logListenerManager);
815* // at some other point if we want to remove the listener, we use the following line
816* hwc.removeLogListener(logListenerManager.listener, logListenerManager);
817*/
818*  
819  
820  
821  
822  
823  
824  
825  
826
for (i = 0; i < hwc._logListeners.length; i++)
{
    if (hwc._logListeners[i] === LogListener &&
        hwc._logListenerContainingObjects[i] === containingObject)
    {
        hwc._logListeners.splice(i, 1);
        hwc._logListenerContainingObjects.splice(i, 1);

        if (hwc._logListeners.length === 0)
        {
            hwc.getDataFromContainer("stoploglistener");
            return;
        }
    }
} finally {
    hwc.traceLeavingMethod("hwc.removeLogListener");
}

/**
 * Sample {@link anonymous.LogListener} callback function.
 *
 * @param {number} milliseconds The date of the log message represented in milliseconds.
 *
 * @param {number} event The that represents which category this event falls under (It will be one of {@link hwc.CONNECTION_ERROR},
 * @param {number} event The that represents which category this event falls under (It will be one of {@link hwc.CONNECTION_OTHER}, {@link hwc.CONNECTION_CONNECTED}, {@link hwc.CONNECTION_DISCONNECTED}, {@link hwc.CONNECTION_RETRIEVED_ITEMS}).
* @param {string} optionalString The string carrying the message of the log event.

```
hwc.sample_LogListener = function ( date, event, optionalString ) {
};
```

// Connection event definitions

```
/**
 * A constant indicating that the log message is about a connection error. Used in `{@link anonymous.LogListener}` callback functions.
 *
 * @type number
 */

hwc.CONNECTION_ERROR = -1;
```

```
/**
 * A constant indicating that the log message is not about the connection. Used in `{@link anonymous.LogListener}` callback functions.
 *
 * @type number
 */

hwc.CONNECTION_OTHER = 0;
```

```
/**
 * A constant indicating that the log message is about the connection being established. Used in `{@link anonymous.LogListener}` callback functions.
 *
 * @type number
 */

hwc.CONNECTION_CONNECTED = 1;
```

```
/**
 * A constant indicating that the log message is about the connection being disconnected. Used in `{@link anonymous.LogListener}` callback functions.
 *
 * @type number
 */

hwc.CONNECTION_DISCONNECTED = 2;
```
/**
 * a constant indicating that the log message is about retrieved items. Used in
 * @link anonymous.LogListener} callback functions.
 */

hwc.CONNECTION_RETRIEVED_ITEMS = 3;

/**
 * Start of hybrid app installation callback functions
 */

/**
 * An array of app installation listeners
 * @private
 * @type Array
 */

hwc._appInstallationListeners = [];

/**
 * This is the main entry of installation notification. The native code should be
 * hardcoded to call this function internally.
 * @private
 */

* @param {number} event A constant indicating whether the app installation is beginning or has just ended
  * (will be either @link hwc.INSTALLATION_BEGIN} or @link hwc.INSTALLATION_END}.
  * @param {number} moduleId The module ID of the hybrid app being installed.
  * @param {number} version The version of the hybrid app being installed.
* @param {string} moduleName The display name of the hybrid app being installed.

hwc.appInstallationListenerNotification = function (event, moduleId, version, moduleName)
{
    var i;

    hwc.traceEnteringMethod("hwc.appInstallationListenerNotification");
    try {
        if (hwc._appInstallationListeners.length === 0) {
            return;
        }
    }
    finally {
        hwc.traceLeavingMethod("hwc.appInstallationListenerNotification");
    }

    for (i = 0; i < hwc._appInstallationListeners.length; i++)
    {
        hwc._appInstallationListeners[i](event, moduleId, version, moduleName);
    }
}

/**
 * Register the application installation listener.
 *
 * @param {anonymous.AppInstallationListener} AppInstallationListener A callback for application installation changes.
 * @example
 * // appInstallListener is the callback function that will be passed to hwc.addAppInstallationListener.
```javascript
* var appInstallListener = function( event, moduleId, version, moduleName )
* {
*    if( event == hwc.INSTALLATION_BEGIN )
*    {
*       alert(moduleName + " has just started the installation process.");
*    }
*    else if( event == hwc.INSTALLATION_END )
*    {
*       alert(moduleName + " has just finished the installation process.");
*    }
* }
* hwc.addAppInstallationListener( appInstallListener );

hwc.addAppInstallationListener = function(AppInstallationListener)
{
  try {
    hwc._appInstallationListeners.push(AppInstallationListener);
    if(hwc._appInstallationListeners.length === 1)
    {
      hwc.getDataFromContainer("startAppInstallationListener");
    }
  } finally {
    hwc.traceLeavingMethod("hwc.addAppInstallationListener");
  }
};
```
/** Remove the application installation listener. This function should be called with identical parameters that were used to add the application installation listener with \@link hwc.addAppInstallationListener. */

@public

@memberOf hwc

@param {anonymous.AppInstallationListener} AppInstallationListener The callback for application installation changes.

@example

// appInstallListener is the callback function that will be passed to hwc.addAppInstallationListener.

var appInstallListener = function( event, moduleId, version, moduleName )

    if( event == hwc.INSTALLATION_BEGIN )
    {
        alert(moduleName + " has just started the installation process.");
    }

    else if( event == hwc.INSTALLATION_END )
    {
        alert(moduleName + " has just finished the installation process.");
    }

hwc.addAppInstallationListener( appInstallListener );

// when we want to remove this listener, we call the following line:

hwc.removeAppInstallationListener( appInstallListener );

*/

hwc.removeAppInstallationListener = function

{...}
var i;

hwc.traceEnteringMethod("hwc.removeAppInstallationListener");

try {
    if (hwc._appInstallationListeners.length === 0) {
        return;
    }

    for (i = 0; i < hwc._appInstallationListeners.length; i++) {
        if (hwc._appInstallationListeners[i] === AppInstallationListener) {
            hwc._appInstallationListeners.splice(i, 1);
            break;
        }
    }

    if (hwc._appInstallationListeners.length === 0) {
        hwc.getDataFromContainer("stopAppInstallationListener");
    }
}

} finally {
    hwc.traceLeavingMethod("hwc.removeAppInstallationListener");
}

/**
 * Sample application listener callback function
Develop Hybrid Apps Using Third-party Web Frameworks

```
1014        * @param {Integer} event            Installation flags
           including, BEGIN(1), END(2), FAIL(3)
1015        * @param {String} moduleId          Optional Module Id
1016        * @param {String} version          Optional Module
1017           * @param {String} moduleName        Optional Module display
1018           * @param {String} designerVersion   Optional Version of
1019           * @param {String} containerVersion  Optional Version of
1020        */
1021       hwc.sample_InstallationAppListener = function (event, moduleId, version, moduleName, designerVersion, containerVersion) {
1022       };
1023
1024       // Installation event definitions
1025       /**
1026        * A constant indicating that the application is starting
1027        * to be installed. Used in {@link anonymous.AppInstallationListener}
1028        * callback functions.
1029        * @type number
1030        */
1031       hwc.INSTALLATION_BEGIN = 1;
1032
1033       /**
1034        * A constant indicating that the application has finished
1035        * being installed. Used in {@link anonymous.AppInstallationListener}
1036        * callback functions.
1037        * @type number
1038        */
1039       hwc.INSTALLATION_END = 2;
1040       hwc.INSTALLATION_FAIL = 3;
1041
1042       /**
1043        * Call this function to get an array of {@link hwc.LogEntry} objects. There will be one
```
* {@link hwc.LogEntry} object for each line in the HWC log.

* @public

* @memberOf hwc

* @returns {hwc.LogEntry[]} An array of hwc.LogEntry objects.

* @example

var log = hwc.getLogEntries();

*/

hwc.getLogEntries = function () {
  var response, logEntries, i, entries, entry;

  hwc.traceEnteringMethod("hwc.getLogEntries");

  response = "";

  logEntries = [];

  try {
      response = hwc.getDataFromContainer("getLogEntries");

      if (response !== null && response !== undefined && response !== "")
      {
          entries = JSON.parse(response);

          for (i=0; i<entries.length; i++) {
              entry = entries[i];
              logEntries[i] = new hwc.LogEntry(new Date(entry.milliseconds), entry.event, entry.message);
          }
      }

      } catch (ex){
        hwc.log("getLogEntries error:" + ex.message, "ERROR", false);
1067          } finally {
1068              hwc.traceLeavingMethod("hwc.getLogEntries");
1069          }
1070
1071          return logEntries;
1072       }
1073
1074       /**
1075        * This object represents a log entry.
1076        * @classdesc
1077        * @public
1078        * @memberOf hwc
1079        * @param {number} date The date the log entry was recorded, in milliseconds since January 1, 1970, 00:00:00 GMT
1080        * @param {number} event The event ID of the log entry (will be one of {@link hwc.CONNECTION_ERROR}, {@link hwc.CONNECTION_CONNECTED}, {@link hwc.CONNECTION_DISCONNECTED}, {@link hwc.CONNECTION_RETRIEVED_ITEMS})
1081        * @param {string} msg The message of the log entry.
1082        */
1083    hwc.LogEntry = function (date, event, msg)
1084    {
1085        this.logdate = date;
1086        this.eventID = event;
1087        this.message = msg;
1088
1089       /**
1090        * Gets the date of the log entry.
1091        * @public
1092        * @memberOf hwc.LogEntry
1093        * @returns {number} The date the log entry was created in the HWC, in milliseconds.
this.getDate = function () {
    return this.logdate;
};

/**
 * Gets the event ID of the log entry to see what this log entry is about.
 * @public
 * @memberOf hwc.LogEntry
 * @returns {number} A constant indication what this log entry is about (will be one of
 *                  {link hwc.CONNECTION_ERROR}, {link hwc.CONNECTION_OTHER},
 *                  {link hwc.CONNECTION_CONNECTED}, {link hwc.CONNECTION_DISCONNECTED},
 *                  {link hwc.CONNECTION_RETRIEVED_ITEMS}).
 */
this.getEventID = function () {
    return this.eventID;
};

/**
 * Gets the message text of the log entry.
 * @public
 * @memberOf hwc.LogEntry
 * @returns {string} The message text of the log entry.
 */
this.getMessage = function () {
    return this.message;
};
hwc._pushnotificationlisteners = []; 

hwc._pushnotificationlistenerContainingObjects = []; 

hwc.traceEnteringMethod("hwc._pushnotificationListenerNotification"); 

try {
    var ret, i, notifications, containingObject;
    hwc.traceEnteringMethod("hwc._pushnotificationListenerNotification");
    try {
        }
ret = hwc.NOTIFICATION_CONTINUE;

try
{
if (hwc._pushnotificationlisteners.length > 0)
{
notifications = JSON.parse(jsonString);

// We must have a valid push data to continue
if (!(notifications === null || notifications === undefined || notifications.length === 0))
{
for (i = 0; i < hwc._pushnotificationlisteners.length; i++)
{
try
{
containingObject = hwc._pushnotificationlistenerContainingObjects[i];

if (containingObject !== null && containingObject !== undefined)
{
ret = hwc._pushnotificationlisteners[i].call(containingObject, notifications);
}
else
{
ret = hwc._pushnotificationlisteners[i](notifications);
}
}
else
{
ret = hwc._pushnotificationlisteners[i](notifications);
}
}
// If the return status is hwc.NOTIFICATION_CANCEL, we need to return immediately.
if (ret === hwc.NOTIFICATION_CANCEL) {
    break;
}

try {
    for (var id in _list) {
        if (hwc.isBlackBerry() || hwc.isIOS())
            return ret;
        else
            hwc.getDataFromContainer("jsmethodreturn", 
"&id=" + id + "&jsreturnvalue=" + ret);
    }
}

finally {
    hwc.traceLeavingMethod("hwc._pushnotificationListenerNotification");
}
 Register a push notification listener.

* @public

@memberOf hwc

@param {function} PushNotificationListener The callback for push notifications.

* @param {Object} [containingObject] Object containing definition for PushNotificationListener. If the listener callback function references variables in its containing object, then the containing object should be passed to this function.

@example

// pushListener is the callback function that will be passed to hwc.addPushNotificationListener.

var pushListener = function( notifications )
{
    alert( "push notification:\n" + JSON.stringify(notifications) );

    return hwc.NOTIFICATION_CONTINUE;

};

hwc.addPushNotificationListener( pushListener );

@example

// pushListenerManager is an object that will contain the listener callback as well as a variable referenced from the callback.

var pushListenerManager = {};

// doSomething is a function that is called from inside the callback.

pushListenerManager.doSomething = function( notifications )
{

```
1230    *    alert("push notification:\n" +
1231    *    JSON.stringify(notifications) );
1232    *    return hwc.NOTIFICATION_CONTINUE;
1233    *    } // This is the callback function.
1234    *    pushListenerManager.listener =
1235    *    function( notifications )
1236    *    {
1237    *        return this.doSomething( notifications );
1238    *    }
1239    *    // Since the callback function references variables in
1240    *    // its containing object, the containing object
1241    *    // must be passed to hwc.addPushNotificationListener as
1242    *    // well.
1243    *    pushListenerManager =
1244    *    hwc.addPushNotificationListener( pushListenerManager.listener,
1245    *    pushListenerManager );
1246    * /
1247    *    hwc.addPushNotificationListener =
1248    *    function(PushNotificationListener, containingObject)
1249    {
1250    hwc.traceEnteringMethod("hwc.addPushNotificationListener");
1251    try {
1252    hwc._pushnotificationlisteners.push(PushNotificationListener);
1253    hwc._pushnotificationlistenerContainingObjects.push(containingObject);
1254    // The native side will start to notify the
1255    // notification when the first
1256    // listener is added
1257    if (hwc._pushnotificationlisteners.length ===
1258    1) {
1259    hwc.getDataFromContainer("startpushnotificationlistener");
1260    }
1261    }
```
```javascript
} finally {
  hwc.traceLeavingMethod("hwc.addPushNotificationListener");
}

/**
 * Remove the push notification listener. This function should be called with identical parameters that were used to add the push notification listener with {link hwc.addPushNotificationListener}.
 *
 * @public
 * @memberOf hwc
 * @param {anonymous.PushNotificationListener} PushNotificationListener The callback for push notifications.
 * @param {Object} [containingObject] The containing object of the listener.
 * @example
 * // pushListener is the callback function that will be passed to hwc.addPushNotificationListener.
 * var pushListener = function( notifications ) {
 *    alert( "push notification:\n" +
 *        JSON.stringify(notifications) );
 *    return hwc.NOTIFICATION_CONTINUE;
 * }
 * hwc.addPushNotificationListener( pushListener );
 * // At some other point if we want to remove the push listener, we call the following line:
 * hwc.removePushNotificationListener( pushListener );
 * @example
 * // pushListenerManager is an object that will contain the listener callback as well as a variable
 * // referenced from the callback.
 * hwc.addPushNotificationListener( pushListenerManager.pushListener );
 * hwc.removeListener('pushListenerManager', function() {
 *    // do something
 * }, null);
 */
```
var pushListenerManager = {};

// doSomething is a function that is called from inside the callback.
pushListenerManager.doSomething = function(notifications) {
    alert("push notification:\n" + JSON.stringify(notifications));
    return hwc.NOTIFICATION_CONTINUE;
}

// This is the callback function.
pushListenerManager.listener = function(notifications) {
    return this.doSomething(notifications);
}

// Since the callback function references variables in its containing object, the containing object must be passed to hwc.addPushNotificationListener as well.
hwc.addPushNotificationListener(pushListenerManager.listener, pushListenerManager);

// when we want to remove the push listener, we call the following line:
hwc.removePushNotificationListener(pushListenerManager);

hwc.removePushNotificationListener = function(PushNotificationListener, containingObject) {
    var i;
    hwc.traceEnteringMethod("hwc.removePushNotificationListener");
    try {
        if (hwc._pushnotificationlisteners.length === 0) {
            
        } else { 
            
        } 
    } catch (e) { 
        
    }
};
for (i = 0; i < hwc._pushnotificationlisteners.length; i++)
{
    if (hwc._pushnotificationlisteners[i] === PushNotificationListener &&
        hwc._pushnotificationlistenerContainingObjects[i] === containingObject)
    {
        hwc._pushnotificationlisteners.splice(i, 1);
        hwc._pushnotificationlistenerContainingObjects.splice(i, 1);
        if (hwc._pushnotificationlisteners.length === 0)
        {
            hwc.getDataFromContainer("stoppushnotificationlistener");
        } else
        {
            return;
        }
    }
} finally {
    hwc.traceLeavingMethod("hwc.removePushNotificationListener");
}

*/
* A constant indicating that other push notification
listeners should continue to be called.
* Used as a return value for @link
anonymous.PushNotificationListener) functions.
* @type number
hwc.NOTIFICATION_CONTINUE = 0;
/**
 * A constant indicating that no more push notification listeners should be called.
 * Used as a return value for {@link anonymous.PushNotificationListener} functions.
 * @type number
 */

hwc.NOTIFICATION_CANCEL = 1;
/**
 * A sample implementation of a {@link anonymous.PushNotificationListener} callback function.
 * @param {Array} notifications Array of notifications.
 */

hwc.sample_PushNotificationListener = function(notifications)
{
    return hwc.NOTIFICATION_CONTINUE;
};

/**
 * This object represents a hybrid app.
 * @classdesc
 * @public
 * @memberOf hwc
 * @param {number} moduleId The module id of this hybrid app.
 * @param {number} version The version of this hybrid app.
 * @param {string} displayName The display name of this hybrid app.

* @param {number} iconIndex The index specifying the icon representing this Hybrid App.

* @param {hwc.CustomIcon} defaultCustomIcon The default custom icon for this hybrid app.

* @param {hwc.CustomIcon[]} customIconList An array of custom icon objects.

*/

hwc.HybridApp = function (moduleId, version, displayName, iconIndex, defaultCustomIcon, customIconList)

{
    this.ModuleID = moduleId;
    this.Version = version;
    this.DisplayName = displayName;
    this.IconIndex = iconIndex;
    this.defIcon = defaultCustomIcon;
    this.IconList = customIconList;


    /**
     * Gets the module ID for this hybrid app.
     * @public
     * @memberOf hwc.HybridApp
     * @returns {number} The module ID.
     */
    this.getModuleID = function ()
     {
        return this.ModuleID;
    };
}

/**
 * Gets the version number for this hybrid app.
 * @public
 * @memberOf hwc.HybridApp
 * @returns {number} The version.
 */

this.getVersion = function (){
    return this.Version;
};

/**
* Gets the display name for this hybrid app.
* @public
* @memberOf hwc.HybridApp
* @returns {string} The display name.
*/
this.getDisplayName = function (){
    return this.DisplayName;
};

/**
* Gets the icon index used in the list of built-in icons.
* @public
* @memberOf hwc.HybridApp
* @returns {number} The icon index
*/
this.getIconIndex = function (){
    return this.IconIndex;
};

/**
* Gets the default custom icon object of this hybrid app.
* @public
* @memberOf hwc.HybridApp
*/
@returns {hwc.CustomIcon} The default custom icon of this hybrid app.  Null if this hybrid app does not have a custom icon.

* /

this.getDefaultCustomIcon = function ()
{
    return this.defIcon;
};

/**
 * Gets the list of custom icons associated with this hybrid app.
 *
 * @public
 * @memberOf hwc.HybridApp
 * @returns {hwc CustomIcon[]} The array of custom icon objects.  Null if this hybrid app has no custom icons.
 */

this.getCustomIconList = function ()
{
    return this.IconList;
};

/**
 * Return a {@link hwc.ClientVariables} object for the given module id and version.
 *
 * @public
 * @memberOf hwc.HybridApp
 * @returns {hwc.ClientVariables} The {@link hwc.ClientVariables} object for this hybrid app.
 */

this.getClientVariables = function()
{
    return hwc.getClientVariables( this.ModuleID, this.Version );
};
An array of `anonymous.ApplicationListener` callback functions.

```javascript
/**
 * An array of `{@link anonymous.ApplicationListener}` callback functions.
 * @private
 * @type {anonymous.ApplicationListener[]} */

hwc._applicationListeners = [];

/**
 * An array of objects containing `{@link anonymous.ApplicationListener} callback functions.
 * The containing objects need to be kept track of in the case that a callback function references
 * a variable from its containing object.
 * @private
 * @type {Array}
 */

hwc._applicationListenerContainingObjects = [];

/**
 * This is the main entry of application notification. The native code should be
 * hardcoded to call this function internally.
 * @private
 */

hwc._applicationListenerNotification = function (event, moduleId, version)
{
  var i, containingObject;
  hwc.traceEnteringMethod("hwc._applicationListenerNotification");

  try {
```
if (hwc._applicationListeners.length === 0) {
    return;
}

for (i = 0; i < hwc._applicationListeners.length; i++) {
    containingObject = hwc._applicationListenerContainingObjects[i];
    if (containingObject !== null && containingObject !== undefined)
        hwc._applicationListeners[i].call(containingObject, event, moduleId, version);
    else
        hwc._applicationListeners[i](event, moduleId, version);
}

} finally {
    hwc.traceLeavingMethod("hwc._applicationListenerNotification");
}

/**
 * Register the application listener.
 *
 * @param {anonymous.ApplicationListener} 
 * ApplicationListener The callback function for application changes.
 *
 * @param {Object} [containingObject] The containing object of the listener method. This parameter is only
* required if the ApplicationListener references the containing object.
* @public
* @memberOf hwc
* @example
* // This is the callback function that will be passed to hwc.addAppListener.
* var appListener = function( event, moduleId, version )
* {
*   if( event == hwc.APP_ADDED )
*     alert("A hybrid app has been added.");
* }
* hwc.addAppListener( appListener );
* @example
* // appListenerManager is an object that will contain the callback function as well as variables
* // the callback function references.
* var appListenerManager = {};
* // doSomething is a function that is called from inside the callback function.
* appListenerManager.doSomething = function( event )
* {
*   if( event == hwc.APP_REMOVED )
*     alert("A hybrid app has been removed.");
* }
* // This is the callback function that will be passed to hwc.addAppListener. It calls doSomething,
* // the definition of which is in the containing function.
appListenerManager.listener = function( event, moduleId, version )
* {
*    this.doSomething( event );
* }
* // Since the listener callback function references a variable from its containing object,
* // the containing object must be passed to hwc.addAppListener.
* hwc.addAppListener( appListenerManager.listener, appListenerManager );
*
* hwc.addAppListener = function (ApplicationListener, containingObject)
* {
*    hwc.traceEnteringMethod("hwc.addAppListener");
*    try {
*        hwc._applicationListeners.push(ApplicationListener);
*        hwc._applicationListenerContainingObjects.push(containingObject);
*        // The native side will start to notify the notification when the first
*        // listener is added
*        if (hwc._applicationListeners.length === 1)
*            hwc.getDataFromContainer("startapplistener");
*    }
*    finally {
*        hwc.traceLeavingMethod("hwc.addAppListener");
*    }
* } finally {
*    hwc.traceLeavingMethod("hwc.addAppListener");
*}
* /**
* Remove the application listener. This function should be called with identical parameters
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
* that were used to add the application listener with
* `{@link hwc.addAppListener}`.

* @public

* @memberOf hwc

* @param {anonymous.ApplicationListener} ApplicationListener The callback for application changes.

* @param {Object} [containingObject] The containing object of the application listener function.

* @example

* // This is the callback function that will be passed to
* hwc.addAppListener.

* var appListener = function( event, moduleId, version )

* {

*    if( event == hwc.APP_ADDED )

*    {

*       alert("A hybrid app has been added.");

*    }

* }

* hwc.addAppListener( appListener );

* // At some other point, if we want to remove the listener
* we use the following line of code:

* hwc.removeAppListener( appListener );

* * @example

* // appListenerManager is an object that will contain the
* callback function as well as variables

* // the callback function references.

* var appListenerManager = {};

* // doSomething is a function that is called from inside
* the callback function.

* appListenerManager.doSomething = function( event )

* {

*    if( event == hwc.APP_REMOVED )
```
alert("A hybrid app has been removed.");

// This is the callback function that will be passed to hwc.addAppListener. It calls doSomething,
// the definition of which is in the containing function.

appListenerManager.listener = function( event, moduleId, version )
{
    this.doSomething( event );
}

// Since the listener callback function references a variable from its containing object,
// the containing object must be passed to hwc.addAppListener.

hwc.addAppListener( appListenerManager.listener, appListenerManager );

// At some other point, if we want to remove the listener we use the following line of code:

hwc.removeAppListener( appListenerManager.listener, appListenerManager );

hwc.removeAppListener = function (ApplicationListener, containingObject)
{
    var i;
    hwc.traceEnteringMethod("hwc.removeAppListener");
    try {
        if (hwc._applicationListeners.length === 0) {
            return;
        }
    }
    for (i = 0; i < hwc._applicationListeners.length; i++)
if (hwc._applicationListeners[i] === ApplicationListener &&
    hwc._applicationListenerContainingObjects[i] === containingObject) {
    hwc._applicationListeners.splice(i, 1);
    hwc._applicationListenerContainingObjects.splice(i, 1);
    if (hwc._applicationListeners.length === 0) {
        hwc.getDataFromContainer("stopapplistener");
        return;
    }
    hwc.getDATAFromContainer("stopapplistener");
}

/**
 * A constant indicating that the application list requires a refresh.
 * Used in {@link anonymous.ApplicationListener} callback functions as a possible value for event.
 * @type number
 */
hwc.APP_REFRESH = 1;

/**
 * A constant indicating that a hybrid app has been added.
 * Used in {@link anonymous.ApplicationListener} callback functions as a possible value for event.
 */
hwc.APP_ADDED = 2;
/**
 * A constant indicating that a hybrid app was updated.
 * Used in `{@link anonymous.ApplicationListener} callback
 * functions as a possible value for event.
 * @type number
 */

hwc.APP_UPDATED = 3;
/**
 * A constant indicating that a hybrid app was removed.
 * Used in `{@link anonymous.ApplicationListener} callback
 * functions as a possible value for event.
 * @type number
 */

hwc.APP_REMOVED = 4;
/**
 * A sample `{@link anonymous.ApplicationListener} callback
 * function.
 */

* @param {number} event A number indicating what event has
 * taken place (will be one of `{@link hwc.APP_REFRESH},
 * @{link hwc.APP_ADDED}, @{link hwc.APP_UPDATED}, @{link
 * hwc.APP_REMOVED}).
 * @param {number} moduleId The module id of the hybrid
 * app the event is about.
 * @param {number} version module The version of the hybrid
 * app the event is about.
 */

hwc.sample_AppListener = function (event, moduleId,
    version) {
    
};
/**
 * Gets the hybrid app that is currently open.
 *
 * @public
 * @memberOf hwc
 * @returns {hwc.HybridApp} The hybrid app that is currently open.
 * @example
 * var openHybridApp = hwc.getCurrentApp();
 */

hwc.getCurrentApp = function()
{
    var response, currentApp, app;

    hwc.traceEnteringMethod("hwc.getCurrentApp");
    response = "";

    try {
        response = hwc.getDataFromContainer("getcurrentapp");
        if (response !== ")
        {
            app = JSON.parse(response);

            currentApp = new hwc.HybridApp(app.moduleId, app.version, app.displayName, app.iconIndex,

            hwc.createCustomIconObject(app.defaultCustomIcon, app.moduleId, app.version, hwc.DEFAULT_CUSTOM_ICON_INDEX),

            hwc.createCustomIconList(app.customIconList, app.moduleId, app.version));

        }
    } catch (ex){

    }
hwc.log("getCurrentApp error:" + ex.message, "ERROR", false);

} finally {
  hwc.traceLeavingMethod("hwc.getCurrentApp");
}

return currentApp;

/**
 * Returns an array of {@link hwc.HybridApp} objects.
 * @public
 * @memberOf hwc
 * @param {boolean} [completeList] If this parameter is set to true, then all apps that are user invocable or require activation will be returned. If set to false or if it is not set, then if there is a default hybrid app only the default hybrid app will be returned (and if there is no default hybrid app it will return all hybrid apps that are user invocable or require activation).
 * @returns {hwc.HybridApp[]} An array of hybrid app objects.
 * @example
 * var apps = hwc.getInstalledApps();
 * @example
 * var apps = hwc.getInstalledApps( true );
 * /
 * hwc.getInstalledApps = function( completeList )
 * {
 *  var formattedCompleteList, response, installedApps, app, apps, i;
 * 

hwc.traceEnteringMethod("hwc.getInstalledApps");
formattedCompleteList = false;
response = "";
installedApps = [];

if( completeList )
{
    formattedCompleteList = true;
}

try {
    response = hwc.getDataFromContainer("getinstalledapps", 
    
    if (response !== null && response !== undefined &&
        response !== "")
    {
        apps = JSON.parse(response);
        for (i=0; i<apps.length; i++) {
            app = apps[i];
            installedApps[i] = new hwc.HybridApp(app.moduleId, app.version, app.displayName, app.iconIndex,
            hwc.createCustomIconObject(app.defaultCustomIcon, app.moduleId, app.version, hwc.DEFAULT_CUSTOM_ICON_INDEX),
            hwc.createCustomIconList(app.customIconList, app.moduleId, app.version));
        }
    }
    catch (ex){
    }
hwc.log("getInstalledApps error:" + ex.message, "ERROR", false);

} finally {
    hwc.traceLeavingMethod("hwc.getInstalledApps");
}

return installedApps;

/**
 * Returns an array of {@link hwc.HybridApp} objects that are server initiated.
 *
 * @public
 * @memberOf hwc
 * @returns {hwc.HybridApp[]} An array of server initiated hybrid apps.
 * @example
 * var serverInitiatedApps = hwc.getServerInitiatedApps();
 */

hwc.getServerInitiatedApps = function()
{
    var response = "", serverInitiatedApps = [], app, apps, i;

    hwc.traceEnteringMethod("hwc.getServerInitiatedApps");
    try {
        response = hwc.getDataFromContainer("getserverinitiatedapps");

        if (response !== null && response !== undefined &&
            response !== "")
            {
                var response = "", serverInitiatedApps = [], app, apps, i;

                hwc.traceEnteringMethod("hwc.getServerInitiatedApps");
                try {
                    response = hwc.getDataFromContainer("getserverinitiatedapps");

                    if (response !== null && response !== undefined &&
                        response !== "")
                    {

apps = JSON.parse(response);

for (i=0; i<apps.length; i++) {
    app = apps[i];
    serverInitiatedApps[i] = new hwc.HybridApp(app.moduleId, app.version, app.displayName, app.iconIndex, hwc.createCustomIconObject(app.defaultCustomIcon, app.moduleId, app.version, hwc.DEFAULT_CUSTOM_ICON_INDEX), hwc.createCustomIconList(app.customIconList, app.moduleId, app.version));
}

} catch (ex){
    hwc.log("getServerInitiatedApps error:" + ex.message, "ERROR", false);
}

} finally {
    hwc.traceLeavingMethod("hwc.getServerInitiatedApps");
}

return serverInitiatedApps;

/**
 * Gets a {link hwc.HybridApp} object with the given module id and version.
 *
 * @public
 * @memberOf hwc
 * @param {number} moduleId The module ID of the hybrid app.
 * @param {number} version The version of the hybrid app.
 */
Develop Hybrid Apps Using Third-party Web Frameworks
1798
* @returns {hwc.HybridApp} The hybrid app object, or null
if there is no hybrid app with the given ID and version.
1799

*

1800

* @example

1801
* // Messages do not have a direct link to the hybrid app
they belong to. Instead they have
1802
belong to.

* // the module ID and version of the hybrid app they
If you have a message and

1803
* // need to access its hybrid app, first you must call
hwc.getAppByID.
1804

* var messages = hwc.getAllMessages();

1805

* if( messages.length > 0 )

1806

* {

1807
*
var app = hwc.getAppByID( messages[0].getModuleId(),
messages[0].getModuleVersion() );
1808

* }

1809

*/

1810

hwc.getAppByID = function (moduleID, version)

1811

{

1812

var response, appInstance, app, params;

1813
1814

hwc.traceEnteringMethod("hwc.getAppByID");

1815

response = "";

1816
version;

params = "&moduleid=" + moduleID + "&moduleversion=" +

1817
1818
1819
params);

try {
response = hwc.getDataFromContainer("getappbyid",

1820
1821

if (response !== "")

1822

{

1823

app = JSON.parse(response);

1824
appInstance = new hwc.HybridApp(app.moduleId,
app.version, app.displayName, app.iconIndex,

Developer Guide: Hybrid Apps

361


hwc.createCustomIconObject(app.defaultCustomIcon, app.moduleId, app.version, hwc.DEFAULT_CUSTOM_ICON_INDEX),

hwc.createCustomIconList(app.customIconList, app.moduleId, app.version));

} catch (ex){
    hwc.log("getAppByID error:" + ex.message, "ERROR", false);
}

} finally {
    hwc.traceLeavingMethod("hwc.getAppByID");
}

return appInstance;

/**
 * A constant indicating that {@link hwc.openApp} completed successfully.
 * This is a possible return value for {@link hwc.openApp}.
 * @type number
 */

hwc.OPEN_APP_SUCCESS = 0;

/**
 * A constant indicating that {@link hwc.openApp} failed because the specified app does not exist.
 * This is a possible return value for {@link hwc.openApp}.
 * @type number
 */

hwc.OPEN_APP_NOT_EXIST = 1;

/**
 * A constant indicating that {@link hwc.openApp} failed for an unspecified reason.
 */
This is a possible return value for `{@link hwc.openApp}`.

@type number

`/`

`hwc.OPEN_APP_OTHER = 2;`

/**
 * Launch the hybrid app with the given module ID and version. The hybrid app will be opened on top of the hybrid app that is open when `hwc.openApp` is called. When the hybrid app that was opened with `hwc.openApp` exits, it will exit to the hybrid app that was open when `hwc.openApp` was called. It is possible to nest open hybrid apps, but it is best not to have too many nested hybrid apps (eg: recursively opening hybrid apps) because each open hybrid app takes up device memory.

* @param {number} moduleId Module id of the hybrid app.

* @param {number} version Version of the hybrid app.

* @returns {number} A constant indicating the result of opening the hybrid app (will be one of `{@link hwc.OPEN_APP_SUCCESS}`, `{@link hwc.OPEN_APP_NOT_EXIST}`, `{@link hwc.OPEN_APP_OTHER}`).

* @public

* @memberOf hwc

* @example

var apps = hwc.getInstalledApps();

if( apps.length > 0 )

{ // Check to make sure the first app is not this app (the app that is currently running),

    // since we don't want to recursively open this app until memory runs out.
if( hwc.getCurrentHybridApp.getDisplayName() !== apps[0].getDisplayName() )
{
    hwc.openApp( apps[0].getModuleID(), apps[0].getVersion() );
}

hwc.openApp = function (moduleId, version)
{
    var response;
    hwc.traceEnteringMethod("hwc.openApp");
    try {
        response = hwc.getDataFromContainer("openhybridapp",
            "&moduleid=" + moduleId + "&moduleversion=" + version);
        return parseInt(response, 10);
    } catch (ex){
        hwc.log("app.open error:" + ex.message, "ERROR", false);
    } finally {
        hwc.traceLeavingMethod("hwc.openApp");
    }
}

/**
 * A constant indicating the custom icon index.
 */
@type number
*/
hwc.DEFAULT_CUSTOM_ICON_INDEX = -1;

/**
 * Gets the Hybrid Web Container application connection ID.
 */
1906       * @public
1907       * @memberOf hwc
1908       * @returns {string} Application connection ID
1909       * @example
1910       * var appConnectionID =
1911       hwc.getApplicationConnectionID();
1912   */
1913   hwc.getApplicationConnectionID = function() {
1914     var response = "";
1915   hwc.traceEnteringMethod("hwc.getApplicationConnectionID");
1916     try
1917     {
1918       response =
1919       hwc.getDataFromContainer("getconnectionid");
1920     }
1921     catch (ex) {
1922       hwc.log("get connection id error:" + ex.message,
1923       "ERROR", false);
1924     } finally {
1925   hwc.traceLeavingMethod("hwc.getApplicationConnectionID");
1926     return String(response);
1927   }
1928   /**
1929   * Gets the client variables of the hybrid app with given
1930   * module id and version.
1931   *
1932   * @public
1933   * @memberOf hwc

* @param {number} moduleID The module ID of the hybrid app.

* @param {number} version The version of the hybrid app.

* @returns {hwc.ClientVariables} A {@link hwc.ClientVariables} object, or null if there are no ClientVariables for the hybrid app with the given module id and version.

* @example

var apps = hwc.getInstalledApps();

// Loop through the apps, showing the client variables for each one.

for( var i = 0; i < apps.length; i++ )
{
  var app = apps[i];

  // Get the client variables.
  var clientVariables =
  hwc.getClientVariables( app.getModuleID(), app.getVersion() );

  if( clientVariables.getCount() > 0 )
  {
    // Get all the names of the variables for this app.
    var keys =
    clientVariables.getAllVariableNames();

    // Loop through all the variable for this app.
    for( var index = 0; index < keys.length; index++ )
    {
      // Get a specific variable by name.
      var variable =
      clientVariables.getVariableValueByName( keys[index] );

      alert( "variable name: " + keys[index] + 
                "\nvariable value: " + variable );
    }
  }
}

hwc.getClientVariables = function (moduleID, version) {
    var response, clientVariables, parsedResponse, params;

    hwc.traceEnteringMethod("hwc.getClientVariables");
    response = "";
    clientVariables = null;

    params = ";moduleid=" + moduleID + ";moduleversion=" + version;

    try {
        response = hwc.getDataFromContainer("getclientvariables", params);
    }
    catch (ex) {
        hwc.log("getClientVariables error:" + ex.message, "ERROR", false);
    }
    finally {
        hwc.traceLeavingMethod("hwc.getClientVariables");
    }

    if (response !== "") {
        parsedResponse = JSON.parse( response );
        clientVariables = new hwc.ClientVariables( parsedResponse.version, parsedResponse.items );
    }

    return clientVariables;
1988       );
1989
1990       /**
1991       * Represents a ClientVariables object.
1992       *
1993       * @classdesc
1994       *
1995       * @public
1996       * @param {number} clientVariablesVersion  The version of
1997       * @param {Object} clientVariableItems    The json object that contains key/value pairs of client variable items.
1998       * @memberOf hwc
1999       */
2000       hwc.ClientVariables = function ( clientVariablesVersion, clientVariableItems )
2001       {
2002          this.version = clientVariablesVersion;
2003          this.items = clientVariableItems;
2004
2005          /**
2006          * Gets the version of the client variables.
2007          * @returns {number} The version of the client variables.
2008          * @public
2009          * @memberOf hwc.ClientVariables
2010          */
2011          this.getVersion = function ()
2012          {
2013             return this.version;
2014          };
2015
2016       /**
* Gets the number of variables this `{@link hwc.ClientVariables}` contains.
* @public
* @memberOf hwc.ClientVariables
* @returns {number} The number of variables.
 */
this.getCount = function ()
{
    var keys = this.getAllVariableNames();

    return keys.length;
};

/**
 * Gets an array containing the names of all variables in this `{@link hwc.ClientVariables}`.
 *
 * @public
 * @memberOf hwc.ClientVariables
 * @returns {string[]} The array holding the names of all variables contained in this `{@link hwc.ClientVariables}`.
 */
this.getAllVariableNames = function ()
{
    var result, prop;

    hwc.traceEnteringMethod("hwc.ClientVariables.getAllVariableNames");
    try {
        result = [];

        if ( this.items !== undefined && this.items !== null )
        {
            for ( prop in this.items )
                result.push(prop);

            return result;
        }
    }
    catch ( e )
    {
        throw new Error(e);
    }
};
if ( this.items.hasOwnProperty( prop ) && typeof this.items[ prop ] === 'string' ) {
    result.push( prop );
}

result.sort();
return result;
} finally {
    hwc.traceLeavingMethod("hwc.ClientVariables.getAllVariableNames");
}

/**
 * Check if this {@link hwc.ClientVariables} has a variable by the given name.
 *
 * @public
 * @memberOf hwc.ClientVariables
 * @param {string} variableName The name of variable to check for.
 *
 * @returns {boolean} True if this {@link hwc.ClientVariables} has a variable by the given name, false otherwise.
 */
this.containsName = function ( variableName ) {
    if ( this.items === undefined || this.items === null || ( typeof this.items[ variableName ] !== 'string' ) ) {
    
}
return false;
}

return true;
};

/**
 * Gets the value of the variable with the given name. If this
 * @link hwc.ClientVariables} does not have a variable
 * by the given name, a {link hwc.ClientVariablesException} will be thrown.
 *
 * @public
 * @memberOf hwc.ClientVariables
 * @param {string} variableName The name of the variable
 * to get the value of.
 *
 * @returns {string} The value of the variable.
 *
 * @throws {hwc.ClientVariableException} This exception
 * is thrown when there is no variable by the given name in this
 * @link hwc.ClientVariables}.
 */

this.getVariableValueByName = function
( variableName )
{
    if ( !this.containsName( variableName ) )
    {
        throw new
        hwc.ClientVariablesException( hwc.ClientVariables.ITEM_NOT_FOUND,
        "Unable to find variable name: " + variableName );
    }
}

return this.items[ variableName ];

};
This exception is thrown when `@link hwc.ClientVariables#getVariableValueByName` is called with a variable name that does not exist.

* @public
* @memberOf hwc
* @param {number} errCode The error code (will be `@link hwc.ClientVariables.ITEM_NOT_FOUND`).
* @param {string} errMsg A message describing the error.
* @classdesc

hwc.ClientVariablesException = function(errCode, errMsg)
{
    this.errCode = errCode;
    this.errMsg = errMsg;
};

A constant indicating that a variable does not exist in a `{@link hwc.ClientVariables} object.

* @type number
* /

hwc.ClientVariables.ITEM_NOT_FOUND = 1;

* Represents a CustomIcon. Used with the `{@link hwc.HybridApp} object.

* @classdesc
* @public
* @memberof hwc
* @param {number} width The width of this custom icon.
```
2128        * @param {number} height The height of this custom icon.
2129        * @param {string} type The image type of this custom icon.
2130        * @param {string} name The name of this custom icon.
2131        * @param {string} path The file path of the unprocessed icon.
2132        * @param {string} processedPath The file path of the processed icon.
2133        * @param {number} moduleId The module ID of the hybrid app this icon is for.
2134        * @param {number} moduleVersion The module version of the hybrid app this icon is for.
2135        * @param {number} index The index of this custom icon.
2136        */
2137        hwc.CustomIcon = function (width, height, type, name, path, processedPath, moduleId, moduleVersion, index)
2138        {
2139            this.w = width;
2140            this.h = height;
2141            this.t = type;
2142            this.n = name;
2143            this.p = path;
2144            this.pp = processedPath;
2145            this.mi = moduleId;
2146            this.mv = moduleVersion;
2147            this.index = index;
2148
2149            /**
2150            * Gets the width of this custom icon.
2151            * @public
2152            * @memberOf hwc.CustomIcon
2153            * @returns {number} The width of this custom icon.
2154            */
2155            this.getWidth = function ()
```
2156          {
2157             return this.w;
2158          };
2159
2160          /**<
2161           * Gets the height of this custom icon.
2162           * @public
2163           * @memberOf hwc.CustomIcon
2164           * @returns {number} The height of this custom icon.
2165           */
2166          this.getHeight = function ()
2167          {
2168             return this.h;
2169          };
2170
2171          /**<
2172           * Gets the image type of this custom icon.
2173           * @public
2174           * @memberOf hwc.CustomIcon
2175           * @returns {string} The file type of the image.
2176           */
2177          this.getType = function ()
2178          {
2179             return this.t;
2180          };
2181
2182          /**<
2183           * Gets the name of this custom icon.
2184           * @public
2185           * @memberOf hwc.CustomIcon
2186           * @returns {string} The name of this custom icon.
2187           */
this.getName = function () {
    return this.n;
};

/**
 * Gets the file path of the unprocessed icon.
 * @public
 * @memberOf hwc.CustomIcon
 * @returns {string} The file path of the unprocessed icon.
 */
this.getImagePath = function () {
    return this.p;
};

/**
 * Gets the file path of the processed icon.
 * @public
 * @memberOf hwc.CustomIcon
 * @returns {string} The file path of the processed icon.
 */
this.getProcessedImagePath = function () {
    return this.pp;
};

/**
 * Gets the URL of this custom icon. It is possible to call this function directly, but generally
 * it is easier simply to call {@link hwc.getAppIconUrl} or {@link hwc.getMsgIconUrl}. Those
* functions handle both cases where there is and isn't a custom icon for the hybrid app or message.

* @public

* @memberOf hwc

* @param {boolean} processed When set to true, the URL of the processed icon will be returned.

* When set to false, the URL of the unprocessed icon will be returned.

* @returns {string} The URL to the target icon.

* @example

```javascript
var apps = hwc.getInstalledApps();
var app = apps[0];

// If app doesn't have a custom icon, then customIcon will be null.
var customIcon = app.getDefaultCustomIcon();
if( customIcon != null ) {
    // Create the image element.
    var image = document.createElement( "img" );
    // Set the source of the image to the icon URL.
    image.setAttribute( 'src', customIcon.getIconUrl() );
    // Add the image element to the page.
    document.body.appendChild( image );
}
```

```javascript
this.getIconUrl = function (processed)
{
    return hwc.getCustomIconUrl(this.mi, this.mv, this.index, processed);
}
```

```javascript
};
```
This method is called internally.

@private

@param {Object} jsonObj The JSON object containing information about the custom icon.

@param {number} moduleId The module ID of the hybrid app this custom icon belongs to.

@param {number} moduleVersion The module version of the hybrid app this custom icon belongs to.

@param {number} index The index of this custom icon.

@returns {hwc.CustomIcon} The new CustomIcon object.

hwc.createCustomIconObject = function(jsonObj, moduleId, moduleVersion, index)
{
    if (jsonObj === null) {
        return null;
    }

    if (jsonObj === undefined) {
        return undefined;
    }

    return new hwc.CustomIcon(jsonObj.width, jsonObj.height, jsonObj.type, jsonObj.name, jsonObj.path, jsonObj.processedPath, moduleId, moduleVersion, index);
}

/**
 * This method is called internally
 * @private
 * @param {Array} jsonArr An array of JSON objects that contain information about custom icons
* @param {number} moduleId The module ID that will be associated with the custom icons
* @param {number} moduleVersion The module version that will be associated with the custom icons
* @returns {hwc.CustomIcon[]} An array of CustomIcon objects
* /

hwc.createCustomIconList = function (jsonArr, moduleId, moduleVersion)
{
  var iconArray, i, icon;
  iconArray = [];

  if (jsonArr === null) {
    return null;
  }
  
  if (jsonArr === undefined) {
    return undefined;
  }
  
  if (jsonArr.length > 0)
  {
    for (i=0; i<jsonArr.length; i++)
    {
      icon = hwc.createCustomIconObject(jsonArr[i], moduleId, moduleVersion, i);
      if (icon !== null && icon !== undefined){
        iconArray.push(icon);
      }
    }
  }

  return iconArray;
/**
 * Gets the URL to the custom icon. This function is used by `hwc.CustomIcon#getIconUrl`.
 *
 * @public
 * @memberOf hwc
 * @param {number} moduleId The module Id of the hybrid app the custom icon belongs to.
 * @param {number} moduleVersion The version of the hybrid app the custom icon belongs to.
 * @param {number} iconIndex The index of the custom icon.
 * @param {boolean} processed Whether to get the processed icon (true), or the unprocessed icon (false).
 *
 * @returns {string} The URL to the target icon.
 */

hwc.getCustomIconUrl = function (moduleId, moduleVersion, iconIndex, processed)
{
    return getRequestUrl("customicon", "moduleid=" + moduleId+ "&moduleversion=" + moduleVersion + "&iconindex=" + iconIndex + "&processed=" + processed);
};

/**
 * Gets the icon URL for the built-in icon. This function is used by `hwc.getMsgIconUrl` and `hwc.getAppIconUrl`.
 * It is possible to call this function directly, but generally it is easier simply to call `hwc.getMsgIconUrl` or `hwc.getAppIconUrl` instead. Those functions handle both cases where there is and isn't a custom icon for the hybrid app or message.
 */
hwc.getBuiltInIconUrl = function (iconIndex, processed) {
    return getRequestUrl("clienticon", "iconindex=" + iconIndex + "&processed=" + processed);
}

/**
 * This function gets the URL of the icon for a message object depending on its
 * processed status and whether there are custom icons defined.
 */

@public
@memberOf hwc
@param {number} iconIndex The index of the built-in icon.
@param {boolean} processed Whether or not to get the URL of the processed icon (true) or the unprocessed icon (false).

@returns {string} The URL to the icon.
* @example
var messages = hwc.getAllMessages();
if( messages.length > 0 )
{
  // Create the image element.
  var messageIcon = document.createElement("img");
  // Set the source of the image to the icon URL.
  messageIcon.setAttribute( 'src', hwc.getMsgIconUrl( messages[0] ) );
  // Add the image element to the page.
  document.body.appendChild( messageIcon );
}

hwc.getMsgIconUrl = function (msg)
{
  hwc.traceEnteringMethod("hwc.getMsgIconUrl");
  try {
    var app = hwc.getAppByID(msg.getModuleId(), msg.getModuleVersion());
    if (app === null || app === undefined) {
      return hwc.getBuiltInIconUrl(msg.getIconIndex(), msg.isProcessed());
    } else {
      return hwc.getAppIconUrl(app, msg.isProcessed());
    }
  }
  finally {
    hwc.traceLeavingMethod("hwc.getMsgIconUrl");
  }
};
This function gets the URL of the icon for a hybrid app depending on whether custom icons are defined.

* @public
* @memberOf hwc
* @param {hwc.HybridApp} app The hybrid app for which the icon URL is desired.
* @param {boolean} processed Whether to get the URL of the processed icon (true) or the URL of the unprocessed icon (false).

* @returns {string} The URL of the icon.

* @example
  var apps = hwc.getInstalledApps();
  if( apps.length > 0 )
  {
    var hybridApp = apps[0];
    // Create the image element.
    var hybridAppIcon = document.createElement("img");
    // Set the source of the image to the icon URL.
    hybridAppIcon.setAttribute( 'src', hwc.getAppIconUrl( hybridApp, false ) );
    // Add the image element to the page.
    document.body.appendChild( hybridAppIcon );
  }

hwc.getAppIconUrl = function(app, processed)
{
  hwc.traceEnteringMethod("hwc.getAppIconUrl");
  try {
    var ci = app.getDefaultCustomIcon();
    if (ci !== null && ci !== undefined)
    {
      return ci.getIconUrl(processed);
    }
  }
}
else {
    return hwc.getBuiltInIconUrl(app.getIconIndex(), processed);
}
} finally {
    hwc.traceLeavingMethod("hwc.getAppIconUrl");
}

/**
 * Represents a message received by the HWC.
 *
 * @classdesc
 * @public
 * @memberOf hwc
 * @param {number} msgId The message ID of this message.
 * @param {Date} date The date this message was received.
 * @param {number} icon The icon index for this message.
 * @param {string} sender The sender of this message.
 * @param {boolean} isRead Whether this message has been read or not.
 * @param {boolean} processed Whether this message has been processed or not.
 * @param {number} priority The priority of this message (must be either `{@link hwc.MSG_PRIORITY_HIGH}` or `{@link hwc.MSG_PRIORITY_NORMAL}`).
 * @param {string} subject The subject of this message.
 * @param {number} module The module ID of the hybrid app associated with this message.
 * @param {number} version The version of the hybrid app associated with this message.
 */

hwc.Message = function (msgId, date, icon, sender, isRead, processed, priority, subject, module, version)
{ 
this.msgId = msgId;
this.recvDate = date;
this.iconIndex = icon;
this.subject = subject;
this.moduleId = module;
this.version = version;
this.processed = processed;
this.sender = sender;
this.isread = isRead;
this.priority = priority;

/**
 * Gets the message ID of this message.
 * @public
 * @memberOf hwc.Message
 * @returns {number} The message ID of this message.
 */
this.getMessageId = function ()
{
    return this.msgId;
};

/**
 * Gets the date this message was received.
 * @public
 * @memberOf hwc.Message
 * @returns {Date} The date this message was received.
 */
this.getReceivedDate = function ()
{

}
return this.recvDate;

/**
 * Gets the icon index of this message.
 * @public
 * @memberOf hwc.Message
 * @returns {number} The icon index of this message.
 */
this.getIconIndex = function ()
{
    return this.iconIndex;
};

/**
 * Gets the sender of this message.
 * @public
 * @memberOf hwc.Message
 * @returns {string} The sender of this message.
 */
this.getSender = function ()
{
    return this.sender;
};

/**
 * Gets whether this message has been read or not.
 * @public
 * @memberOf hwc.Message
 * @returns {boolean} Whether this message has been read (true) or not (false).
 */
this.isRead = function ()
{
    return this.isread;
};

/**
 * Gets the subject of this message.
 * @public
 * @memberOf hwc.Message
 * @returns {string} The subject of this message.
 */
this.getSubject = function ()
{
    return this.subject;
};

/**
 * Gets the module ID of the hybrid app this message belongs to.
 * @public
 * @memberOf hwc.Message
 * @returns {number} The module ID of the hybrid app this message belongs to.
 */
this.getModuleId = function ()
{
    return this.moduleId;
};

/**
 * Gets the version of the hybrid app this message belongs to.
 * @public
 */
this.getModuleVersion = function ()
{
    return this.version;
};

/**
 * Gets whether this message has been processed or not. A message is generally marked as processed once
 * the user submits changes from the hybrid app that was launched from the message.
 */
this.isProcessed = function ()
{
    return this.processed;
};

/**
 * Gets the priority of the message.
 */
this.getPriority = function ()
{
    return this.priority;
};

/**
 * The version of the hybrid app this message belongs to.
 */
this.getModuleVersion = function ()
{
    return this.version;
};
this.getPriority = function () {
    if (this.priority === hwc.MSG_PRIORITY_HIGH) {
        return hwc.MSG_PRIORITY_HIGH;
    } else {
        return hwc.MSG_PRIORITY_NORMAL;
    }
};

this.updateRead = function (status) {
    hwc.updateMessageRead(this.msgId, status);
    this.isread = status;
};

this.updateProcessed = function (status) {
    /* Updates the processed status of the message. */
    /* @public */
    /* @memberOf hwc.Message */
    /* @param {boolean} status The new processed status. */
    */
    }
hwc.updateMessageProcessed(this.msgId, status);

this.processed = status;

/**
 * Represents a filter used to filter messages.
 * Pass in null for any parameter you do not wish to filter
 * (or do not pass in such parameters at all).
 */

* @classdesc
* @public
* @memberOf hwc

* @param {string} [sender] The sender of the message.
* @param {string} [subject] The subject of the message.
* @param {number} [moduleId] The associated application module ID.
* @param {number} [version] The associated application module versions.
* @param {boolean} [isread] The read status.
* @param {boolean} [processed] The processed status.

hwc.MessageFilter = function (sender, subject, moduleId, version, isread, processed)
{
    this.sender = sender;
    this.subject = subject;
    this.moduleId = moduleId;
    this.version = version;
    this.isRead = isread;
    this.processed = processed;
};
/**
 * An array of message listener callback functions.
 * @private
 * @type {anonymous.MessageListener[]} */
hwc._messageListeners = [];
/**
 * An array of objects containing message listener callback functions.
 * The containing objects need to be kept track of since the callback functions may reference
 * variables in the containing object.
 * @private
 * @type Array */
hwc._messageListenerContainingObjects = [];
/**
 * An array of \{@link hwc.MessageFilter\} objects.
 * @private
 * @type \{hwc.MessageFilter[]\} */
hwc._messageListenerFilters = []; // Array of MessageFilter
/**
 * This is the main entry of message notification. The native code should be
 * hardcoded to call this function
 * @private
 * @param {number} flag Will be one of: \{@link hwc.MSG_ADDED\}, \{@link hwc.MSG_REMOVED\}, \{@link hwc.MSG_UPDATED\}, \{@link hwc.MSG_REFRESH\} */
 * @param {number} msgId The message id that this notification is about.
 * */
hwc._messageListenerNotification = function (flag, msgId)
{
    var i, filter, msg, containingObject;

    hwc.traceEnteringMethod("hwc._messageListenerNotification");
    try {
        if (hwc._messageListeners.length === 0)
        {
            return;
        }
        msg = hwc.getMessageByID(msgId);
        for (i = 0; i < hwc._messageListeners.length; i++)
        {
            filter = hwc._messageListenerFilters[i];
            if (filter !== null && filter !== undefined)
            {
                if (msg === null)
                {
                    // a null message should pass no filter
                    continue;
                }
                if (filter.sender !== null && filter.sender !== undefined)
                {
                    if (msg.getSender().toLowerCase() !== filter.sender.toLowerCase()) {
                        continue;
                    }
                }
                if (filter.sender !== null && filter.sender !== undefined)
                {
                    if (msg.getSender().toLowerCase() !== filter.sender.toLowerCase())
                    {
                        continue;
                    }
                }
            }
        }
    }
if (filter.subject !== null && filter.subject !== undefined)
{
    if (msg.getSubject() !== filter.subject)
    {
        continue;
    }
}

if (filter.moduleId !== null && filter.moduleId !== undefined)
{
    if (msg.getModuleId() !== filter.ModuleId)
    {
        continue;
    }
}

if (filter.version !== null && filter.version !== undefined)
{
    if (msg.getVersion() !== filter.version)
    {
        continue;
    }
}

if (filter.isRead !== null && filter.isRead !== undefined)
{
    if (msg.getRead() !== filter.isRead) {
        continue;
    }
}
if (filter.processed !== null && filter.processed !== undefined) {
    if (msg.getProcessed() !== filter.processed) {
        continue;
    }
}

containingObject = hwc._messageListenerContainingObjects[i];
if (containingObject !== null && containingObject !== undefined) {
    hwc._messageListeners[i].call(containingObject, flag, msgId);
}
else {
    hwc._messageListeners[i](flag, msgId);
}
} finally {
    hwc.traceLeavingMethod("hwc._messageListenerNotification");
}

/**
 * Registers a message listener.
 *
 * @public
 * @memberOf hwc
 */
* @param {hwc.MessageFilter} filters The message filter that message events must pass to get passed to the {link anonymous.MessageListener}.

* If no filter is desired, then null can be used for this parameter.

* @param {anonymous.MessageListener} MessageListener The callback function for message changes.

* @param {Object} [containingObject] The containing object of the message listener. If a message listener callback function references variables in its containing object, then the containing object should be passed to this function.

* @example

* // soSomething is a global function called by the listener callback.

* var doSomething = function()

* {

*   alert("New message!");

* }

* // messageListener is the callback function passed to hwc.addMessageListener.

* var messageListener = function( flag, messageId )

* {

*   if( flag == hwc.MSG_ADDED )

*   {

*     doSomething();

*   }

* }

* // We do not want to filter the message events the listener will get invoked for, so pass null for the first parameter.

* hwc.addMessageListener( null, messageListener );

* @example

* // someObject is an object that will contain the listener callback as well as a variable referenced by the callback.

* var someObject = {};
2761        * // doSomething is a function referenced by the callback function.
2762        * someObject.doSomething = function()
2763        * {
2764        *    alert("New message!");
2765        * }
2766        * // messageListener is the callback that will be passed to hwc.addMessageListener.
2767        * someObject.messageListener = function( flag, messageId )
2768        * {
2769        *    if( flag == hwc.MSG_ADDED )
2770        *    {
2771        *       this.doSomething();
2772        *    }
2773        * }
2774        * // Create a filter so that not all message events will invoke our callback function.
2775        * // Only events about messages with a subject of "Subject" will trigger our callback function.
2776        * var filter = new hwc.MessageFilter( null, "Subject", null, null, null, null);
2777        * // The callback function references a variable in its containing object, so we need to pass in the containing object
2778        * // in addition to the filter and the callback function.
2779        * hwc.addMessageListener( filter, someObject.messageListener, someObject );
2780        */
2781        hwc.addMessageListener = function (filters, MessageListener, containingObject) {
2782            hwc.traceEnteringMethod("hwc.addMessageListener");
2783            try {
2784                hwc._messageListenerFilters.push(filters);
2785                hwc._messageListeners.push(MessageListener);

hwc._messageListenerContainingObjects.push(containingObject);

if (hwc._messageListeners.length === 1)
{
    hwc.getDataFromContainer("startmsglistener");
}

finally {
    hwc.traceLeavingMethod("hwc.addMessageListener");
}

/**
 * Removes the message listener. The two parameters passed in to this function should match exactly the corresponding parameters passed into {@link hwc.addMessageListener} when the message listener was added.
 *
 * @public
 * @memberOf hwc
 * @param {anonymous.MessageListener} MessageListener The callback for message changes.
 * @param {Object} [containingObject] If the containing object was given to {@link hwc.addMessageListener} when the message listener was added, then it also must be passed into this function.
 *
 * @example
 * // soSomething is a global function called by the listener callback.
 * var doSomething = function()
 * {
 *     alert("New message!");
 * }
 * // messageListener is the callback function passed to hwc.addMessageListener.
 * var messageListener = function( flag, messageId )
if (flag == hwc.MSG_ADDED)
{
    doSomething();
}

// We do not want to filter the message events the listener will get invoked for, so pass null for the first parameter.

hwc.addMessageListener(null, messageListener);

// If we want to remove the listener at some other point, use the following line of code:

hwc.removeMessageListener(messageListener);

@example

// someObject is an object that will contain the listener callback as well as a variable referenced by the callback.

var someObject = {};

// doSomething is a function referenced by the callback function.

someObject.doSomething = function()
{
    alert("New message!");
}

// messageListener is the callback that will be passed to hwc.addMessageListener.

someObject.messageListener = function (flag, messageId)
{
    if (flag == hwc.MSG_ADDED)
    {
        this.doSomething();
    }
}
* // Create a filter so that not all message events will
* invoke our callback function.

* // Only events about messages with a subject of "SI<4>"
* will trigger our callback function.

* var filter = new hwc.MessageFilter( null, "SI<4>", null,
null, null, null);

* // The callback function references a variable in its
* containing object, so we need to pass in the containing object

* // in addition to the filter and the callback
* function.

* hwc.addMessageListener( filter,
someObject.messageListener, someObject );

* // If we want to remove the listener at some other point,
* use the following line of code:

* hwc.removeMessageListener( messageListener,
someObject );

* /

hwc.removeMessageListener = function (MessageListener,
containingObject)
{
    var i;
    hwc.traceEnteringMethod("hwc.removeMessageListener");
    try {
        if (hwc._messageListeners.length === 0) {
            return;
        }

        for (i = 0; i < hwc._messageListeners.length; i++)
        {
            if (hwc._messageListeners[i] === MessageListener+
                containingObject)
            {
                hwc._messageListenerContainingObjects[i] ===
                containingObject
                {
                    hwc._messageListeners.splice(i, 1);
                    hwc._messageListenerFilters.splice(i, 1);
hwc._messageListenerContainingObjects.splice(i, 1);

if (hwc._messageListeners.length === 0)
{
    hwc.getDataFromContainer("stopmsglistener");
    return;
}

finally {
    hwc.traceLeavingMethod("hwc.removeMessageListener");
}

/**
 * A constant indicating that a message needs to be refreshed. Used in {@link anonymous.MessageListener} callback functions.
 * @type number
 */

hwc.MSG_REFRESH = 1;

/**
 * A constant indicating that a message has been added. Used in {@link anonymous.MessageListener} callback functions.
 * @type number
 */

hwc.MSG_ADDED = 2;

/**
 * A constant indicating that a message has been updated. Used in {@link anonymous.MessageListener} callback functions.
 * @type number
 */

hwc.MSG_UPDATED = 3;
/**
 * A constant indicating that a message has been removed. Used in
 * @link anonymous.MessageListener} callback functions.
 * @type number
 */

hwc.MSG_REMOVED = 4;

/**
 * A constant indicating a message has normal priority. Used in
 * @link hwc.Message}.
 * @type number
 */

hwc.MSG_PRIORITY_NORMAL = 1;

/**
 * A constant indicating a message has high priority. Used in
 * @link hwc.Message}.
 * @type number
 */

hwc.MSG_PRIORITY_HIGH = 3;

/**
 * A sample @link anonymous.MessageListener} callback function.
 *
 * @param {number} flag A number indicating which message
 * event occurred (will be one of MSG_* constants).
 *
 * @param {number} msgId The message id of the affected
 * message.
 */

hwc.sample_MessageListener = function (flag, msgId) {
};

/**
 * Gets received messages based on a filter and the
 * existence of a default hybrid app.
 */
* @public

* @memberOf hwc

* @param {hwc.MessageFilter} [messageFilter] A filter that all returned messages will pass.

* If you do not want to filter based on a certain attribute, use null for that attribute when creating the filter.

* If you do not want to filter at all, pass in null for this parameter or do not pass in this parameter at all.

* @param {boolean} [completeList] If this parameter is set to true, then all messages will be returned.

* If this parameter is set to false or if it is not set, then if there is a default hybrid app only the messages belonging to the default hybrid app will be returned (and if there is no default hybrid app all messages will be returned).

* @returns {hwc.Message[]} An array of {link hwc.Message} objects - the received messages.

* // get all messages that have the subject "a subject".

* var filter = new hwc.MessageFilter( null, "a subject", null, null, null, null );

* var messages = hwc.getAllMessages(filter);

* // Get all messages without filtering, but if there is a default hybrid app only return its messages.

* var messages = hwc.getAllMessages();

* // Get all messages (without filtering) for all hybrid apps, even if there is a default hybrid app.

* var messages = hwc.getAllMessages( null, true );

* /

hwc.getAllMessages = function (filters, completeList) {
  var filtersUrlString, messages, i, message,
  formattedCompleteList,response, messageInstances;

hwc.traceEnteringMethod("hwc.getAllMessages");
formattedCompleteList = false;
response = "";
messageInstances = [];

if ( completeList )
{
    formattedCompleteList = true;
}

try {
    // Create filter url argument
    filtersUrlString = "";
    if( filters )
    {
        if( filters.sender !== undefined && filters.sender !== null )
        {
            filtersUrlString = filtersUrlString + 
"&filtermessagesender=" + encodeURIComponent(filters.sender);
        }
        if( filters.subject !== undefined && filters.subject !== null && filters.subject !== undefined)
        {
            filtersUrlString = filtersUrlString + 
"&filtermessagesubject=" + encodeURIComponent(filters.subject);
        }
        if( filters.moduleId !== undefined && filters.moduleId !== null )
        {
            filtersUrlString = filtersUrlString + 
"&filtermessagemoduleid=" + encodeURIComponent(filters.moduleId);
        }
        if( filters.moduleId !== undefined && filters.moduleId !== null )
        {
            filtersUrlString = filtersUrlString + 
"&filtermessagemoduleid=" + encodeURIComponent(filters.moduleId);
        }
    }
}
if( filters.version !== undefined && filters.version !== null )
{
    filtersUrlString = filtersUrlString + 
    "&filtermessageversion=" + encodeURIComponent(filters.version);
}
if( filters.isRead !== undefined && filters.isRead !== null )
{
    filtersUrlString = filtersUrlString + 
    "&filtermessageisread=" + encodeURIComponent(filters.isRead);
}
if( filters.processed !== undefined && filters.processed !== null )
{
    filtersUrlString = filtersUrlString + 
    "&filtermessageisprocessed=" + 
    encodeURIComponent(filters.processed);
}
filtersUrlString += 
    "&getcompletelist=" + formattedCompleteList;
response = hwc.getDataFromContainer("getmessages", filtersUrlString);
if (response !== null && response !== undefined && response !== 
"")
{
    messages = JSON.parse(response);
    for (i=0; i<messages.length; i++)
    {
        message = messages[i];
        messageInstances[i] = new hwc.Message(message.id, new Date(message.milliseconds),
message.iconIndex, message.sender, message.isRead,}
message.isProcessed, message.priority, message.subject, message.module, message.version); } } } catch (ex){
    hwc.log("messages.getAll error:" + ex.message, "ERROR", false);
} finally {
    hwc.traceLeavingMethod("hwc.getAllMessages");
} return messageInstances;

/**
 * Gets a {link hwc.Message} object with the given message ID.
 * 
 * @public
 * @memberOf hwc
 * @param {number} msgId The message ID of the message to get.
 * 
 * @returns {hwc.Message} A message object, or null if no message with given ID.
 * 
 * @example
 * // A message listener is one place that would likely need to call hwc.getMessageByID.
 * var messageListener = function( flag, messageID )
 * {
 *     // Since the callback function only gets the messageId, not the message itself, if we want
 *     // more information about the message we must call hwc.getMessageByID.
```javascript
var message = hwc.getMessageByID( messageID );

if( message.getSubject() == "a special subject" )
{
    alert( "An event occurred for a special message!" );
}

hwc.addMessageListener( null, messageListener );

hwc.getMessageByID = function (msgId)
{
    var response, messageInstance, message;

    hwc.traceEnteringMethod("hwc.getMessageByID");
    response = ";
    messageInstance = null;

    try {
        response = hwc.getDataFromContainer("getmessagebyid", "&msgid=" + msgId);

        if (response !== null && response !== undefined && response !== ")
        {
            message = JSON.parse(response);
            messageInstance = new hwc.Message(message.id, new Date(message.milliseconds), message.iconIndex, message.sender, message.isRead, message.isProcessed, message.priority, message.subject, message.module, message.version);
        }
    }catch (ex){
        hwc.log("messages.getMessageByID error:" + ex.message, "ERROR", false);
    } finally {
        try {
            message = JSON.parse(response);
            messageInstance = new hwc.Message(message.id, new Date(message.milliseconds), message.iconIndex, message.sender, message.isRead, message.isProcessed, message.priority, message.subject, message.module, message.version);
        }catch (ex){
            hwc.log("messages.getMessageByID error:" + ex.message, "ERROR", false);
        }
    }
```
hwc.traceLeavingMethod("hwc.getMessageByID");

return messageInstance;

/**
   * Updates the message read status.
   *
   * @public
   * @memberOf hwc
   * @param {number} msgId The id of message to update the read status for.
   * @param {boolean} status Whether the message will be set to read (true) or unread (false).
   * @example
   * // set all messages as read
   * var messages = hwc.getAllMessages();
   * for( var index = 0; index < messages.length; index++ )
   * {
   *     hwc.updateMessageRead( messages[index].getMessageId(), true );
   * }
   */

hwc.updateMessageRead = function (msgId, status)
{
    var updateParms;
    hwc.traceEnteringMethod("hwc.updateMessageRead");
    try {
        updateParms = ";msgid=" + msgId + ";msgfield=read" + ";status=" + status;
        hwc.getDataFromContainer("updatemessage", updateParms);
    } catch (e) {
        console.error(e);
    }
}
```javascript
} catch (ex) {
    hwc.log("Message.updateMsgRead error:" + ex.message, "ERROR", false);
} finally {
    hwc.traceLeavingMethod("hwc.updateMessageRead");
}

/**
 * Updates the message processed status.
 *
 * @public
 * @memberOf hwc
 * @param {number} msgId The id of message to update the processed status for.
 * @param {boolean} status Whether the message will be set to processed (true) or unprocessed (false).
 * @example
 * // set all messages as processed
 * var messages = hwc.getAllMessages();
 * for( var index = 0; index < messages.length; index++ )
 * {
 *     hwc.updateMessageProcessed( messages[index].getMessageId(), true );
 * }
 * @example
 * @example
 * hwc.updateMessageProcessed = function (msgId, status)
 * {
 *     var updateParms;
 *     hwc.traceEnteringMethod("hwc.updateMessageProcessed");
 *     try {
 *         updateParms = ";&msgid=" + msgId + ";&msgfield=processed" + ";&status=" + status;
 *     } catch (ex) {
 *         hwc.log("Message.updateMsgRead error:" + ex.message, "ERROR", false);
 *     }
 *     hwc.traceLeavingMethod("hwc.updateMessageProcessed");
 *     var url = hwc.getMessageBaseURL() + ";" + updateParms;
 *     var request = hwc.makeRequest(
 *         url, "GET", null, null, null,
 *         function (response) {
 *             // handle response
 *         },
 *         function (error) {
 *             // handle error
 *         }
 *     );
 * }
 */
```

hwc.getDataFromContainer("updatemessage", updateParms);

} catch (ex){

    hwc.log("Message.updateMsgProcessed error:" + ex.message, "ERROR", false);

} finally {

    hwc.traceLeavingMethod("hwc.updateMessageProcessed");

}

/**
 * Removes (deletes) a message.
 *
 * @public
 * @memberOf hwc
 * @param {number} msgId The id of the message to be removed.
 *
 * @example
 * // remove all messages
 * var messages = hwc.getAllMessages();
 * for( var index = 0; index < messages.length; index++ )
 * {
 *    hwc.removeMessage( messages[index].getMessageId() );
 * }
 */

hwc.removeMessage = function(msgId) {

    hwc.traceEnteringMethod("hwc.removeMessage");

    try {

        hwc.getDataFromContainer("removemessage", ";&msgid=" + msgId);

    } catch (ex){hwc.log("messages.remove error:" + ex.message, "ERROR", false);}

    hwc.traceLeavingMethod("hwc.removeMessage");

};
finally
{hwc.traceLeavingMethod("hwc.removeMessage");}

/**
 * A constant indicating that a message was successfully opened. This is a possible return value for {@link hwc.openMessage}.
 * @type number
 */
hwc.OPEN_MSG_SUCCESS = 0;

/**
 * A constant indicating that a message could not be opened because no message with the given ID exists.
 * This is a possible return value for {@link hwc.openMessage}.
 * @type number
 */
hwc.OPEN_MSG_NOT_EXIST = 1;

/**
 * A constant indicating that a message could not be opened because there was no associated hybrid app.
 * This is a possible return value for {@link hwc.openMessage}.
 * @type number
 */
hwc.OPEN_MSG_APP_NOT_EXIST = 2;

/**
 * A constant indicating that a message could not be opened due to an unspecified error.
 * This is a possible return value for {@link hwc.openMessage}.
 * @type number
 */
hwc.OPEN_MSG_OTHER = 3;
* Launch the server initiated hybrid app associated with a message. The hybrid app will be opened on top of the hybrid app that is open when `hwc.openMessage` is called. When the hybrid app that was opened with `hwc.openMessage` exits, it will exit to the hybrid app that was open when `hwc.openMessage` was called. It is possible to nest open hybrid apps, but it is best not to have too many nested hybrid apps (e.g.: recursively opening hybrid apps) because each open hybrid app takes up device memory.

* @public
* @memberOf hwc
* @param {number} msgId The id of message to open.

* @returns {number} A number indicating the success or failure of opening the message (will be one of @link `hwc.OPEN_MSG_SUCCESS`), @link `hwc.OPEN_MSG_NOT_EXIST`, @link `hwc.OPEN_MSG_APP_NOT_EXIST`, @link `hwc.OPEN_MSG_OTHER`).

* @example
* // get all messages, then open the first one
* var messages = hwc.getAllMessages();
* if( messages.length > 0 )
* {
*    hwc.openMessage( messages[0].getMessageId() );
* }
```javascript
3191            return parseInt(response, 10);
3192        } catch (ex){
3193            hwc.log("messages.open error:" + ex.message, "ERROR", false);
3194        } finally {
3195            hwc.traceLeavingMethod("hwc.openMessage");
3196        }
3197    }
3198
3199    /**
3200    * This function takes care of handling the XML HTTP request to communicate with the HWC native code on the different platforms.
3201    *
3202    * @private
3203    * @memberOf hwc
3204    *
3205    * @param {string} queryType A string indicating the type of query being sent to the native code.
3206    * This parameter must match up with a constant defined in the native code of the HWC.
3207    * @param {string} urlParams A string of parameters for the query, in a format such that it can be added directly to the url.
3208    * @returns {string} The response text of the request.
3209    * @example
3210    * // This example is an excerpt from hwc.getInstalledApps. There are many examples of how to use this function in this file.
3211    * response = hwc.getDataFromContainer("getinstalledapps", ";getcompletelist=true");
3212    * if (response != null && response != undefined && response != "")
3213    * {
3214    *    var apps = JSON.parse(response);
```
for(var i = 0; i < apps.length; i++) {
    var app = apps[i];
    installedApps[i] = new hwc.HybridApp(app.moduleId, app.version, app.displayName, app.iconIndex,
        hwc.createCustomIconObject(app.defaultCustomIcon, app.moduleId, app.version, hwc.DEFAULT_CUSTOM_ICON_INDEX),
        hwc.createCustomIconList(app.customIconList, app.moduleId, app.version));
}

hwc.getDataFromContainer = function(queryType, urlParams)
{
    var response, xmlhttp;
    hwc.traceEnteringMethod("hwc.getDataFromContainer");
    response = ";
    if (urlParams === null || urlParams === undefined) {
        urlParams = ";
    }
    try {  
        if (hwc.isWindowsMobile()) {
            xmlhttp = hwc.getXMLHTTPRequest();
            xmlhttp.open("GET", "/sup.amp?querytype=" + queryType + ";" + hwc.versionURLParam + ";" + urlParams, false );
            xmlhttp.send("\n");
            response = xmlhttp.responseText;
        }
        else if (hwc.isAndroid()) {
            response = _HWC.getData("http://localhost/sup.amp?querytype=" + queryType + ";" + hwc.versionURLParam + urlParams);
        }
    }
}
else if (hwc.isBlackBerry()) {
    // CR661210 and NA3-2487
    if (hwc.isClosed()) {
        return;
    }

    xmlhttp = hwc.getXMLHTTPRequest();
    xmlhttp.open("POST", "http://localhost/sup.amp?querytype=" + queryType + "+" + hwc.versionURLParam + urlParams, false);
    xmlhttp.send();
    response = xmlhttp.responseText;
}

else if (hwc.isIOS()) {
    xmlhttp = hwc.getXMLHTTPRequest();
    xmlhttp.open("GET", "http://localhost/sup.amp?querytype=" + queryType + "+" + hwc.versionURLParam + urlParams, false);
    try
    {
        xmlhttp.send(""");
    }
    catch (ex)
    {
        if (ex.message.search(/XMLHttpRequest Exception 101/) === -1)
        {
            throw ex;
        }
    }
    response = xmlhttp.responseText;
}

return response;
catch (ex1)
{
    hwc.log("hwc.getDataFromContainer error: " +
    ex1.message, "ERROR", false);
} finally {

hwc.traceLeavingMethod("hwc.getDataFromContainer");

}

/**
* This function takes care of handling the XML HTTP request to communicate with the HWC native code on different platforms.
* @private
* @memberOf hwc

* @param {string} queryType Indicates the type of query being sent to the native code.

* @param {string} data Data to be sent with the request.

* @returns {string} The response text of the request.
*/

hwcpostDataToContainer = function(queryType, data)
{
    var response, xmlhttp;
    response = "";
    try
    {
        if (hwc.isWindowsMobile()) {
            xmlhttp = hwc/XMLHTTPRequest();
        
    
SAP Mobile Platform
xmlhttp.open("POST", "/sup.amp?querytype=" + queryType + "&" + hwc.versionURLParam, false);

xmlhttp.send(data);

response = xmlhttp.responseText;

}

else if (hwc.isAndroid()) {

response = _HWC.postData("http://localhost/sup.amp?querytype=" + queryType + "&" + hwc.versionURLParam, data);

}

else if (hwc.isBlackBerry()) {

// CR661210 and NA3-2487

if (hwc.isClosed()) {

return;

}

xmlhttp = hwc.getXMLHTTPRequest();

xmlhttp.open("POST", "http://localhost/sup.amp?querytype=" + queryType + "&" + hwc.versionURLParam, false);

xmlhttp.send(data);

response = xmlhttp.responseText;

}

else if (hwc.isIOS()) {

xmlhttp = hwc.getXMLHTTPRequest();

xmlhttp.open("POST", "http://localhost/sup.amp?querytype=" + queryType + "&" + hwc.versionURLParam, false);

try

xmlhttp.send(data);

}

catch (ex)

{

if (ex.message.search(/XMLHttpRequest Exception 101/) === -1)


throw ex;
}
}
response = xmlhttp.responseText;
}
return response;
}
catch (ex1)
{
    hwc.log( "hwc.postDataToContainer error: " + ex1.message, "ERROR", false);
}

var partialRequestUrl = null;

/**
 * Gets a URL that can be used to get resources from the 
 * HWC.
 * 
 * @private
 * @memberOf hwc
 * @param {string} queryType The type of query
 * @param {string} urlParams Additional parameters to send with the request. Must be formed such that it can be appended to the url
 * (eg: "firstParam=value1&secondParam=value2").
 * 
 * @returns {string} A URL that can be used to access resources.
 */
getRequestUrl = function ( queryType, urlParams )
{
// Lazy load to prevent platform identification errors

if (!partialRequestUrl)
{
  partialRequestUrl = hwc.isWindowsMobile() ? "/sup.amp?querytype=" :

  hwc.isAndroid() ?

    ( window.location.protocol + "//" +
    window.location.hostname + "/" +
    window.location.pathname.split( '/' )[1] + "/sup.amp/
    querytype=" ) :

  hwc.isBlackBerry() || hwc.isIOS() ? "http://localhost/sup.amp?querytype=" :

  ";

}

return partialRequestUrl + queryType + "&" +
hwc.versionURLParam + (urlParams ? '&' : "") + urlParams;

/**
 * Represents a Media Cache. This object gives the option
to use the cache when accessing .
 *
 * @classdesc
 * @public
 * @memberOf hwc
 * @static
 */

hwc.MediaCache = {};

/**
 * hwc.MediaCache.Policy An object containing constants
representing the different caching policies.
 *
 * @memberOf hwc.MediaCache
 */
hwc.MediaCache.Policy = {};

/**
 * hwc.MediaCache.Policy.SERVER_FIRST Use server first policy: requests will only be served from the cache if the server is unavailable.
 */

hwc.MediaCache.Policy.SERVER_FIRST = "ServerFirst";

/**
 * hwc.MediaCache.Policy.CACHE_FIRST Use cache first policy: requests will be served from the cache if possible.
 */

hwc.MediaCache.Policy.CACHE_FIRST = "CacheFirst";

/**
 * Creates a media cache URL for the resource. The cache first policy will be used if no policy is specified.
 */

* @public

* @memberOf hwc.MediaCache

* @param {string} resourceUrl The URL to the resource

* @param {hwc.MediaCache.Policy} [policy] The optional cache policy to use.

* If set, it must be either {@link hwc.MediaCache.Policy.SERVER_FIRST} or {@link hwc.MediaCache.Policy.CACHE_FIRST}.

* Default policy is cache first.

* @returns {string} The URL that can be used to access the resource with the specified caching policy.

* @example
This line creates a url that can be used to retrieve the picture from the cache if possible, and from the server otherwise.

```javascript
```

The following function adds a picture to the page. Since the mediaCacheURL variable is used for the url, the picture will be retrieved from the cache if possible.

```javascript
var addPicFromMediaCache = function()
{
    // Create the image element.
    var image = document.createElement( "img" );
    // Set the source of the image to the media cache URL.
    image.setAttribute( 'src', mediaCacheURL );
    // Add the image element to the page.
    document.body.appendChild( image );
}
```

This line creates a url that can be used to retrieve the picture from the server if it is available, or the cache otherwise.

```javascript
var mediaCacheURL_serverFirst = hwc.MediaCache.getUrl( "http://yourserver.com/Pictures/pentagon.jpg", hwc.MediaCache.Policy.SERVER_FIRST );
```

The following function adds a picture to the page. Since the mediaCacheURL_serverFirst variable is used for the url, the picture will be gotten from the server if the server is available, and from the cache otherwise.

```javascript
var addPicFromMediaCache_ServerFirst = function()
{
    // Create the image element.
    var image = document.createElement( "img" );
```
3435        *     // Set the source of the image to the media cache URL.
3436        *     image.setAttribute('src', mediaCacheURL_serverFirst);
3437        *     // Add the image element to the page.
3438        *     document.body.appendChild(image);
3439        * }
3440        *
3441        */
3442       hwc.MediaCache.getUrl = function (resourceUrl, policy) {
3443           hwc.traceEnteringMethod("hwc.MediaCache.getUrl");
3444           try {
3445               policy = policy ? policy : hwc.MediaCache.Policy.CACHE_FIRST;
3446               return getRequestUrl("mediacache", "url=" + encodeURIComponent(resourceUrl) + 
3447                   "&policy=" + policy + 
3448                   
3449                   
3450       }
3451       
3452       
3453       /**<
3454       * Represents an E2E Trace. This object is used for debugging and analysis.
3455       * @classdesc
3456       */
3457       hwc.e2eTrace = {};
3458       hwc.e2eTrace.TraceLevel = {};
3460       /**<
* A constant indicating a high level of detail for the trace.
* Use this level for functional analysis and detailed functional logging and tracing.
* @type string
* @memberOf hwc.e2eTrace

```javascript
hwc.e2eTrace.TraceLevel.HIGH = "HIGH";
```

/**
 * A constant indicating a low level of detail for the trace.
 * Use this level for response-time-distribution analysis: see how much time is spent on each server component to find bottlenecks.
 * @type string
 * @memberOf hwc.e2eTrace

```javascript
hwc.e2eTrace.TraceLevel.LOW = "LOW";
```

/**
 * A constant indicating a medium level of detail for the trace.
 * Use this level for performance analysis (performance traces are triggered on server-side).
 * @type string
 * @memberOf hwc.e2eTrace

```javascript
hwc.e2eTrace.TraceLevel.MEDIUM = "MEDIUM";
```

/**
 * Gets whether the e2e tracing has been requested to be started.
 * This function returns true between calls to {@link hwc.e2eTrace#startTrace} and {@link hwc.e2eTrace#stopTrace}.
 * @type boolean
 * @memberOf hwc.e2eTrace
```
* @returns {boolean} True if trace is enabled, false otherwise.
 *
 hwc.e2eTrace.isTraceEnabled = function() {

 hwc.traceEnteringMethod("hwc.e2eTrace.isTraceEnabled");

 try {

 return parseBoolean(hwc.getDataFromContainer("e2etrace", "&method=istraceenabled"));

 } finally {

 hwc.traceLeavingMethod("hwc.e2eTrace.isTraceEnabled");

 }

 };

 /**
 * Sets the passport e2eTrace level. This function must be
called before {@link hwc.e2eTrace#startTrace}.
 *
 * @memberOf hwc.e2eTrace
 *
 * @param {string} The trace level. Must be one of {@link
hwc.e2eTrace.TraceLevel.LOW}, {@link
hwc.e2eTrace.TraceLevel.MEDIUM}, or
*
 * {@link hwc.e2eTrace.TraceLevel.HIGH}.
 *
 * @returns {boolean} True if trace is enabled, false otherwise.
 */

 hwc.e2eTrace.setTraceLevel = function(level) {

 hwc.traceEnteringMethod("hwc.e2eTrace.setTraceLevel");

 try {

 hwc.getDataFromContainer("e2etrace", "&method=settracelevel&level=" + level);

 } finally {

 hwc.traceLeavingMethod("hwc.e2eTrace.setTraceLevel");

 };
/**
 * Starts tracing user actions and requests. Before this function is called, the trace level must be set with
 * @link hwc.e2eTrace#setTracelevel).
 *
 * @memberOf hwc.e2eTrace
 */

hwc.e2eTrace.startTrace = function() {
    hwc.traceEnteringMethod("hwc.e2eTrace.startTrace");
    try {
        hwc.getDataFromContainer("e2etrace", "&method=starttrace");
    } finally {
    hwc.traceLeavingMethod("hwc.e2eTrace.startTrace");
    }
};

/**
 * Stops tracing user actions and requests.
 *
 * @memberOf hwc.e2eTrace
 */

hwc.e2eTrace.stopTrace = function() {
    hwc.traceEnteringMethod("hwc.e2eTrace.stopTrace");
    try {
        hwc.getDataFromContainer("e2etrace", "&method=stoptrace");
    } finally {
    hwc.traceLeavingMethod("hwc.e2eTrace.stopTrace");
    }
};
hwc.e2eTrace.uploadTrace = function() {
  hwc.traceEnteringMethod("hwc.e2eTrace.uploadTrace");
  try {
    return parseBoolean(hwc.getDataFromContainer("e2etrace", "&method=uploadtrace"));
  } finally {
    hwc.traceLeavingMethod("hwc.e2eTrace.uploadTrace");
  }
}

/**
 * Represents the Performance Manager.
 * @classdesc
 * @memberOf hwc
 * @example
 * // Start performance collection.
 * if (hwc.perf.isEnabled())
 * {
 *   hwc.perf.stopInteraction();
 * }
 * 
 * hwc.perf.startInteraction('someinteraction'); // Start an optional interval.
* Upload the Business Transaction XML (BTX) to the server.
* To upload, the SAP Solution Manager URL must be set in SAP Control Center configuration.
* @memberOf hwc.e2eTrace
* @returns {boolean} True if the upload is successful, false otherwise.
*/

Develop Hybrid Apps Using Third-party Web Frameworks

SAP Mobile Platform
* // Stop performance collection. Logs will be written.
  if (hwc.perf.isEnabled())
  {
    hwc.perf.stopInterval('IntervalName'); // Stop an optional interval.
    hwc.perf.stopInteraction();
  }
*/

hwc.perf = {};

/**
 * Gets whether the performance agent is enabled.
 * @memberOf hwc.perf
 * @returns {boolean} True if the performance agent is enabled, false otherwise.
 */

hwc.perf.isEnabled = function() {
  hwc.traceEnteringMethod("hwc.perf.isEnabled");
  try {
    return parseBoolean(hwc.getDataFromContainer("perf", 
"&method=isenabled"));
  } finally {
    hwc.traceLeavingMethod("hwc.perf.isEnabled");
  }
};

/**
 * Starts the interaction.
 * @memberOf hwc.perf
 * @param {string} interactionName The name of the interaction.
 */
hwc.perf.startInteraction = function(interactionName) {  
    hwc.traceEnteringMethod("hwc.perf.startInteraction");  
    try {  
        hwc.getDataFromContainer("perf",  
            
"&method=startinteraction&interactionname=" +  
encodeURIComponent(interactionName));  
    } finally {  
    hwc.traceLeavingMethod("hwc.perf.startInteraction");  
    }  
  }  

/**  
* Stops the interaction.  
* @memberOf hwc.perf  
*/  

hwc.perf.stopInteraction = function() {  
    hwc.traceEnteringMethod("hwc.perf.stopInteraction");  
    try {  
        hwc.getDataFromContainer("perf",  
            
"&method=stopinteraction");  
    } finally {  
    hwc.traceLeavingMethod("hwc.perf.stopInteraction");  
    }  
  }  

/**  
* Starts an interval.  
* @memberOf hwc.perf  
* @param {string} intervalName The name of the interval.  
* @param {string} intervalType The type of the interval.  
*/
hwc.perf.startInterval = function(intervalName, intervalType) {
    hwc.traceEnteringMethod("hwc.perf.startInterval");
    try {
        hwc.getDataFromContainer("perf", 
            
            
            "$\text{method=startinterval&intervalname=}
            
            \text{encodeURIComponent(intervalName)}
            
            "$\text{&intervaltype=}
            
            \text{encodeURIComponent(intervalType)}\));
        } finally {
        hwc.traceLeavingMethod("hwc.perf.startInterval");
    }
};

/**
 * Stops the interval.
 * @memberof hwc.perf
 * @param {string} intervalName The name of the interval.
 */

hwc.perf.stopInterval = function(intervalName) {
    hwc.traceEnteringMethod("hwc.perf.stopInterval");
    try {
        hwc.getDataFromContainer("perf", 
            "$\text{method=stopinterval&intervalname=}
            
            \text{encodeURIComponent(intervalName)}\));
        } finally {
        hwc.traceLeavingMethod("hwc.perf.stopInterval");
    }
};

/**
 * Internal function to parse a boolean
 * @private
 */
function parseBoolean(val)
{
    return val === 'true';
}
})})(hwc);

/**
* Used to group anonymous objects and callback functions used as method parameters. Methods and fields in this
* namespace cannot be instantiated. Used for API docs generation only.
* @namespace
*/
anonymous = (typeof anonymous === "undefined" || !anonymous) ? {} : anonymous;      // SUP 'namespace'

/**
* Callback function that will be invoked when the connection state changes. Connection listeners can be added with {@link hwc.addConnectionListener}.
* @name anonymous.ConnectionStateListener
* @param {number} event A number indicating the event that occurred (will be {@link hwc.CONNECTED} or {@link hwc.DISCONNECTED}).
* @param {number} errorCode An error code (0 indicating success).
* @param {string} errorMessage Text of the error message. Will be empty if there is no error.
*/
* Callback function that will be invoked when events are logged to the event log. Log listeners can be added with `@link hwc.addLogListener`.

* @name anonymous.LogListener

* @param {number} milliseconds The date of the log message represented in milliseconds.

* @param {number} event A number that represents which category this event falls under (It will be one of `@link hwc.CONNECTION_ERROR`, `@link hwc.CONNECTION_OTHER`, `@link hwc.CONNECTION_CONNECTED`, `@link hwc.CONNECTION_DISCONNECTED`, `@link hwc.CONNECTION_RETRIEVED_ITEMS`).

* @param {string} optionalString The string carrying the message of the log event.

* @function

---

* Callback function that will be invoked on hybrid app installation events. App installation listeners can be added with `@link hwc.addAppInstallationListener`.

* @name anonymous.AppInstallationListener

* @param {number} event A number indicating the event (will be either `@link hwc.INSTALLATION_BEGIN` or `@link hwc.INSTALLATION_END`).

* @param {string} moduleId The module ID of the hybrid app the event is about.

* @param {string} version The version of the hybrid app the event is about.

* @param {string} moduleName The display name of the hybrid app the event is about.

* @function
* Callback function that will be invoked when push notifications are available.

* Push notification listeners can be added with {

* @name anonymous.PushNotificationListener

* @param {Array} notifications An array of notifications.

* @returns {number} A number indicating whether other push notification listeners should be called after this one.

* Must be either {hwc.NOTIFICATION_CANCEL} (if no more listener callbacks should be called) or {hwc.NOTIFICATION_CONTINUE}

* (if more listener callbacks should be called).

* Callback function that will be invoked on hybrid app installation events.

* @name anonymous.AppInstallationListener

* @param {Integer} event Installation flags including, BEGIN(1), END(2), FAIL(3)

* @param {String} moduleId Optional Module Id

* @param {String} version Optional Module version

* @param {String} moduleName Optional Module display name

* @param {String} designerVersion Optional Version of designer used to create app

* @param {String} containerVersion Optional Version of hybrid web container

*/
* Callback function that will be invoked on hybrid app installation events.

* @name anonymous.AppInstallationListener

* @param {Integer} event Installation flags including, BEGIN(1), END(2)

* @param {String} moduleId Optional Module Id

* @param {String} version Optional Module version

* @param {String} moduleName Optional Module display name

* @callback

* @function

*/

/**

* Callback function that will be invoked on hybrid app events.

* Application listeners can be added with {@link hwc.addAppListener}.

* @name anonymous.ApplicationListener

* @param {number} event A number indicating what event has taken place (will be one of {@link hwc.APP_REFRESH}, {@link hwc.APP_ADDED}, {@link hwc.APP_UPDATED}, {@link hwc.APP_REMOVED}).

* @param {number} moduleId The module id of the hybrid app the event is about.

* @param {number} version module The version of the hybrid app the event is about.

* @function

*/

/**

* Callback function that will be invoked on message events. Message listeners can be added with {@link hwc.addMessageListener}.
Develop Hybrid Apps Using Third-party Web Frameworks

    3758     *
    3759     * @name anonymous.MessageListener
    3760     *
    3761     * @param {number} flag A number indicating which message
    3762     * event occurred (will be one of {@link hwc.MSG_ADDED}, {@link
    3763     * hwc.MSG_REMOVED},
    3764     * {@link hwc.MSG_UPDATED}, {@link hwc.MSG_REFRESH}).
    3765     * @param {number} msgId The message id of the affected
    3766     * message.
    3767     *
    3768     * @function
    3769     */

hwc-comms.js

    1     /**
    2     * Sybase Hybrid App version 2.3.4
    3     *
    4     * API.js
    5     * This file will not be regenerated, so it is possible to
    6     * modify it, but it
    7     * is not recommended.
    8     *
    9     * The template used to create this file was compiled on Thu
    10    Jun 07 14:57:11 EDT 2012
    11     *
    12     * Copyright (c) 2012 Sybase Inc. All rights reserved.
    13     */
    14    /**
    15    * Holds all the Hybrib Web Container javascript
    16    */
    17    hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc;  // SUP 'namespace'
/**
 * Global Legacy Mapping
 * Needed because called by generated HTML or hardcoded in workflow.js or XBWUtil.java or in container callbacks.
 */

/**
 * @deprecated Deprecated since version 2.2 - use hwc.guid()
 */

function guid() {
    return hwc.guid();
}

/**
 * @deprecated Deprecated since version 2.2 - use hwc.getXMLHTTPRequest()
 */

function getXMLHTTPRequest() {
    return hwc.getXMLHTTPRequest();
}

/**
 * @deprecated Deprecated since version 2.2 - use hwc.log(sMsg, eLevel, notifyUser)
 */

function logToWorkflow(sMsg, eLevel, notifyUser) {
    return hwc.log(sMsg, eLevel, notifyUser);
}

/**
 * @deprecated Deprecated since version 2.2 - use hwc.close()
 */

function closeWorkflow() {
    return hwc.close();
}
function clearCacheItem(cachekey) {
  return hwc.clearCacheItem(cachekey);
}

function clearCache() {
  return hwc.clearCache();
}

function expireCredentials() {
  return hwc.expireCredentials();
}

function showCertificatePicker() {
  return hwc.showCertificatePicker();
}

function saveLoginCertificate(certificate) {
  return hwc.saveLoginCertificate(certificate);
}

function saveLoginCredentials(userName, password) { 
  return hwc.saveLoginCredentials(userName, password);
}
function saveLoginCredentials(userName, password) { return hwc.saveLoginCredentials(userName, password); }

/**
 * @deprecated Deprecated since version 2.2 - use hwc.activationRequired()
 */

function activationRequired() { return hwc.activationRequired(); }

/**
 * @deprecated Deprecated since version 2.2 - use hwc.showUrlInBrowser(url)
 */

function showUrlInBrowser(url) { return hwc.showUrlInBrowser(url); }

/**
 * @deprecated Deprecated since version 2.2 - use hwc.markAsProcessed()
 */

function markAsProcessed() { return hwc.markAsProcessed(); }

/**
 * @deprecated Deprecated since version 2.2 - use hwc.markAsActivated()
 */

function markAsActivated() { return hwc.markAsActivated(); }

/**
 * Delegate for data message processing details. In the custom case, the user is expected to provide their own implementation.
 */
function processDataMessage(incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType) {
    if (typeof(hwc.processDataMessage) === 'function') {
        return hwc.processDataMessage(incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType);
    }
    else {
        // get the users attention
        hwc.log("Implementation required for hwc.processDataMessage", "ERROR", true);
        throw new Error("Implementation required for either global processDataMessage or hwc.processDataMessage");
    }
}

/**
 * @deprecated Deprecated since version 2.2 - use hwc.processDataMessage(incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType)
 */

function processWorkflowMessage(incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType) {
    return processDataMessage(incomingDataMessageValue, noUI, loading, fromActivationFlow, dataType);
}
/**
 * This function is invoked by the container when there is a native error to report.
 * Use {@link hwc.setReportErrorFromNative} to set the callback function this function will call.
 * This function is not intended to be called except by the container.
 * @private
 * @param {string} errString The string contains error message
 */

function reportErrorFromNative(errString) {
    var reportErrorCallback = hwc.getReportErrorFromNativeCallback();
    if (typeof reportErrorCallback === "function") {
        reportErrorCallback(errString);
    }
}

/**
* Container API
*/

(function(hwc, window, undefined) {

/**
 * A number representing the logging level. The logging level must be an integer from the range \([1..4]\)
 * with the higher numbers being more verbose.
 */

* @type {number}
*/

var requestedLoggingLevel,
149         /**
150          * A callback function used when `{@link hwc.log}` is
151          * invoked with `true` for the `notifyUser` parameter.
152          * This callback should notify the user of the log message
153          * in an appropriate manner.
154          *
155          * @type {anonymous.alertDialogCallbackFunction}
156          */
157         requestedAlertDialogCallback,
158         /**
159          * A callback function used when there is a native error to
160          * report
161          *
162          * @type {anonymous.errorCallbackFunction}
163          */
164         reportErrorFromNativeCallback;
165          /**
166          * This object contains constants representing the status
167          * of the hybrid app.
168          *
169          * @constant
170          * @type {number}
171          * @member of hwc.STATUS
172          * @public
173          */
174         hwc.STATUS.CLOSED = 1;
175         /**
176         */
177         */
A constant indicating the hybrid app is running.

@constant
@type {number}
@member of hwc.STATUS
@public

```javascript
hwc.STATUS.RUNNING = 2;
```

A status representing the hybrid app, default is running.

```javascript
var status = hwc.STATUS.RUNNING;
```

This function sets the callback used by hwc.log when it is required to notify the user of a log item.

```javascript
hwc.setLoggingAlertDialog = function( newAlertDialogCallback )
```

*param {anonymous.alertDialogCallbackFunction} newAlertDialogCallback The alert dialog to use.

```javascript
* @example
customLogAlert = function( message )
{
    alert( "New log message: " + message );
}
```

```javascript
hwc.setLoggingAlertDialog( customLogAlert );
```

@memberOf hwc
@public
requestedAlertDialogCallback = newAlertDialogCallback;
}

/**
 * This function gets the callback used by hwc.log when it
 * is required to notify the user of a log item.
 *
 * @memberOf hwc
 * @public
 * @returns {anonymous.alertDialogCallbackFunction} The
 * alert dialog callback function.
 */

hwc.getLoggingAlertDialog = function()
{
    return requestedAlertDialogCallback;
}

/**
 * This function sets the logging level. The logging level
 * set with this function only persists as long as this javascript
 * context does.
 *
 * When the hybrid app is closed, the value set with this
 * function is lost.
 *
 * @memberOf hwc
 * @public
 * @param {number} newLoggingLevel The number representing
 * the new logging level.
 *
 * Must be an integer in the range [1..4]. The higher
 * numbers represent more verbose logging levels
 *
 * from 1 for ERROR level logging up to 4 for DEBUG level
 * logging.
 *
 * @example
 * // Set the logging level to debug.
 * hwc.setLoggingCurrentLevel( 4 );

hwc.setLoggingCurrentLevel = function( newLoggingLevel ) {
    requestedLoggingLevel = newLoggingLevel;
};

/**
 * This function gets the logging level.
 *
 * @memberOf hwc
 * @public
 * @returns {number} A number representing the logging level. Will be an integer in the range [1..4].
 * The higher numbers represent more verbose logging levels from 1 for ERROR level logging up to 4 for DEBUG level logging.
 *
 * @example
 * // Get the logging level
 * var loggingLevel = hwc.getLoggingCurrentLevel();
 */

hwc.getLoggingCurrentLevel = function() {
    var logLevel;
    if (requestedLoggingLevel === undefined) {
        logLevel = hwc.getQueryVariable("loglevel");
        requestedLoggingLevel = logLevel ? parseInt(logLevel, 10) : 1;
    }
    return requestedLoggingLevel;
};
Calling this function will replace any callback that had been set previously.

* @memberOf hwc
* @public
* @param {function} callbackToSet The callback function.
* @example
* var errorCallback = function( errorString )
* {
*   alert( "There was a native error: " +
*          errorString );
* }
* hwc.setReportErrorFromNativeCallback( errorCallback );

```javascript
hwc.setReportErrorFromNativeCallback = function( callbackToSet )
{
    reportErrorFromNativeCallback = callbackToSet;
};
```

This function returns the callback function that will be called by {link reportErrorFromNative}.

This function is not intended to be called by any function but {link reportErrorFromNative}.

* @private
* @returns {function} The callback function.
* /

```javascript
hwc.getReportErrorFromNativeCallback = function()
{
    return reportErrorFromNativeCallback;
};
```

/**
 * This function returns the callback function that will be called by {link reportErrorFromNative}.
 * This function is not intended to be called by any function but {link reportErrorFromNative}.
 */
* This function looks in the query string on the URL for the value corresponding to the given name.

* @memberOf hwc

* @public

* @param {string} variable The name of the variable in the URL to retrieve the value for.

* @returns {string} The value corresponding to the given name.

* @example

* // Get the pageToShow variable from the URL query string

* var pageToShow = hwc.getQueryVariable( "pageToShow" );

* hwc.getQueryVariable = function(variable) {

  var query, vars, i, pair;

  query = window.location.search.substring(1);

  vars = query.split("&");

  for (i = 0; i < vars.length; i++) {

    pair = vars[i].split("=");

    if (pair[0] === variable) {

      return unescape(pair[1]);

    }

  }

};

/**

* This object contains constants representing the different types of public native error codes.

* Error codes larger than 500 are reserved for server communication errors which may occur as the result of online requests and/or attachment downloads.

*
```javascript
hwc.NativeErrorCodes = {};

/**
 * A constant indicating there was an unknown error.
 */

hwc.NativeErrorCodes.UNKNOWN_ERROR = 1;

/**
 * A constant indicating the attachment has not been downloaded.
 */

hwc.NativeErrorCodes.ATTACHMENT_NOT_DOWNLOADED = 100;

/**
 * A constant indicating there was an unknown MIME type.
 */

hwc.NativeErrorCodes.UNKNOWN_MIME_TYPE = 101;

/**
 * A constant indicating there was a filename without an extension.
 */
```
hwc.NativeErrorCodes.FILENAME_NO_EXTENSION = 102;

/**
 * A constant indicating a required parameter was not available.
 */

hwc.NativeErrorCodes.REQUIRED_PARAMETER_NOT_AVAILABLE = 103;

/**
 * A constant indicating there was no certificate selected by the user.
 */

hwc.NativeErrorCodes.CERTIFICATE_NOT_SELECTED = 104;

/**
 * A constant indicating the attachment type is not supported.
 */

hwc.NativeErrorCodes.UNSUPPORTED_ATTACHMENT_TYPE = 105;
/**
 * A constant indicating there was an SSO certificate manager exception.
 */

*hwc.NativeErrorCodes.SSOCERT_EXCEPTION = 106;

/**
 * A constant indicating a failure to save a credential.
 */

*hwc.NativeErrorCodes.FAIL_TO_SAVE_CREDENTIAL = 107;

/**
 * A constant indicating a failure to save a certificate.
 */

*hwc.NativeErrorCodes.FAIL_TO_SAVE_CERTIFICATE = 108;

/**
 * A constant indicating the device is not connected.
 */

Develop Hybrid Apps Using Third-party Web Frameworks
hwc.NativeErrorCodes.DEVICE_NOT_CONNECTED = 109;

/**
 * A constant indicating the response it too large for a javascript variable.
 */

hwc.NativeErrorCodes.RESPONSE_TOO_LARGE = 110;

/**
 * A constant indicating that opening the URL failed.
 */

hwc.NativeErrorCodes.NAVIGATION_ERROR = 111;

/**
 * A constant indicating an invalid common name was passed while requesting a certificate from Afaria.
 */

hwc.NativeErrorCodes.INVALID_COMMON_NAME = 112;

/**
 * A utility function for use in generating a GUID
 */
function S4() {
    return (((1 + Math.random()) * 0x10000) | 0).toString(16).substring(1);
}

hwc.guid = function() {
    return (S4() + S4() + "-" + S4() + "-" + S4() + "-" + S4() + "-" + S4() + "-" + S4() + S4() + S4());
};

hwc.getXMLHTTPRequest = function getXMLHTTPRequest() {
Avoid this endless loop:

```javascript
hwc.log()
```
hwc.setScreenTitle_CONT = function(screenTitle) {
    hwc.traceEnteringMethod("hwc.setScreenTitle_CONT");
    try {
        if (hwc.isWindows()) {
            document.title = screenTitle;
        }
        else {
            if (hwc.iOS() || hwc.isAndroid()) {
                hwc.getDataFromContainer("setscreentitle", "&title=" + encodeURIComponent(screenTitle));
            }
            else {
                hwc postDataToContainer("setscreentitle", "&title=" + encodeURIComponent(screenTitle));
            }
        }
    } finally {
        hwc.traceLeavingMethod("hwc.setScreenTitle_CONT");
    }
};

/**
 * This class represents a collection of menu items.
 */

Develop Hybrid Apps Using Third-party Web Frameworks
* // This is the function we'll use as a callback for the first menu item.
* var callback = function()
* {
*   alert( "You clicked the first menu item!" );
* }

* // This is the function we'll use as a callback for the second menu item.
* var callback2 = function()
* {
*   alert( "You clicked the second menu item!" );
* }

* // This function creates and adds a menu item collection.
* var addMenuItems = function()
* {
*   var menuItemCollection = new hwc.MenuItemCollection();
*   menuItemCollection.addMenuItem("menu item 1", "callback()");
*   menuItemCollection.addMenuItem("menu item 2", "callback2()");
*   hwc.addMenuItemCollection( menuItemCollection );
* }

hwc.MenuItemCollection = function() {
  this.menuItems = [];
  this.subMenuName = null;
  this.okAction = null;
};

/**
This function adds a menu item to the collection.

* @memberOf hwc.MenuItemCollection
* @public
* @param {string} title The display text for the menu item.
* @param {anonymous.genericCallbackFunction} callback The function to call when the menu item is clicked.
* @param {boolean} [isDefault] Determines if the menu item is selected by default on BlackBerry.

If more than one menu item is added to the same collection with true for this parameter, the last menu item added with true for this parameter will be selected by default on BlackBerry.

* @example

```javascript
var callbackFunctionName = function()
{
    alert( "Menu item clicked!" );
}

var menuItemCollection = new hwc.MenuItemCollection();

menuItemCollection.addMenuItem("menu item name", "callbackFunctionName()", true);
```

```javascript
hwc.MenuItemCollection.prototype.addMenuItem = function(title, callback, isDefault) {
    hwc.traceEnteringMethod("hwc.MenuItemCollection.addMenuItem");
    try {
        this.menuItems.push( { "name" : title, "action" : callback, "default" : isDefault ? "true" : "false" } );
    } finally {
        hwc.traceLeavingMethod("hwc.MenuItemCollection.addMenuItem");
    }
};
```
/**
 * This function sets the sub menu name to use on Windows Mobile.
 */

@memberOf hwc.MenuItemCollection
@public
@param {string} name The sub menu name to use.
@example
var callbackFunctionName = function()
{
    alert( "Menu item clicked!" );
}

var menuItemCollection = new hwc.MenuItemCollection();
menuItemCollection.setSubMenuName( "Custom Menu" );
menuItemCollection.addMenuItem("menu item name", "callbackFunctionName()");

hwc.MenuItemCollection.prototype.setSubMenuName = function(name) {
    this.subMenuName = name;
};

/**
 * This function sets the OK action to use on WM.
 */

@memberOf hwc.MenuItemCollection
@public
@param {anonymous.genericCallbackFunction} callback The function to call when the OK button is pressed.
@example
var callbackFunctionName = function()
{
    alert( "Menu item clicked!" );
}
617 * )
618 * var okActionFunction = function()
619 * {
620 *   alert( "A OKAY!" );
621 * }
622 * var menuItemCollection = new hwc.MenuItemCollection();
623 * menuItemCollection.setOKAction( "okActionFunction()" );
624 * menuItemCollection.addMenuItem("menu item name",
625 * "callbackFunctionName()" );
626 */
627 hwc.MenuItemCollection.prototype.setOKAction =
628 function(callback) {
629   this.okAction = callback;
630 }
631/**
632 * This function converts the menu item collection to a JSON
633 * string. This function
634 * is used as a helper for {link hwc.addMenuItemCollection}.
635 *
636 * @memberOf hwc.MenuItemCollection
637 * @public
638 * @returns {string} The JSON string representing this menu
639 * item collection.
640 */
641 var callbackFunctionName = function()
642 * {
643 *   alert( "Menu item clicked!" );
644 * }
645 * var menuItemCollection = new hwc.MenuItemCollection();
646 * var jsonMenuItemsCollection =
647 * menuItemCollection.stringify();
648 */
hwc.MenuItemCollection.prototype.stringify = function()
{
    return JSON.stringify({
        "menuitems" : this.menuItems,
        "submenuname" : this.subMenuName,
        "OK" : this.okAction
    });
}

/**
 * This function adds a menu item collection to the menu items for the screen.
 *
 * @memberOf hwc
 * @public
 * @param {hwc.MenuItemCollection} collection The collection of menu items to add to the screen.
 *
 * @example
 * var callbackFunctionName = function()
 * {
 *     alert( "Menu item clicked!" );
 * }
 * var menuItemCollection = new hwc.MenuItemCollection();
 * menuItemCollection.addMenuItems("menu item name",
 * "callbackFunctionName()");
 * hwc.addMenuItemCollection( menuItemCollection );
 */

hwc.addMenuItemCollection = function(collection) {
    hwc.traceEnteringMethod("hwc.addMenuItemCollection");
    try {
        if (isBlackBerry() || isWindowsMobile() || isAndroid()) {
            var request = "menuitems=" +
                           encodeURIComponent(collection.stringify());

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps
hwc.postDataToContainer("addallmenuitems", request);
}
} finally {

hwc.traceLeavingMethod("hwc.addMenuItemCollection");
}

/**
 * Allows the user to add a menuitem with the specified name and with the specified
 * callback, which will be invoked when the menuitem is clicked. This function should
 * only be used in hybrid apps generated with the Unwired Workspace designer.
 *
 * @memberOf hwc
 * @private
 *
 * @param {string} menuItemName The specified menuitem name.
 * @param {string} functionName The string representing the call to the {@link anonymous.genericCallbackFunction} callback function.
 * @param {string} subMenuName The specific sub-menu name for Windows Mobile.
 * @param {string} screenToShow The screen about to be shown.
 * @param {string} [menuItemKey] The menuitem's key.
 * @example
 * var callbackFunction = function()
 * {
 *    alert( "Menu Item Clicked!" );
 * }
 * hwc.addMenuItem_CONT( "Custom Menu Item", "callbackFunction()", "Custom Sub Menu", "Start" );
hwc.addMenuItem_CONT = function(menuItemName, functionName, subMenuName, screenToShow, menuItemKey) {
    var div, menuStr, idxOfMenuItemName, comma, request;
    hwc.traceEnteringMethod("hwc.addMenuItem_CONT");
    try {
        //first add the item to sup_menuitems
        div = document.getElementById(screenToShow + "ScreenDiv");
        menuStr = div.getAttribute("sup_menuitems");
        idxOfMenuItemName = menuStr.indexOf(menuItemName);
        if (idxOfMenuItemName !== -1) {
            return;
        }
        return;
    } catch (e) { }
    comma = (menuStr.length > 0) ? "," : "";
    menuStr = menuStr + comma + menuItemName + "," + menuItemKey;
    try {
        div.setAttribute("sup_menuitems", menuStr); //has no affect on Windows Mobile
    }
    catch (e) { }
    request = "menuitemname=" + encodeURIComponent(menuItemName);
    request += ("&onmenuclick=" + encodeURIComponent(functionName) + "()");
    if (hwc.isWindowsMobile()) {
        request += "&submenuname=";
    }
    if (subMenuName) {
        request += encodeURIComponent(subMenuName);
    }
    else {

if (resources) {
    request += encodeURIComponent(resources.getString("MENU"));
}
else {
    request += "Menu";
}

hwc.postDataToContainer("addMenuItem", request);

else if (hwc.isAndroid() || hwc.isBlackBerry()) {
    hwc.postDataToContainer("addMenuItem", request);
}

} finally {
    hwc.traceLeavingMethod("hwc.addMenuItem_CONT");
}

/**
 * This function removes all menu items that were added by the hybrid app.
 * Note: This API does not support on iOS platform.
 *
 * @memberOf hwc
 * @public
 * @example
 * hwc.removeAllMenuItems();
 */

hwc.removeAllMenuItems = function() {
    hwc.traceEnteringMethod("hwc.removeAllMenuItems_CONT");
    try {

if (hwc.isAndroid() || hwc.isWindowsMobile() ||
hwc.isBlackBerry() ) {
    hwc.getDataFromContainer("removeallmenuitems");
}
} finally {
    hwc.traceLeavingMethod("hwc.removeAllMenuItems");
}
};

/**
 * This function sets the activation required state of this
 * hybrid app to true. After calling this
 * function, the current hybrid app will need to be
 * activated.
 * @memberOf hwc
 * @public
 * @example
 * hwc.activationRequired();
 */

hwc.activationRequired = function() {
    hwc.getDataFromContainer("requiresactivation");
} finally {
    hwc.traceLeavingMethod("hwc.activationRequired");
};

/**
 * This function sets the activation required state for the
 * current hybrid app to false. After calling this
 * function, the current hybrid app will not need to be
 * activated.
 */
hwc.markAsActivated = function() {
    hwc.traceEnteringMethod("hwc.markAsActivated");
    try {
        hwc.getDataFromContainer("markasactivated");
    } finally {
        hwc.traceLeavingMethod("hwc.markAsActivated");
    }
};

/**
 * Allows the user to set the processed state to true for the current message.
 */

hwc.markAsProcessed = function() {
    hwc.traceEnteringMethod("hwc.markAsProcessed");
    try {
        hwc.getDataFromContainer("markasprocessed");
    } finally {
        hwc.traceLeavingMethod("hwc.markAsProcessed");
    }
};

/**
Allows the user to set the credentials to the expired state for the current hybrid app.

@memberOf hwc
@public
@example
hwc.expireCredentials();
*/

hwc.expireCredentials = function() {
    hwc.traceEnteringMethod("hwc.expireCredentials");
    try {
        hwc.getDataFromContainer("expirecredentials");
    } finally {
        hwc.traceLeavingMethod("hwc.expireCredentials");
    }
}

This function clears the contents of the on-device request result cache for the current hybrid app.

@memberOf hwc
@public
@example
hwc.clearCache();
*/

hwc.clearCache = function() {
    hwc.traceEnteringMethod("hwc.clearCache");
    try {
        hwc.getDataFromContainer("clearrequestcache");
    } finally {
        hwc.traceLeavingMethod("hwc.clearCache");
    }
};
/**
 * This function clears an item from the contents of the on-device request result cache for the current hybrid app.
 *
 * @memberOf hwc
 * @public
 * @param {string} cachekey The key for the cache item to be removed. This is the same key that was passed to hwc.doOnlineRequest.
 *
 * @example
 * // The cache key is set when calling hwc.doOnlineRequest_CONT
 * hwc.doOnlineRequest( ., ., ., ., ., ., ., cacheKey, ., .);
 *
 * // At some later point if we want to clear the cache for the above request, we use the following code:
 * hwc.clearCacheItem( cacheKey );
 */

hwc.clearCacheItem = function( cachekey ) {
    var request;
    hwc.traceEnteringMethod("hwc.clearCacheItem");
    try {
        request = "cachekey=" + encodeURIComponent(cachekey);
        hwc.postDataToContainer("clearrequestcacheitem", request);
    } finally {
        hwc.traceLeavingMethod("hwc.clearCacheItem");
    }
};
* Allows the user to log a message to the device trace log which can be remotely retrieved from the server.

* Whether the message actually gets logged will depend on how the log level that the administrator has selected for this device user compares with the log level of this message.

* The logging level and alert dialog callback can be set with `{@link hwc.setLoggingCurrentLevel} and `{@link hwc.setLoggingAlertDialog}.

* @memberOf hwc

* @public

* @param {string} sMsg The message to be logged.

* @param {string} eLevel The error level for this message. This parameter must be one of: "ERROR", "WARN", "INFO" or "DEBUG".

* @param {boolean} notifyUser Whether the logging alert callback will be invoked. This parameter is independent of the logging level (the logging alert callback will always be invoked if this is true, and never if this is false).

* @example

```javascript
var logAlert = function( message )
{
    alert( "New log message: " + message );
}

hwc.setLoggingAlertDialog( logAlert );
hwc.setLoggingCurrentLevel( 3 );
// The following will be logged, and the logging alert dialog will be invoked.

hwc.log( "info message notify", "INFO", true );
// The following will be logged, but the logging alert dialog will not be invoked.

hwc.log( "info message", "INFO", false );
// The following will not be logged, but the logging alert dialog will be invoked.

hwc.log( "debug message notify", "DEBUG", true );
// The following will not be logged, and the logging alert dialog will not be invoked.
```
hwc.log( "debug message", "DEBUG", false );

hwc.log = function log(sMsg, eLevel, notifyUser) {
    var msgLogLevel;
    if( !sMsg ) {
        return;
    }
    if (notifyUser && hwc.getLoggingAlertDialog()) {
        (hwc.getLoggingAlertDialog())(sMsg);
    }
    switch (eLevel) {
        case "ERROR":
            msgLogLevel = 1;
            break;
        case "WARN":
            msgLogLevel = 2;
            break;
        case "INFO":
            msgLogLevel = 3;
            break;
        case "DEBUG":
            msgLogLevel = 4;
            break;
        default:
            msgLogLevel = 1;
    }
    if((sMsg === "") || (msgLogLevel > hwc.getLoggingCurrentLevel()) || (hwc.isWindows())) {
        return;
    }
}
if (hwc.isAndroid()) {
    _HWC.log(sMsg, msgLogLevel);
} else {
    hwc.postDataToContainer("logtoworkflow", "loglevel=" + msgLogLevel + ",logmessage=" + encodeURIComponent(sMsg));
}

/**
 * This function opens a form on the device that allows the user to specify the credentials for the use of certificate-based authentication. If the user picks a certificate, then that certificate is saved in the credentials cache.

 * @memberOf hwc
 * @public
 * @example
 * hwc.showCertificatePicker();
 */

hwc.showCertificatePicker = function() {
    hwc.traceEnteringMethod("hwc.showCertificatePicker");
    try {
        hwc.getDataFromContainer("showcertpicker");
    } finally {
        hwc.traceLeavingMethod("hwc.showCertificatePicker");
    }
};

/**
* This function saves login credentials from a certificate to the credential cache.

* The common name is used for the username and the signed certificate is used for the password.

* @memberOf hwc
* @public
* @param {object} certificate The values certificate.subjectCN and certificate.signedCertificate must be defined.

* @example
* var certInfo = {};
* certInfo.subjectCN = "sampleCommonName";
* certInfo.signedCertificate = "samplePassword";
* hwc.saveLoginCertificate( certInfo );
* /

hwc.saveLoginCertificate = function(certificate) {
  hwc.traceEnteringMethod("hwc.saveLoginCertificate");
  try {
    hwc.saveLoginCredentials(certificate.subjectCN, certificate.signedCertificate, true);
  } finally {
    hwc.traceLeavingMethod("hwc.saveLoginCertificate");
  }
};

/**
* This function saves login credentials to the credential cache.
* @memberOf hwc
* @public
* @param {string} userName The user name to save
* @param {string} password The password to save
hwc.saveLoginCredentials = function(userName, password) {
    var requestData;
    hwc.traceEnteringMethod("hwc.saveLoginCredentials");
    try {
        requestData = "supusername=" + encodeURIComponent(userName) + "&suppassword=" + encodeURIComponent(password);

        if (hwc.isAndroid()) {
            _HWC.saveCredentials( userName, password );
        } else {
            hwc.postDataToContainer("savecredential", requestData);
        }
    } finally {
        hwc.traceLeavingMethod("hwc.saveLoginCredentials");
    }
}

/**
* This function opens the supplied URL in a browser. The browser opens on top of the hybrid app - the
* context of the hybrid app is undisturbed.
* @memberOf hwc
* @public
* @param {string} url The URL to be shown in a browser.
* @example
* hwc.showUrlInBrowser( "http://www.google.com" );
hwc.showUrlInBrowser = function showUrlInBrowser(url)
{
    var idxOfColon;
    hwc.traceEnteringMethod("hwc.showUrlInBrowser");
    try {
        url = hwc.trimSpaces(url, true);
        idxOfColon = url.indexOf(":");
        if (idxOfColon === -1 || (idxOfColon > 7)) {
            url = "http://" + url;
        }
        if (hwc.isWindowsMobile() || hwc.isAndroid() || hwc.isIOS() || hwc.isBlackBerry()) {
            hwc.getDataFromContainer("showInBrowser", "+url=" + encodeURIComponent(url));
        } else {
            window.open(url);
        }
    } finally {
        hwc.traceLeavingMethod("hwc.showUrlInBrowser");
    }
};

/**
 * Shows the given file contents in a content-appropriate way. The type of the content is
 * supplied by either the MIME type or the filename, at least one of which must be supplied.
 * The content itself should be presented as a base64-encoded string. Not all file types may
 * be supported on all platforms.
 * @memberOf hwc
* @public
* @param {string} contents The base-64 encoded version of the binary content of the attachment to be displayed.
* @param {string} mimeType The MIME type of the file.
* @param {string} fileName The name of the file.
* @param {anonymous.genericCallbackFunction} waitDialogCallbackString The callback function used to close a wait dialog once the attachment is done opening.

* is done opening.
* @example
* var openAttachmentBase64StringPng = function()
* {
*   // How you want get the base 64 encoding of the file is up to you. This string represents a small png image.
*   var data = "iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAAAXNSR0IArs4c6QAAA
RnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAA0SURBVFhH7dAxEQAACAM3CAT6eVwZKkh8/dsmc7n6jN+bQCIECBAgAAAgACBBb3SkJeQ67u1AAAAE1FTkSuQmCC";
*   hwc.showProgressDialog();
*   // Don't have to pass the filename because we are passing the MIME type.
*   hwc.showAttachmentContents_CONT( data, "image/png", null, "hwc.hideProgressDialog()" );
* }
* @example
* var openAttachmentBase64StringTxt = function()
* {
*   // How you want get the base 64 encoding of the file is up to you. This string represents a short text file.
*   var data = "VGhpcyBpcyBwYXJ0IG9mIGEgaHlicmlkIGFwcC4=";
*   // Don't have to pass the MIME type because we are passing the filename.
*   hwc.showAttachmentContents_CONT( data, null, "attach.txt" );
* }

Develop Hybrid Apps Using Third-party Web Frameworks
hwc.showAttachmentContents_CONT = function(contents, mimeType, fileName, waitDialogCallbackString) {
    var request;

    hwc.traceEnteringMethod("hwc.showAttachmentContents_CONT");
    try {
        request = "callback=" + waitDialogCallbackString;

        if (hwc.isWindowsMobile()) {
            contents = contents.replace(/=/g, "~");
            request += ";Attachmentdata=" + contents;
        } else {
            request += ";Attachmentdata=" + encodeURIComponent(contents);
        }

        if (mimeType) {
            request += ";mimetype=" + encodeURIComponent(mimeType);
        }

        if (fileName) {
            request += ";filename=" + encodeURIComponent(fileName);
        }

        hwc.postDataToContainer("showattachment", request);
    } finally {
        hwc.traceLeavingMethod("hwc.showAttachmentContents_CONT");
    }
}

* Shows the given file contents in a content-appropriate way. The type of the content is
* supplied by either the MIME type or the filename, at least one of which must be supplied.

* The content itself will be a unique key supplied earlier to a call to doAttachmentDownload.

* @memberOf hwc

* @public

* @param {string} uniqueKey The unique key for the attachment.

* @param {string} mimeType The MIME type of the file.

* @param {string} fileName The name of the file.

* @param {string} waitDialogCallbackString string with the value for the 'callback=' parameter.

*/

hwc.showAttachmentFromCache_CONT = function(uniqueKey, mimeType, fileName, waitDialogCallbackString) {
    var request;

    hwc.traceEnteringMethod("hwc.showAttachmentFromCache_CONT");

    try {
        request = "callback=" + waitDialogCallbackString;

        request += ";uniquekey=" + encodeURIComponent(uniqueKey);

        if (mimeType) {
            request += ";mimetype=" + encodeURIComponent(mimeType);
        }

        if (fileName) {
            request += ";filename=" + encodeURIComponent(fileName);
        }

        hwc postDataToContainer("showattachment", request);
    } finally {
        hwc.traceLeavingMethod("hwc.showAttachmentFromCache_CONT");
    }
hwc.showLocalAttachment = function showLocalAttachment(key) {
    hwc.traceEnteringMethod("hwc.showLocalAttachment");
    try {
        if (hwc.isWindowsMobile() || hwc.isAndroid() || hwc.isIOS()) {
            hwc.getDataFromContainer("showlocalattachment", "&key=" + encodeURIComponent(key));
        } else if (hwc.isBlackBerry()) {
            if (key.indexOf("file://") > -1) {
                window.location = key;
            } else {
                window.location = "http://localhost/" + key;
            }
        } else {
            window.open(key);
        }
    } finally {
    }
hwc.traceLeavingMethod("hwc.showLocalAttachment");

/**
 * Internal function to allow the user to cause an operation/object query to be invoked. This function should probably only be used by designer generated javascript.
 *
 * @memberOf hwc
 * @private
 *
 * @param {string} credInfo Credential info in the format "supusername=usernameValue&suppassword=passwordValue"
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 * @param {boolean} hasFileMessageValue Whether the data message to send has a file message value. This parameter should be obtained by calling getHasFileMessageValue
 * @param {number} timeout Specifies the time, in seconds, to wait before giving up waiting for a response.
 * @param {string} cacheTimeout Specifies the time, in seconds, since the last invocation with the same input parameter values to use the same response as previously retrieved without making a new call to the server. If this parameter is NEVER, the cache content will never expire.
 * @param {string} errorMessage Specifies the string to display if an online request fails.
 * @param {anonymous.errorCallbackFunction} errorCallback Name of the function to be called if an online request fails. If this parameter is null, 'reportRMIError' will be used.
@param {string} cacheKey String used as the key for this request in the on-device request result cache.

@param {string} cachePolicy Specifies cache lookup policy used by container. If this parameter is 'serverfirst' (ignoring case) then the cache policy

* used for this online request will be to check the server before the cache. If this parameter is any other value then a cache first policy will be used.

* If this parameter is absent and cache is enabled, the container uses default cache lookup policy to get data from cache if it is not expired.

* @param {boolean} asynchronous Specifies whether container will make the request in synchronous or asynchronous mode.

* If this parameter is absent, the container makes the request to the server in synchronous mode.

*/

hwc.doOnlineRequest_CONT = function( credInfo,
serializeDataMessageToSend,
hasFileMessageValue,
timeout, cacheTimeout,
errorMessage, errorCallback,
cacheKey, cachePolicy,
asynchronous) {

hwc.traceEnteringMethod("hwc.doOnlineRequest_CONT");

try {
  var request, xmlhttp, response, encodedMessage, url, funcCall, responseDataType;

  request = "xmlWorkflowMessage=" +
  encodeURIComponent(serializeDataMessageToSend);

  if (credInfo) {
    request += ("&" + credInfo);
  }

  request += ("&cachekey=" +
  encodeURIComponent(cacheKey));
if (timeout) {
    request += ("&rmitimeout=" + timeout);
}
if (cacheTimeout) {
    request += ("&RequestExpiry=" + cacheTimeout);
}
if (hasFileMessageValue) {
    request += ("&parse=true");
}
if (errorMessage) {
    if( hwc.isBlackBerry() ) {
        encodedMessage = encodeURIComponent(escape(errorMessage));
    } else {
        // This is a temporary fix for a bug in the
        // container that calls
        // encodeURIComponent on the whole query string
        // for Android. See
        // IR 676161-2.
        encodedMessage = encodeURIComponent(errorMessage);
    } else {
        // This is a temporary fix for a bug in the
        // container that calls
        request += ("&onErrorMsg=" + encodedMessage);
    }
}
if (!errorCallback) {
    errorCallback = "hwc.reportRMIError";
}
if (cachePolicy) {
    request += ("&cachePolicy=" + cachePolicy);
}
if (asynchronous) {
    request += ("&asynchronous=" + asynchronous);
}
request += ("&onErrorCallback=" + errorCallback);
if (hwc.isWindowsMobile() || hwc.isWindows()) {
    //make xmlhttp request to load the rmi response from server
    xmlhttp = hwc.getXMLHTTPRequest();

    if (hwc.isWindowsMobile()) {
        xmlhttp.open("POST", "/sup.amp?querytype=rmi&" + hwc.versionURLParam, true);

        xmlhttp.onreadystatechange = function() {
            if (xmlhttp.readyState === 4) {
                if (xmlhttp.status === 200 ||
                    xmlhttp.status === 0) {
                    response = xmlhttp.responseText;
                    var responseDataType = xmlhttp.getResponseHeader("OnlineRequest-Response-Data-Type");
                    processDataMessage(response, null, null, null, responseDataType);
                }
            }
        }
    }
    else { // hwc.isWindows()
        xmlhttp.open("POST", "rmi.xml", false);
        xmlhttp.send(request);
    }
}

try {
    xmlhttp.send(request);
} catch (excep1) {
    hwc.log("Error: Unable to retrieve the message from the server", "ERROR", true);
}
else { // hwc.isWindows()
if (xmlhttp.status === 200 || xmlhttp.status === 0) {
    response = xmlhttp.responseText;
    processDataMessage(response);
}
else {
    hwc.log("Error: Unable to retrieve the message from the server", "ERROR", true);
}

else if (hwc.isAndroid()) {
    url = 'http://localhost/sup.amp?querytype=rmi&' + hwc.versionURLParam;
    funcCall = "_HWC.postData('" + url + '", "" + request + ")"; // method processDataMessage invoked by native container.
    setTimeout(funcCall, 5);
}
else { //BB and iPhone
    xmlhttp = hwc.getXMLHTTPRequest();
    xmlhttp.open("POST", "http://localhost/sup.amp?querytype=rmi&" + hwc.versionURLParam, true);
    if (hwc.isBlackBerry()) {
        xmlhttp.onreadystatechange = function() {
            if (xmlhttp.readyState === 4) {
                if (xmlhttp.status === 200) {
                    // do something
                }
            }
        }
    }
    if (hwc.isBlackBerry()) {
        xmlhttp.onreadystatechange = function() {
            if (xmlhttp.readyState === 4) {
                if (xmlhttp.status === 200) {
                    // do something
                }
            }
        }
    }
}
response = xmlhttp.responseText;

responseDataType = xmlhttp.getResponseHeader("OnlineRequest-Response-Data-Type");

processDataMessage(response, null, null, null, responseDataType);

}

}

}

try {
    xmlhttp.send(request);
}

catch (excep2) {
    hwc.log("Error: Unable to retrieve the message from the server", "ERROR", true);
}

} finally {
    hwc.traceLeavingMethod("hwc.doOnlineRequest_CONT");
}

/**
 * Allows the user to cause an operation/object query to be invoked. This function should probably only be used by
 * designer generated javascript.
 */

* @memberOf hwc

* @private

* @param {string} credInfo Credential info in the format "supusername=usernameValue&suppassword=passwordValue"
* @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling `serializeToString` on the result from `hwc.getMessageValueCollectionForOnlineRequest`.

* @param {string} attachmentKey The specified key of the result will not be returned in the data message but will instead be stored on the device for later access via `{@link hwc.showAttachmentFromCache_CONT}`.

* @param {string} requestGUID Represents a unique key that can be used to store/access the cached key value from the request results.

* @param {callback function} downloadCompleteCallback A function that will be invoked when the attachment has been downloaded to the device and is ready to be accessed.

* /

`hwc.doAttachmentDownload_CONT = function(credInfo, serializeDataMessageToSend, attachmentKey, requestGUID, downloadCompleteCallback) {
  hwc.traceEnteringMethod("hwc.doAttachmentDownload_CONT");
  try {
    var request, xmlhttp;
    request = "xmlWorkflowMessage=" + encodeURIComponent(serializeDataMessageToSend);
    if (credInfo) {
      request += ("&" + credInfo);
    }
    request += ("&attachmentkey=" + attachmentKey);
    request += ("&uniquekey=" + requestGUID);
    request += ("&ondownloadcomplete=" + downloadCompleteCallback);
    if (hwc.isWindowsMobile() || hwc.isWindows()) {
      xmlhttp = hwc.getXMLHTTPRequest();
      xmlhttp.open("POST", "/sup.amp?querytype=downloadattachment&" + hwc.versionURLParam, true );
    }
  }
};`
Develop Hybrid Apps Using Third-party Web Frameworks

    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState === 4) {
            if (xmlhttp.status === 200) {
                window[downloadCompleteCallback].call(this,
                decodeURIComponent(requestGUID), xmlhttp.responseText);
            }
        }
    }

    try {
        xmlhttp.send(request);
    }
    catch (e3) {}

    else if (hwc.isAndroid()) {
        hwc.postDataToContainer("downloadattachment", request);
    }

    else {
        xmlhttp = hwc.getXMLHTTPRequest();
        xmlhttp.open("POST", "http://localhost/sup.amp?querytype=downloadattachment&" + hwc.versionURLParam, true);
        if (hwc.isBlackBerry()) {
            xmlhttp.onreadystatechange = function() {
                if (xmlhttp.readyState === 4) {
                    if (xmlhttp.status === 200) {
                        window[downloadCompleteCallback].call(this,
                        decodeURIComponent(requestGUID), xmlhttp.responseText);
                    }
                }
            }
        }
    }
xmlhttp.send(request);

try {
    catch (e1) {}
} finally {
    hwc.traceLeavingMethod("hwc.doAttachmentDownload_CONT");
}

/**
 * Allows the user to cause an operation/object query to be invoked. Will close the hybrid app application when finished. This function should probably only be used by designer generated javascript.
 *
 * @memberOf hwc
 * @private
 *
 * @param {string} credInfo Credential info in the format "supusername=usernameValue&suppassword=passwordValue"
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 * @param {boolean} hasFileMessageValue Whether the data message to send has a file message value. This parameter should be obtained by calling getHasFileMessageValue
 *
 * @param {function} hwc.doSubmitWorkflow.CONT = function(credInfo, serializeDataMessageToSend, hasFileMessageValue) {
    hwc.traceEnteringMethod("hwc.doSubmitWorkflow_CONT");
    try {
        
        
        
        
        } finally {
            
            
            
            
        }
        
    }
*/
var request = "xmlWorkflowMessage=" +
    encodeURIComponent(serializeDataMessageToSend);

if (credInfo) {
    request += ("&" + credInfo);
}

if (hasFileMessageValue) {
    request += ("&parse=true");
}

hwc.postDataToContainer("submit", request);

hwc.traceLeavingMethod("hwc.doSubmitWorkflow_CONT");

/**
 * Internal function to allow the user to cause an operation/object query to be invoked. This function should probably only be used by designer generated javascript.
 *
 * @memberOf hwc
 * @private
 *
 * @param {string} credInfo Credential info in the format "supusername=usernameValue&suppassword=passwordValue"
 *
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 *
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 *
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 *
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 *
 * @param {string} serializeDataMessageToSend The data message, already serialized. This parameter should be obtained by calling serializeToString
 */

hwc.doActivateWorkflow_CONT = function(credInfo, serializeDataMessageToSend) {
var request, xmlhttp;

hwc.traceEnteringMethod("hwc.doActivateWorkflow_CONT");

try {
    request = "xmlWorkflowMessage=" + encodeURIComponent(serializeDataMessageToSend);

    if (credInfo) {
        request += ("&" + credInfo);
    }

    hwc.postDataToContainer("activate", request);
}

finally {
    hwc.traceLeavingMethod("hwc.doActivateWorkflow_CONT");
}

/**
 * This function should probably only be used by designer
 * generated javascript.
 *
 * @memberOf hwc
 * @private
 *
 * @param {string} credInfo Credential info in the format
 *   "supusername=usernameValue&suppassword=passwordValue"
 *
 * @param serializeDataMessageToSend The data message,
 *   already serialized. This parameter should be obtained by calling
 *   serializeToString
 *
 * @param hwc.getMessageValueCollectionForOnlineRequest.
 *
 */

hwc.doCredentialsSubmit_CONT = function(credInfo, serializeDataMessageToSend) {

    hwc.traceEnteringMethod("hwc.doCredentialsSubmit_CONT");

    hwc.traceLeavingMethod("hwc.doCredentialsSubmit_CONT");

};
```
try {

    var request = "xmlWorkflowMessage=" +
    encodeURIComponent(serializeDataMessageToSend);

    if (credInfo) {
        request += ("&" + credInfo);
    }

    hwc.postDataToContainer("credentials", request);
} finally {

    hwc.traceLeavingMethod("hwc.doCredentialsSubmit_CONT");
}

/**
 * This function shows a progress dialog with spinner. The dialog created by this function will block all
 * user input until {@link hwc.hideProgressDialog} is called. It is important to be sure that
 * {@link hwc.hideProgressDialog} will be called after a call to this function.
 *
 * @memberof hwc
 * @public
 * @param {string} [message] The message to show on the progress dialog. This message is displayed on Android
 * platforms only - other platforms show only a spinner.
 * @example
 * var showProgress = function()
 * {
 *     hwc.showProgressDialog("a message");
 *     setTimeout( hideProgress, 10000 );
 * } 
```
var hideProgress = function() {
    hwc.hideProgressDialog();
}

hwc.showProgressDialog = function(message) {
    hwc.traceEnteringMethod("hwc.showProgressDialog");
    try {
        hwc.getDataFromContainer("showprogressdialog", 
        "&message=" + message);
    } finally {
        hwc.traceLeavingMethod("hwc.showProgressDialog");
    }
}

/**
 * This function hides the progress dialog displaying the spinner. This function should be used to hide
 * the progress dialog after a call to {link hwc.showProgressDialog}. If this function is called while there
 * is no progress dialog, then nothing will happen.
 * @memberOf hwc
 * @public
 * @example
 * var showProgress = function()
 * {
 *     hwc.showProgressDialog( "a message" );
 *     setTimeout( hideProgress, 10000 );
 * }
 * */
hwc.hideProgressDialog = function() {
    hwc.traceEnteringMethod("hwc.hideProgressDialog");
    try {
        hwc.getDataFromContainer("hideprogressdialog");
    } finally {
        hwc.traceLeavingMethod("hwc.hideProgressDialog");
    }
};

/**
 * Displays an alert dialog to the user. This function blocks until it receives a response from the user.
 *
 * @memberOf hwc
 * @public
 * @param {string} message The message to display
 * @param {string} [title] The title doesn't actually get displayed.
 *
 * @example
 * hwc.showAlertDialog("This is a fancy alert dialog", "With a Title");
 * /

hwc.showAlertDialog = function(message, title) {
    if(hwc.isIOS()){
        // For ios client, creating an IFRAME element for the alert message, so as to hide the
        // title bar in the alert box
        var iframe = document.createElement("IFRAME");
        iframe.setAttribute("src", 'data:text/plain');
        document.documentElement.appendChild(iframe);
window.frames[window.frames.length-1].alert(message);

iframe.parentNode.removeChild(iframe);
}
else{
    alert(message);
}
};

/**
 * Shows a confirm dialog to the user. This function blocks until it receives a response from the user.
 */

* @memberOf hwc
* @public
* @param {string} message The message to display in the dialog.
* @param {string} [title] The title doesn't actually get displayed.
* @returns {boolean} The user's choice from the confirm dialog.
* @example
* var userConfirm = hwc.showConfirmDialog( "Are you sure you want to see an alert message?", "Confirm Alert" );
* if( userConfirm )
* {
*    alert( "This is what you wanted." );
* }
*/

hwc.showConfirmDialog = function(message, title) {
    return confirm(message);
};
/**
 * This function closes the hybrid app.
 * @memberOf hwc
 * @public
 * @example
 * hwc.close();
 */

hwc.close = function() {
    workflowMessage = "";
    hwc.supUserName = "";
    hwc.traceEnteringMethod("hwc.close");
    try {
        if (hwc.isWindowsMobile()) {
            if(typeof(hwc.setWindowBlankScreen) === 'function') {
                hwc.setWindowBlankScreen();
            }
            hwc.getDataFromContainer("close");
        }
        else if (hwc.isIOS()) {
            hwc.getDataFromContainer("close");
        }
        else if (hwc.isAndroid()) {
            hwc.log("Closing Hybrid App", "INFO");
            _HWC.close();
        }
        else {
            window.close();
        }
    }
}

Develop Hybrid Apps Using Third-party Web Frameworks

SAP Mobile Platform
status = hwc.STATUS.CLOSED;
} finally {
    hwc.traceLeavingMethod("hwc.close");
}

/**
 * This function checks if the hybrid app has been closed.
 * @returns {boolean} true if hybrid app is closed, otherwise false.
 * @memberOf hwc
 * @public
 * @example
 * hwc.isClosed();
 */
hwc.isClosed = function() {
    return status === hwc.STATUS.CLOSED;
};

})(hwc, window);

/**
 * A callback function invoked when `{@link hwc.log}` is invoked with true for the `notifyUser` parameter.
 * This callback should notify the user of the log message in an appropriate manner.
 * @name anonymous.alertDialogCallbackFunction
 * @param {string} message The message that the user should be notified of.
 * @function
 */
Develop Hybrid Apps Using Third-party Web Frameworks

```
   1618 /*
   1619   * A callback function invoked if there is an error.
   1620   *
   1621   * @name anonymous.errorCallbackFunction
   1622   *
   1623   * @param {string} errorMessage The message describing the error.
   1624   *
   1625   * @function
   1626   */
   1627
   1628 /**
   1629   * A generic callback function that takes no parameters. Used to execute code when a certain event occurs.
   1630   *
   1631   * @name anonymous.genericCallbackFunction
   1632   *
   1633   * @function
   1634   */

hwc-utils.js

   1   /**
   2   * Sybase Hybrid App version 2.3.4
   3   *
   4   * Utils_CONT.js - container maintained aspect
   5   *
   6   * This file will not be regenerated, so it is possible to modify it, but it
   7   * is not recommended.
   8   *
   9   * The template used to create this file was compiled on Thu Jun 07 14:57:11 EDT 2012
   10   */
```
Develop Hybrid Apps Using Third-party Web Frameworks

11       * Copyright (c) 2012 Sybase Inc. All rights reserved.
12       */
13      /**
14       * The namespace for the Hybrid Web Container javascript
15       * @namespace */
16      hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc; // SUP 'namespace'
17
18      /**
19       * Container Utilities
20       */
21      (function(hwc, window, undefined) {
22
23      /****************************** PUBLIC CONSTANTS
24      ******************************
25      /** @private */
26      hwc.versionURLParam = "version=2.2";
27
28      /****************************** PUBLIC API ******************************/
29
30      /**
31      * The version number sent with the HTTP messages to the
32      * native code.
33      * Used for internal versioning only
34      * @private
35      * @returns {String} the version string
36      */
37      hwc.getVersionURLParam = function() {
38          return hwc.versionURLParam;
39      };
/**
 * Internal worker for initial HybridApp loading.
 * Returns the response message or empty.
 * @private
 */

hwc.onHybridAppLoad.CONT = function() {
    var response = hwc.getTransformData();
    processDataMessage(response, false, true);
};

/**
 * Returns the transform data for the hybridapp. Only a server-initiated app will have this data.
 * @example
 * TODO: Add an example
 * @returns the transform data.
 * @public
 * @memberOf hwc
 */

hwc.getTransformData = function() {
    var xmlhttp;
    hwc.traceEnteringMethod("hwc.getTransformData");

    try {
        if (hwc.isWindows()) {
            xmlhttp = hwc.getXMLHTTPRequest();
            xmlhttp.open("GET", "transform.xml", false);
            xmlhttp.send("");
            if (xmlhttp.status === 200 || xmlhttp.status === 0) { //Win32 returns 200 for OK, WM returns 0 for OK
                return xmlhttp.responseText;
            }
        }
    }
else
{
    return
hwc.getDataFromContainer("loadtransformdata");
}
} finally {
    hwc.traceLeavingMethod("hwc.getTransformData");
}

/**Internal worker for adding a single menu item.
 * @private
 * @param {string} menuStr Information of the menu.
 */
hwc.addNativeMenuItem_CONT = function (menuStr ) {
    hwc.postDataToContainer("addallmenuitems",
"menuitems=" + encodeURIComponent(menuStr));
};

/**Internal worker setting credential information.
 * @private
 * @param {string} credInfo Information of the credential.
 */
hwc.handleCredentialChange_CONT = function(credInfo) {
    var requestData = credInfo ? credInfo : "";
    if (requestData) {
        if (!hwc.isWindows()) {
            hwc.postDataToContainer("formredirect",
requestData);
        }
    }
};
 Remove spaces from the specified string.

@private

@param {string} str The specified string

@param {boolean} leftAndRightOnly When true removes leading and trailing spaces

@returns {string} The trimmed string

memberOf hwc

/**
 * @memberOf hwc
 * @param {string} value The string to be parsed.
 */

hwc.parseBoolean = function(value) {
    if (value) {
        return hwc.trimSpaces(value, true).toLowerCase() === "true";
    }
    else {
        return false;
    }
};

/**
 * Extract the error message from a URL string. The parameter name of the error message should be "onErrorMsg".
 */
hwc.getOnErrorMessageFromNativeError = function getOnErrorMessageFromNativeError(errString) {
    hwc.traceEnteringMethod("hwc.getOnErrorMessageFromNativeError");
    try {
        if( hwc.isBlackBerry() ) {
            return unescape(hwc.getURLParamFromNativeError("onErrorMsg", errString));
        } else {
            // This is a temporary fix for a bug in the container that calls
            // encodeURIComponent on the whole query string for Android. See
            // IR 676161-2.
            return hwc.getURLParamFromNativeError("onErrorMsg", errString);
        }
    } finally {
        hwc.traceLeavingMethod("hwc.getOnErrorMessageFromNativeError");
    }
};
function getCallbackFromNativeError(errString) {
    hwc.traceEnteringMethod("hwc.getCallbackFromNativeError");
    try {
        return hwc.getURLParamFromNativeError("onErrorCallback", errString);
    } finally {
        hwc.traceLeavingMethod("hwc.getCallbackFromNativeError");
    }
}

function getCodeFromNativeError(errString) {
    hwc.traceEnteringMethod("hwc.getCodeFromNativeError");
    try {
        return hwc.getURLParamFromNativeError("errCode", errString);
    } finally {
        hwc.traceLeavingMethod("hwc.getCodeFromNativeError");
    }
}
186     }
187
188     /**
189     * Extract a native message from a URL string. The parameter name of the native message should be "nativeErrMsg".
190     * @param {String} errString The error string URL
191     * @returns {String} the native message
192     * @memberOf hwc
193     * @public
194     */
195     hwc.getNativeMessageFromNativeError = function
196             getNativeMessageFromNativeError( errString ) {
197                 try {
198                     return hwc.getURLParamFromNativeError("nativeErrMsg", errString);
199                 } finally {
200                     hwc.traceLeavingMethod("hwc.getNativeMessageFromNativeError");
201             }
202         }
203
204     /**
205     * Extract a parameter value from a URL string with a given parameter name.
206     * @param {String} paramName The parameter name
207     * @param {String} url The containing URL of the parameter
208     * @returns {String} The parameter value
209     * @memberOf hwc
210     * @public
211     */
212     hwc.getURLParamFromNativeError = function
213             getURLParamFromNativeError(paramName, url) {
214                 try {
215                     return hwc.getURLParamFromNativeError("nativeErrMsg", errString);
216                 } finally {
217                     hwc.traceLeavingMethod("hwc.getURLParamFromNativeError");
218                 }
219             }
220         };
```javascript
var indxofS, indxofE, pName, pValue, paramSection, ret,
    paramSectionsAmp, ampSections, indxofA;

hwc.traceEnteringMethod("hwc.getURLParamFromNativeError");

try {
    if (hwc.isBlackBerry()) {
        paramSection = url;
    } else {
        // This is a temporary fix for a bug in the container that calls
        // encodeURIComponent on the whole query string for Android. See
        // IR 676161-2.
        paramSection = decodeURIComponent(url);
    }

    indxofA = paramSection.indexOf("&");
    if (indxofA > 0) {//there is one or more parameters in the & section
        paramSectionsAmp = paramSection.substring(indxofA + 1);
        ampSections = paramSection.split("&");
        if (ampSections.length === 1) {
            indxofE = paramSectionsAmp.indexOf("=");
            pName = paramSectionsAmp.substring(0, indxofE);
            if (pName.toLowerCase() === paramName.toLowerCase()) {
                pValue = paramSectionsAmp.substring(indxofE + 1);
                ret = decodeURIComponent(pValue);
                return ret;
            }
        } else {  //multiple parameters in the & section
            for (indxofS in ampSections) {
                pValue = paramSectionsAmp.substring(indxofE + 1);
                ret = decodeURIComponent(pValue);
                return ret;
            }
        }
    } else {  //there is one or more parameters in the & section
        pName = paramSectionsAmp.substring(0, indxofE);
        if (pName.toLowerCase() === paramName.toLowerCase()) {
            pValue = paramSectionsAmp.substring(indxofE + 1);
            ret = decodeURIComponent(pValue);
            return ret;
        }
    }
}
```

idxofE = ampSections[indxofS].indexOf("=");

pName = ampSections[indxofS].substring(0, idxofE);

if (pName.toLowerCase() === paramName.toLowerCase()) {
    pValue = ampSections[indxofS].substring(idxofE + 1);
    ret = decodeURIComponent(pValue);
    return ret;
}

// ok did not find paramName in & section look for it at the start
idxofE = paramSection.indexOf("=");
pName = paramSection.substring(0, idxofE);

if (pName.toLowerCase() === paramName.toLowerCase()) {
    pValue = paramSection.substring(idxofE + 1, idxofA);
    ret = decodeURIComponent(pValue);
    return ret;
}

} else { // only one param
idxofE = paramSection.indexOf("=");
pName = paramSection.substring(0, idxofE);

if (pName.toLowerCase() === paramName.toLowerCase()) {
    pValue = paramSection.substring(idxofE + 1);
    ret = decodeURIComponent(pValue);
    return ret;
}

}
return pValue;
}

finally {

hwc.traceLeavingMethod("hwc.getURLParamFromNativeError");
}

/**
 * Log the behavior of entering a JavaScript method.
 * @param {String} methodName The target method name.
 * @private
 */

hwc.traceEnteringMethod = function (methodName) {
    if (hwc.getLoggingCurrentLevel() >= 4) { hwc.log("entering " + methodName + "()", "DEBUG", false); }
}

/**
 * Log the behavior of leaving a JavaScript method.
 * @param {String} methodName The target method name.
 * @private
 */

hwc.traceLeavingMethod = function (methodName) {
    if (hwc.getLoggingCurrentLevel() >= 4) { hwc.log("exiting " + methodName + "()", "DEBUG", false); }
}

})(hwc, window);

PlatformIdentification.js

/*
 * Sybase Hybrid App version 2.3.4
 */
* PlatformIdentification.js

* This file will not be regenerated, so it is possible to modify it, but it is not recommended.

* Original file date: 2012-Oct-22

* Copyright (c) 2012 Sybase Inc. All rights reserved.

*/

/**
* The namespace for the Hybrid Web Container javascript
* @namespace
*/

hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc;  // SUP 'namespace'

(function(hwc, window, undefined) {

    // private variables
    // platform identifiers are all calculated once and cached
    var _isIOS = false, _isIPad = false, _isIOS5 = false, _isIOS6 = false, _isIOS7 = false, _isIOS4 = false,
    _isBB = false, _isBB5 = false, _isBB5Touch = false, _isBB6NonTouch = false, _isBB7 = false,
    _isAndroid = false, _isAndroid3 = false,
    _isWindows = false, _isWinMobile = false;

    // public API
    /**
     * Returns true if the hybrid app application is being run on an iOS (e.g. iPhone, iPad) platform.
     * @desc Platform
     * @memberOf hwc
     * @public
     */

});
Develop Hybrid Apps Using Third-party Web Frameworks

32          *
33          * @returns {boolean} True if the hybrid app application is being run on an iOS (e.g. iPhone, iPad) platform.
34          */
35          hwc.isIOS  = function() { return _isIOS;  };
36          /**
37          * Returns true if the hybrid app application is being run on an iPad.
38          * @desc Platform
39          * @memberOf hwc
40          * @public
41          * @returns {boolean} True if the hybrid app application is being run on an iPad.
42          */
43          hwc.isIPad = function() { return _isIPad; };
44          /**
45          * Returns true if the hybrid app application is being run on iOS5
46          * @desc Platform
47          * @memberOf hwc
48          * @public
49          * @returns {boolean} True if the hybrid app application is being run on iOS5
50          */
51          hwc.isIOS5 = function() { return _isIOS5; };
52          /**
53          * Returns true if the hybrid app application is being run on iOS6
54          * @memberOf hwc
55          * @public
56          * @returns {boolean} True if the hybrid app application is being run on iOS6
57          */
58          hwc.isIOS6 = function() { return _isIOS6; };
59          /**
Develop Hybrid Apps Using Third-party Web Frameworks

59  * Returns true if the hybrid app application is being run on iOS7
60  * @public
61  * @return {boolean} True if the hybrid app application is being run on iOS7
62  * @memberof hwc
63  * @public
64  */
65  hwc.isIOS7 = function() { return _isIOS7; };
66
67  /**
68  * Returns true if the hybrid app application is being run on iOS4
69  * @feature Platform
70  * @return {Boolean} True if the hybrid app application is being run on iOS4
71  * @memberof hwc
72  * @public
73  */
74  hwc.isIOS4 = function() { return _isIOS4; };
75
76  /**
77  * Returns true if the hybrid app application is being run on a BlackBerry platform.
78  * @desc Platform
79  * @memberof hwc
80  * @public
81  * @returns {boolean} True if the hybrid app application is being run on a BlackBerry platform.
82  */
83  hwc.isBlackBerry = function() { return _isBB; };
84  /**
85  * Returns true if the hybrid app application is being run on a BlackBerry 5.0 OS
* @desc Platform
* @memberOf hwc
* @public
* @returns {boolean} True if the hybrid app application is being run on a BlackBerry 5.0 OS
*/

hwc.isBlackBerry5 = function() { return _isBB5; };

/**
* Returns true if the hybrid app application is being run on a BlackBerry 7.x OS
* @desc Platform
* @memberOf hwc
* @public
* @returns {boolean} True if the hybrid app application is being run on a BlackBerry 7.x OS
*/

hwc.isBlackBerry7 = function() { return _isBB7; };

/**
* Returns true if the hybrid app application is being run on a BlackBerry 5.0 OS with a touch screen
* @desc Platform
* @memberOf hwc
* @public
* @returns {boolean} True if the hybrid app application is being run on a BlackBerry 5.0 OS with a touch screen
*/

hwc.isBlackBerry5WithTouchScreen = function() { return _isBB5Touch; };

/**
* Returns true if the hybrid app application is being run on a BlackBerry 6.0 OS without a touch screen
* @desc Platform
* @memberOf hwc
* @public
* @returns {boolean} True if the hybrid app application is being run on a BlackBerry 6.0 OS without a touch screen

```javascript
hwc.isBlackBerry6NonTouchScreen = function() { return _isBB6NonTouch; };
```

/**
 * Returns true if the hybrid app application is being run on a Windows Mobile platform.
 *
 * @returns {boolean} True if the hybrid app application is being run on a Windows Mobile platform.
 */

```javascript
hwc.isWindowsMobile = function() { return _isWinMobile; };
```

/**
 * Returns true if the hybrid app application is being run on a Windows platform.
 *
 * @returns {boolean} True if the hybrid app application is being run on a Windows platform.
 */

```javascript
hwc.isWindows = function() { return _isWindows; };
```

/**
 * Returns true if the hybrid app application is being run on an Android 3.0 OS
 */

```javascript
hwc.isAndroid30 = function() { return _isAndroid30; };
```
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
139         * @returns {boolean} True if the hybrid app application is
140         * being run on an Android 3.0 OS
141         */
142         hwc.isAndroid3 = function() { return _isAndroid3; };
143         /**
144         * Returns true if the hybrid app application is being run
145         * on an Android platform.
146         * @desc Platform
147         * @memberOf hwc
148         * @public
149         * @returns {boolean} True if the hybrid app application is
150         * being run on an Android platform.
151         */
152         hwc.isAndroid = function() { return _isAndroid; };
153         /**
154         * @private
155         * @returns {boolean} True if this code is running on a
156         * Blackberry 5 OS
157         */
158         function _isBlackBerry5() {
159             var ua = navigator.userAgent;
160             if (ua.indexOf("Blackberry 9800") >= 0) {
161                 return false;
162             }
163             if (ua.match(/5\.[0-9]\.[0-9]/i) !== null) {
164                 return true;
165             }
166             return false;
167         }
```
function _isBlackBerry5WithTouchScreen() {
    if (isBlackBerry5()) {
        var ua = navigator.userAgent;
        if (ua.length > 12 && ua.substring(0, 12) === "BlackBerry95") {
            return true;
        }
    }
    return false;
}

function _isBlackBerry6NonTouchScreen() {
    if (navigator.userAgent.match(/Version/6./i)) {
        var ua = navigator.userAgent;
        if ((ua.indexOf('9780') > 0) || (ua.indexOf('9700') > 0) || (ua.indexOf('9650') > 0) || (ua.indexOf('9300') > 0) || (ua.indexOf('9330') > 0)) {
            return true;
        }
    }
    return false;
}
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
197          * @returns {boolean} True if this code is running on a Blackberry & OS
198          */
199         function _isBlackBerry7() {
200             if (navigator.userAgent.match(/Version\//7\.[0-9]\. [0-9]/i) !== null){
201                 return true;
202             }
203             else {
204                 return false;
205             }
206         }
207
208         /**
209          * Execute once to identify and cache
210          */
211         {
212             // apple products
213             _isIOS = ((navigator.platform.indexOf("i") === 0));
214             if ( _isIOS ) {
215                 _isIOS4 = (navigator.userAgent.match(/OS 4_[0-9]+ like Mac OS X/i) !== null);
216                 _isIOS5 = (navigator.userAgent.match(/OS 5_[0-9]+ like Mac OS X/i) !== null);
217                 _isIOS6 = (navigator.userAgent.match(/OS 6_[0-9]+ like Mac OS X/i) !== null);
218                 _isIOS7 = (navigator.userAgent.match(/OS 7_[0-9]+ like Mac OS X/i) !== null);
219                 _isIPad = (navigator.userAgent.match(/iPad/i) !== null);
220             }
221         }
222
223         // BlackBerry
```
224             _isBB  = (navigator.platform === "BlackBerry");
225             if( _isBB ) {
226                 _isBB5 = _isBlackBerry5();
227                 _isBB7 = _isBlackBerry7();
228                 _isBB5Touch = _isBlackBerry5WithTouchScreen();
229                 _isBB6NonTouch = _isBlackBerry6NonTouchScreen();
230             }
231
232             // Android
233             _isAndroid  = (navigator.userAgent.indexOf("Android") > -1);
234             if( _isAndroid ) {
235                 _isAndroid3 = (navigator.userAgent.indexOf("3.0") > -1);
236             }
237
238             // Windows
239             _isWinMobile = (navigator.platform === "WinCE");
240             _isWindows   = ( (navigator.platform === "Win32") ||
                                 (navigator.platform === "Win64") ||
                                 (navigator.platform === "MacIntel") ||
                                 (!_isAndroid &&
                                 (navigator.platform.indexOf("Linux") === 0)) );
241
242             //alert("Platform Identified: Win=" + _isWindows + ",
243             BB=" + _isBB);
244             })(hwc, window);
245         }
246     })(hwc, window);
247
248
249     /**
250      * Returns true if the hybrid app application is being run on
251      an iOS (e.g. iPhone, iPad) platform.
252  */

251    * @private
252    * @returns {boolean} True if the hybrid app application is
253    * being run on an iOS (e.g. iPhone, iPad) platform.
254    */
255    function isIOS() { return hwc.isIOS(); }
256
257    /**
258    * Returns true if the hybrid app application is being run on
259    * iOS5
260    * @private
261    * @returns {boolean} True if the hybrid app application is
262    * being run on iOS5
263    */
264    function isIOS5() { return hwc.isIOS5(); }
265
266    /**
267    * Returns true if the hybrid app application is being run on
268    * an iPad.
269    * @private
270    * @returns {boolean} True if the hybrid app application is
271    * being run on an iPad.
272    */
273    function isIPad() { return hwc.isIPad(); }
274
275    /**
276    * Returns true if the hybrid app application is being run on
277    * a BlackBerry platform.
278    * @private
279    * @returns {boolean} True if the hybrid app application is
280    * being run on a BlackBerry platform.
281    */
282    function isBlackBerry() { return hwc.isBlackBerry(); }
function isBlackBerry5() { return hwc.isBlackBerry5(); }

function isBlackBerry5WithTouchScreen() { return hwc.isBlackBerry5WithTouchScreen(); }

function isBlackBerry6NonTouchScreen() { return hwc.isBlackBerry6NonTouchScreen(); }

function isBlackBerry7() { return hwc.isBlackBerry7(); }
>Returns true if the hybrid app application is being run on a Windows Mobile platform.

* @private

* @returns {boolean} True if the hybrid app application is being run on a Windows Mobile platform.

*/

function isWindowsMobile() { return hwc.isWindowsMobile(); }

/**

Returns true if the hybrid app application is being run on a Windows platform.

* @private

* @returns {boolean} True if the hybrid app application is being run on a Windows platform.

*/

function isWindows() { return hwc.isWindows(); }

/**

Returns true if the hybrid app application is being run on an Android platform.

* @private

* @returns {boolean} True if the hybrid app application is being run on an Android platform.

*/

function isAndroid() { return hwc.isAndroid(); }

/**

Returns true if the hybrid app application is being run on an Android 3.0 OS

* @private

* @returns {boolean} True if the hybrid app application is being run on an Android 3.0 OS

*/
Plugins/AppLog/applog.js

1       /*
2        * Sybase Hybrid App version 2.3.4
3        * Sybase PhoneGap AppLog plugin
4        *
5        * applog.js
6        */
7        * This file will not be regenerated, so it is possible to modify it, but it
8        * is not recommended.
9        *
10       */
11
12       /**
13       * The namespace for AppLog plugin
14       * @namespace
15       */
16       AppLog = (typeof AppLog === "undefined" || !AppLog) ? {} : AppLog; // 'namespace'
17
18       (function(AppLog, window, undefined) {
19         /**
20         * Constant indicating the operation failed with unknown error. Used in {@link anonymous.AppLogErrorCallbackParameter}.
21         * @type number
22         */
23         AppLog.ERR_UNKNOWN = -1;
24
25       /**
26       * Constant indicating an app log entry is associated with an unknown event. Used in {@link AppLog.LogEntry}.
27       */
Develop Hybrid Apps Using Third-party Web Frameworks

```java
27    * @type number
28    */
29    AppLog.STATUS_EVENT_UNKNOWN = 1;
30
31    /**
32    * Constant indicating an app log entry is associated with starting the client connection to the SUP server. Used in {@link AppLog.LogEntry}.
33    * @type number
34    */
35    AppLog.STATUS_EVENT_STARTUP = 2;
36
37    /**
38    * Constant indicating an app log entry is associated with shutting down the client connection to the SUP server. Used in {@link AppLog.LogEntry}.
39    * @type number
40    */
41    AppLog.STATUS_EVENT_SHUTDOWN = 3;
42
43    /**
44    * Constant indicating an app log entry is associated with restarting the client connection to the SUP server. Used in {@link AppLog.LogEntry}.
45    * @type number
46    */
47    AppLog.STATUS_EVENT_RESTART = 4;
48
49    /**
50    * Constant indicating an app log entry is associated with the client successfully connecting to the SUP server. Used in {@link AppLog.LogEntry}.
51    * @type number
52    */
53    AppLog.STATUS_EVENT_CONNECTED = 5;
```
54
55    /**
56     * Constant indicating an app log entry is associated with the client losing connection to the SUP server. Used in {@link AppLog.LogEntry}.
57     * @type number
58     */
59     AppLog.STATUS_EVENT_DISCONNECTED = 6;
60
61    /**
62     * Constant indicating an app log entry is associated with the client going into flight mode. Used in {@link AppLog.LogEntry}.
63     * @type number
64     */
65     AppLog.STATUS_EVENT_FLIGHT_MODE = 7;
66
67    /**
68     * Constant indicating an app log entry is associated with the client going out of network. Used in {@link AppLog.LogEntry}.
69     * @type number
70     */
71     AppLog.STATUS_EVENT_OUT_OF_NETWORK = 8;
72
73    /**
74     * Constant indicating an app log entry is associated with the client waiting to connect to the SUP server. Used in {@link AppLog.LogEntry}.
75     * @type number
76     */
77     AppLog.STATUS_EVENT_WAITING_TO_CONNECT = 9;
78
79    /**
80     * Constant indicating an app log entry is associated with the client losing connection to the SUP server due to roaming. Used in {@link AppLog.LogEntry}.

* @type number

```javascript
AppLog.STATUS_EVENT_DISCONNECTED_ROAMING = 10;
/**
 * Constant indicating an app log entry is associated with the client losing connection to the SUP server due to low storage. Used in {@link AppLog.LogEntry}.
 */
AppLog.STATUS_EVENT_DISCONNECTED_LOW_STORAGE = 11;
/**
 * Constant indicating an app log entry is associated with the client starting registration. Used in {@link AppLog.LogEntry}.
 */
AppLog.STATUS_EVENT_REGISTRATION_STARTED = 12;
/**
 * Constant indicating an app log entry is associated with the client receiving a notification. Used in {@link AppLog.LogEntry}.
 */
AppLog.STATUS_EVENT_NOTIFICATION_RECEIVED = 13;
/**
 * Constant indicating an app log entry is associated with a default app being set from the server. Used in {@link AppLog.LogEntry}.
 */
AppLog.STATUS_EVENT_SET_DEFAULT_ITEM = 14;
```
/**
 * Constant indicating an app log entry is associated with a default app being unset from the server. Used in {@link AppLog.LogEntry}.
 * @type number
 */
AppLog.STATUS_EVENT_UNSET_DEFAULT_ITEM = 15;

/**
 * This object represents a log entry.
 *
 * @classdesc
 * @public
 * @memberOf AppLog
 * @param {number} logDate The date the log entry was recorded, in milliseconds since January 1, 1970, 00:00:00 GMT.
 * @param {number} event The event ID of the log entry (will be one of the AppLog status events, or possibly a custom value).
 * @param {string} msg The message of the log entry.
 */
AppLog.LogEntry = function ( logDate, event, msg ) {
    this.date = logDate;
    this.statusCode = event;
    this.message = msg;
};

/**
 * Call this function to get an array of {@link AppLog.LogEntry} objects. There will be one
 * {AppLog.LogEntry} object for each line in the app log.
 *
 * @public
 * @memberOf AppLog
 */
successCB The callback function that will receive the asynchronous
        callback with the log entries.
    errorCB The callback function that will be invoked on errors.

    @example

    // A global function called with the log entries.
    function onLogEntriesSuccessCallback(data) {
        for ( var i = 0; i < data.length; i++ )
            var logEntry = data[ i ];
        alert('Log entry ' + ( i + 1 ) + ':
            Date (ms): ' + logEntry.date + '
            Status code: ' + logEntry.statusCode + '
            Message: ' + logEntry.message
        );
    }

    // A global function called if there is an error
    retrieving log entries.
    function onLogEntriesFailureCallback(error) {
        alert('Error retrieving log entries: ' + error);
    }

    // Get the log entries
    AppLog.getLogEntries(onLogEntriesSuccessCallback,
        onLogEntriesFailureCallback);
*/

AppLog.getLogEntries = function( successCB, errorCB ) { try
    try {
cordova.exec( successCB, errorCB, "AppLog", "getLogEntries", [] );

} catch (ex) {
    setTimeout( errorCB({errorCode : AppLog.ERR_UNKNOWN, description : ex.message}), 0 );
}

/**
 * Registers a log listener.
 *
 * @public
 * @memberOf AppLog
 * @param {anonymous.startOrStopLogListenerSuccessCallback} successCB A callback function that will be invoked if the log listener is successfully registered.
 * @param {anonymous.startOrStopLogListenerErrorCallback} errorCB A callback function that will be invoked if there is an error registering the log listener.
 * @param {anonymous.logListner} logListener The callback to register. This will be invoked when new entries are added to the log.
 * @param {Object} [containingObject] Object containing the definition for logListener. If a log listener callback function references variables in its containing object, then the containing object should be passed to this function.
 *
 * @example
 * // This example shows how to use this function with a globally-scoped logListener.
 * var doSomething = function() {
 *     // A global function called by the log listener.
 *     ...
 * }
alert("this gets displayed when there is a new log entry.");

} } 

* // The log listener callback function that will be passed to AppLog.startLogListener.

* // This function will be invoked whenever there is a new log entry.

var logListener = function( date, statusCode, message )

{ 
  doSomething();
}

* 

* function onStartLogListenerSuccessCallback() {
  * // Do something here after listener has been added
}

* 

* function onStartLogListenerFailureCallback(error) {
  * // React to error here
}

* 

* // Add the log listener.

AppLog.startLogListener( onStartLogListenerSuccessCallback,

onStartLogListenerFailureCallback,

logListener );

* 

* @example

* // This example shows how to use this function with a logListener contained in an object.

* // logListenerManager is an object that will contain the listener callback as well

* // as a function that will be invoked from the listener callback function.
* var logListenerManager = {};

* // This is a function that is called from the listener callback.

* logListenerManager.doSomething = function()

*  alert("this gets displayed when there is a new log entry.");

* }

* // This is the listener callback that will be passed to AppLog.startLogListener.

* // Since a variable is referenced from the containing object, the containing object

* // will need to be passed to AppLog.startLogListener.

* logListenerManager.listener = function( date, statusCode, message )

*  this.doSomething();

* }

* function onStartLogListenerSuccessCallback() {

*  // Do something here after listener has been added

* }

* function onStartLogListenerFailureCallback(error) {

*  // React to error here

* }

* // Pass both the listener callback and the containing object.

* AppLog.startLogListener( onStartLogListenerSuccessCallback,

* onStartLogListenerFailureCallback,
AppLog.startLogListener = function( successCB, errorCB, logListener, containingObject ) {

    try {

        if (!logListener || ( typeof logListener !== "function" ) ) {

            throw new Error( "AppLog.startLogListener Error: logListener is not a function" );

        } else {

            var newListener = new AppLog.logListenerCallBack( logListener, containingObject );

            AppLog.logListeners_internal.push( newListener );

            if ( AppLog.logListeners_internal.length === 1 ) {

                cordova.exec( successCB, errorCB, "AppLog", "startLogListener", [] );

            } else {

                setTimeout( successCB(), 0 );

            }

        } catch (ex) {

            setTimeout( errorCB( {errorCode : AppLog.ERR_UNKNOWN, description : ex.message} ), 0 );

        }

    } catch (ex) {

    }

}
/**
 * Removes a log listener. This function should be called
 * with identical parameters that were used
 * when adding the log listener with {@link AppLog.startLogListener}.
 *
 * @param {anonymous.startOrStopLogListenerSuccessCallback} successCB A
 * callback function that will be invoked
 * if the log listener is successfully removed.
 *
 * @param {anonymous.startOrStopLogListenerErrorCallback} errorCB A callback function that will be invoked if there
 * is an error removing the log listener.
 *
 * @param {anonymous.logListener} logListener The callback
 * that was added with {@link AppLog.startLogListener}.
 *
 * @param {Object} [containingObject] Object containing the
 * definition for logListener.
 *
 * @public
 * @memberOf AppLog
 *
 * @example
 * // This example shows how to use this function with a
 * globally-scoped logListener.
 * // A global function called by the log listener.
 * var doSomething = function()
 * {
 *    alert("this gets displayed when there is a new log
 * entry.");
 * }
 *
 * // The log listener callback function that will be
 * passed to AppLog.startLogListener.
 * // This function will be invoked whenever there is a new
 * log entry.
 * var logListener = function( date, statusCode, message )
 * {

doSomething();

function onStartLogListenerSuccessCallback() {
    // Do something here after listener has been added
}

function onStartLogListenerFailureCallback(error) {
    // React to error here
}

function onStopLogListenerSuccessCallback() {
    // Do something here after listener has been removed
}

function onStopLogListenerFailureCallback(error) {
    // React to error here
}

// Add the log listener.
AppLog.startLogListener( onStartLogListenerSuccessCallback,
    onStartLogListenerFailureCallback,
    logListener );

// At some other point if we want to remove the listener, we use the following line.
AppLog.stopLogListener( onStopLogListenerSuccessCallback,
    onStopLogListenerFailureCallback,
    logListener );
* @example

// This example shows how to use this function with a logListener contained in an object.

// logListenerManager is an object that will contain the listener callback as well

// as a function that will be invoked from the listener callback function.

var logListenerManager = {};

// This is a function that is called from the listener callback.

log ListenerManager.doSomething = function()

{ alert("this gets displayed when there is a new log entry.");

}

// This is the listener callback that will be passed to AppLog.startLogListener.

// Since a variable is referenced from the containing object, the containing object

// will need to be passed to AppLog.startLogListener.

logListenerManager.listener = function( date, statusCode, message )

{ this.doSomething();

}

* function onStartLogListenerSuccessCallback() {

  // Do something here after listener has been added

}

* function onStartLogListenerFailureCallback(error) {

  // React to error here
354    * }
355    *
356    * function onStopLogListenerSuccessCallback() {
357    *     // Do something here after listener has been
358    *     removed
359    * }
360    *
361    * function onStopLogListenerFailureCallback(error) {
362    *     // React to error here
363    * }
364    *
365    * // Pass both the listener callback and the containing
366    * object.
367    * AppLog.startLogListener( onStartLogListenerSuccessCallback,
368    *                         onStartLogListenerFailureCallback,
369    *                         logListenerManager.listener,
370    *                         logListenerManager );
371    *
372    * // At some other point if we want to remove the listener, we use the following line.
373    *
374    * AppLog.stopLogListener( onStopLogListenerSuccessCallback,
375    *                         onStopLogListenerFailureCallback,
376    *                         logListenerManager.listener,
377    *                         logListenerManager );
378    */
379    AppLog.stopLogListener = function( successCB, errorCB, logListener, containingObject ) {
380    try
381    {
382        if ( !logListener || ( typeof logListener !== "function" ) )
383        {
384            // Do something here after listener has been removed
385            // React to error here
386        }
387    }
388    catch (e) {
389        // Handle exception
390    }
throw new Error( "AppLog.stopLogListener Error: logListener is not a function" );

if ( AppLog.logListeners_internal.length > 0 )
{
    var foundListener = false;
    for ( var i = 0; i < AppLog.logListeners_internal.length; i++ )
    {
        var listener = AppLog.logListeners_internal[i];
        if ( listener.listener === logListener &&
            listener.containingObject ===
            containingObject )
        {
            foundListener = true;
            AppLog.logListeners_internal.splice(i, 1);
        }
    }
    if ( !foundListener )
    {
        throw new Error( "AppLog.stopLogListener Error: logListener was not found. Nothing removed" );
    }
}

if ( AppLog.logListeners_internal.length === 0 )
{
    cordova.exec( null, null, "AppLog", "stopLogListener", [] );
}
else
setTimeout( successCB(), 0 );

} // close else

throw new Error( "AppLog.stopLogListener Error: There are no registered listeners" );

} // close catch

setTimeout( errorCB( {errorCode : AppLog.ERR_UNKNOWN, description : ex.message} ), 0 );

} // close function

/**
 * @private
 * @param {anonymous.logListener} logListener The callback to register. This will be invoked when new entries are added to the log.
 * @param {Object} [containingObject] Object containing the definition for logListener. If a log listener callback function
 */

AppLog.logListenerCallBack = function( logListener, containingObject ) {

    this.containingObject = containingObject;
    this.listener = logListener;

} // close function

AppLog.logListeners_internal = [];

/**
 * @private
* @param {AppLog.LogEntry} logEntry Object for each line in the app log. */

AppLog.logListener_internal = function(logEntry) {
  if (AppLog.logListeners_internal.length === 0) {
    return;
  }

  // The incoming date is number of millisecond, we need to change it to real JavaScript Date type.
  var dateInJS = new Date(logEntry.date);

  for (var i = 0; i < AppLog.logListeners_internal.length; i++) {
    var logCallBack = AppLog.logListeners_internal[i];
    var containingObject = logCallBack.containingObject;
    var callbackFunction = logCallBack.listener;

    if (containingObject !== null && containingObject !== undefined)
      callbackFunction.call(containingObject, dateInJS, logEntry.statusCode, logEntry.message);
    else
      callbackFunction(dateInJS, logEntry.statusCode, logEntry.message);
  }
}(AppLog, window);
/**
 * Used to group anonymous objects and callback functions used as method parameters. Methods and fields in this
 * namespace cannot be instantiated. Used for API docs generation only.
 * @namespace
 */

anonymous = (typeof anonymous === "undefined" || !anonymous) ? {} : anonymous;

/**
 * Callback function that will be invoked with all the entries in the app log. There will be one
 * @link AppLog.LogEntry object for each line in the app log.
 * Log entries can be retrieved with @link AppLog.getLogEntries).
 * @function
 */

/**
 * Callback function that will be invoked when @link AppLog.getLogEntries} fails.
 * @function
 */
* Callback function that will be invoked upon successfully starting a log listener via `{@link AppLog.startLogListener}`, or upon successfully removing a log listener via `{@link AppLog.stopLogListener}`.

@name anonymous.startOrStopLogListenerSuccessCallback
@function

/*

/**
 * Callback function that will be invoked upon failure to start a log listener via `{@link AppLog.startLogListener}`, or upon failure to removing a log listener via `{@link AppLog.stopLogListener}`.

@name anonymous.startOrStopLogListenerErrorCallback
@function

*/

/**
 * Object used in `{@link anonymous.getLogEntriesErrorCallback} and `{@link anonymous.startOrStopLogListenerErrorCallback}` functions.

@class
@name anonymous.AppLogErrorCallbackParameter

@property {number} errorCode Predefined error code
@property {string} description The description of the error

*/

/**
* Callback function that will be invoked when events are logged to the app log. Log listeners can be added with `{@link AppLog.startLogListener}`.

* @name anonymous.logListener

* @param {Date} date The date of the log entry.

* @param {number} event The event ID of the log entry (will be one of the AppLog status events, or possibly a custom value).

* @param {string} message The string carrying the message of the log entry.

* @function

Plugins/HttpsProxy/datajs-https-proxy.js

// it is depending on Datajs.js and httpsproxy.js

/*
 * Sybase Hybrid App version 2.3.4
 * Sybase datajs integration with PhoneGap HTTPS proxy
 */

datajs-https-proxy.js

* This file will not be regenerated, so it is possible to modify it, but it is not recommended.

* Copyright (c) 2013 Sybase Inc. All rights reserved.

* The namespace for HTTP(S) proxy

* @namespace
HttpsConnection = (typeof HttpsConnection === "undefined" || !HttpsConnection) ? {} : HttpsConnection; // 'namespace'

(function(HttpsConnection, window, undefined) {

    /**
     * Generate an OData HttpClient object over https proxy of native platform.
     *
     * This object will re-direct all odata request to the http proxy because even with HTTP connection, there are
     * are some known issue by default setting since the application in device is cross server accessing the odata service.
     *
     * See: http://datajs.codeplex.com/discussions/396112 for details of the issue.
     *
     * Call this method normally on HTML page load event to replace the default odata HTTP client.
     *
     * @memberOf HttpsConnection
     * @public
     * @example
     * // Call datajs api without certificate, users could call just as normal by passing
     * // URL as first argument
     * var length = 0;
     * var updateUri = server + "/example.svc/ Categories(1)";
     *
     * OData.read(server + "/example.svc/Categories",
     * function (data, response) {
     * alert("length " + data.results.length);
     * length = data.results.length;
     * if ( length > 0 )
     * {
     * var updateRequest = {

* `requestUri: updateUri`,
* `method: "PUT"`,
* `data:
  * { Picture: new Date().getTime(),
     * Description: "Update Record",
     * CategoryName: "Updated Category",
     * CategoryID: 1
  }
* `OData.request(updateRequest,
  * function (data, response) {
    * alert("Response " +
      JSON.stringify(response));
    * },
  * function (err) {
    * alert("Error occurred " +
      err.message);
    * });
* `var length = 0;
* // However, to specify certificate source in the method
call, users need to pass in
* // the request object instead of URL,
* // and add the field "certificateSource" to the request
object.
var updateUri = server + "/example.svc/Categories(1)";

OData.read({ requestUri: server + "/example.svc/Categories", certificateSource : cert},
  function (data, response) {
    alert("length " + data.results.length);
    length = data.results.length;
    if ( length > 0 ) {
      var updateRequest = {
        requestUri: updateUri,
        certificateSource : cert,
        method: "PUT",
        data: {
          Picture: new Date().getTime(),
          Description: "Update Record",
          CategoryName: "Updated Category",
          CategoryID: 1
        }
      };

      OData.request(updateRequest,
        function (data, response) {
          alert("Response " + JSON.stringify(response));
        },
        function (err) {
          alert("Error occurred " + err.message);
        }
      );
    }
  });
Develop Hybrid Apps Using Third-party Web Frameworks

```javascript
104          * },
105          * function (err) {
106          *     alert("Error occurred " + err.message);
107          * });
108          */
109         HttpsConnection.generateODataHttpClient = function () {
110             if ( HttpsConnection.sendRequest && !
111                 OData.defaultHttpClient.HttpsConnectionWrapper) {
112                 OData.defaultHttpClient = {
113                     HttpsConnectionWrapper: true,
114                     request: function (request, success, error)
115                         var url, requestHeaders, requestBody,
116                         statusCode, statusText, responseHeaders;
117                         var responseBody, requestTimeout, requestUserName,
118                         requestPassword, requestCertificate;
119                         var client, result;
120                         url = request.requestUri;
121                         requestHeaders = request.headers;
122                         requestBody = request.body;
123                         var successCB = function( data ) {
124                             var response = {
125                                 requestUri: url,
126                                 statusCode: data.status,
127                                 statusText: data.status,
128                                 body: (data.responseText ?
129                                     data.responseText : data.responseBase64)
130                             };
131                         if (response.statusCode >= 200 &&
```
if ( success ) {
    success(response);
} } else {
    if ( error ) {
        error({ message: "HTTP request failed", request: request, response: response });
    } 
    
} 

var errorCB = function( data ) { 
    if ( error ) {
        error({message: data});
    } 
};

if ( request.timeoutMS ) {
    requestTimeout = request.timeoutMS / 1000;
}

if ( request.certificateSource ) {
    requestCertificate = request.certificateSource;
}

if ( request.user ) {
    requestUserName = request.user;
}

if ( request.password ) {
    requestPassword = request.password;
client = HttpsConnection.sendRequest(request.method || "GET", url, requestHeaders, requestBody, successCB, errorCB, requestUserName, requestPassword, requestTimeout, requestCertificate );

result = {};
result.abort = function () {
    client.abort();
    if ( error ) {
        error({ message: "Request aborted" });
    }
}
return result;

}(HttpsConnection, window);

Plugins/HttpsProxy/https-proxy.js

/*
 * Sybase Hybrid App version 2.3.4
 * Sybase PhoneGap HTTPS proxy
 * https-proxy.js
 * This file will not be regenerated, so it is possible to modify it, but it is not recommended.
 */
(function (window, undefined) {
    if (!window.HttpsConnection) {
        window.HttpsConnection = {};  
    }

    var HttpsConnection = window.HttpsConnection;

    /**
     * Constant definitions for registration methods
     */

    /**
     * Constant indicating the operation failed with unknown error. Used in ){@link anonymous.sendRequestErrorCBParameter}
     * @type number
     */
    HttpsConnection.ERR_UNKNOWN = -1;

    /**
     * Constant indicating the operation has invalid parameter. Used in ){@link anonymous.sendRequestErrorCBParameter}
     * @type number
     */
    HttpsConnection.ERR_INVALID_PARAMETER_VALUE = -2;

    /**
     * Constant indicating the operation failed because of missing parameter. Used in ){@link anonymous.sendRequestErrorCBParameter}
     */
* @type number

**

HttpConnection.ERR_MISSING_PARAMETER = -3;

/**

* Constant indicating there is no such cordova action for the current service. Used in {@link anonymous.sendRequestErrorCBParameter}

* @type number

/**

HttpConnection.ERR_NO_SUCH_ACTION = -100;

/**

* Constant indicating certificate from file keystore is not supported on current platform. Used in {@link anonymous.sendRequestErrorCBParameter}

* @type number

/**

HttpConnection.ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED  = -101;

/**

* Constant indicating certificate from system keystore is not supported on current platform. Used in {@link anonymous.sendRequestErrorCBParameter}

* @type number

/**

HttpConnection.ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED = -102;

/**

* Constant indicating certificate from Afaria server is not supported on current platform. Used in {@link anonymous.sendRequestErrorCBParameter}

* @type number

/**

HttpConnection.ERR_AFARIA_CERTIFICATE_SOURCE_UNSUPPORTED = -103;
* Constant indicating the certificate with given alias could not be found. Used in `{@link anonymous.sendRequestErrorCBParameter}

* @type number

```typescript
HttpsConnection.ERR_CERTIFICATE_ALIAS_NOT_FOUND = -104;
```

* Constant indicating the certificate file could not be found. Used in `{@link anonymous.sendRequestErrorCBParameter}

* @type number

```typescript
HttpsConnection.ERR_CERTIFICATE_FILE_NOT_EXIST = -105;
```

* Constant indicating incorrect certificate file format. Used in `{@link anonymous.sendRequestErrorCBParameter}

* @type number

```typescript
HttpsConnection.ERR_CERTIFICATE_INVALID_FILE_FORMAT = -106;
```

* Constant indicating failed in getting certificate. Used in `{@link anonymous.sendRequestErrorCBParameter}

* @type number

```typescript
HttpsConnection.ERR_GET_CERTIFICATE_FAILED = -107;
```

* Constant indicating the provided certificate failed validation on server side. Used in `{@link anonymous.sendRequestErrorCBParameter}

* @type number

```typescript
HttpsConnection.ERR_CLIENT_CERTIFICATE_VALIDATION = -108;
```

* Constant indicating the server certificate failed validation on client side. Used in `{@link anonymous.sendRequestErrorCBParameter}`
* @type number

```
HttpsConnection.ERR_SERVER_CERTIFICATE_VALIDATION = -109;
/**
* Constant indicating the server request failed. Used in
{@link anonymous.sendRequestErrorCB}
* @type number
*/
HttpsConnection.ERR_SERVER_REQUEST_FAILED = -110;
/**
* Constant indicating timeout error while connecting to
the server. Used in {@link anonymous.sendRequestErrorCB}
* @type number
*/
HttpsConnection.ERR_HTTP_TIMEOUT = -120;
/**
* Create certificate source description object for
certificates from a keystore file.
* <b> Not supported on Blackberry platform </b>
* @class
* @memberOf HttpsConnection
* @public
* @param {string} Path Path of the keystore file. For iOS
client, it first tries to load the
  * relative file path from application's
  * Documents folder; if it fails, then tries
  * to load the file path from application's
  * main bundle. In addition, before trying
  * to load the certificate from file system,
  * iOS client first checks whether the
  * specified certificate key already exists
  * in the key store, if so, it just loads
  * the existing certificate from key store,
  * instead of loading the certificate from
```
* file system.
* @param {string} Password Password of the keystore.
* @param {string} CertificateKey An unique key that will be used to locate the certificate.
*/
HttpsConnection.CertificateFromFile = function (Path, Password, CertificateKey) {
    this.Source = "FILE";
    this.Path = Path;
    this.Password = Password;
    this.CertificateKey = CertificateKey;
};

/**
 * Create certificate source description object for certificates from Afaria.
 * @class
 * @memberOf HttpsConnection
 * @public
 * @param {string} CN Common Name (CN) for CA/SCEP protocol. For iOS, the retrieved certificate is stored in the key store with the common name as the certificate key, the following requests for the same common name will just load the saved certificate from key store, instead of sending a new request to Afaria server.
 * @param {string} [ChallengeCode] Challenge code for CA/SCEP protocol.
 */
HttpsConnection.CertificateFromAfaria = function (CN, ChallengeCode) {
    this.Source = "AFARIA";
    this.CN = CN;
    this.ChallengeCode = ChallengeCode;
};
Create certificate source description object for certificates from system keystore (Keystore in BB, Keychain in iOS and Android).

* The certificateKey is not used on the BB platform. BB will prompt the user to select a certificate if a certificate was not already used for the server connection.

@class

@memberOf HttpsConnection

@public

@param {string} CertificateKey An unique key that will be used to locate the certificate. Not used in BB platform.

HttpsConnection.CertificateFromStore = function (CertificateKey) {
    this.Source = "SYSTEM";
    this.CertificateKey = CertificateKey;
};

HttpsConnection.MSG_MISSING_PARAMETER = "Missing a required parameter: ";

HttpsConnection.MSG_INVALID_PARAMETER_VALUE = "Invalid Parameter Value for parameter: ";

/**
 * @private

@param {Object} [certSource] Certificate description object. It can be one of {@link HttpsConnection.CertificateFromFile}, {@link HttpsConnection.CertificateFromStore}, or {@link HttpsConnection.CertificateFromAfaria}.

@param {function} errorCB Callback method upon failure.

*/
HttpsConnection.validateCertSource = function(certSource, errorCB) {
    if (!certSource) {
        // The certificate is not present, so just ignore it.
        return true;
    }

    // errorCB required.
    // First check this one. We may need it to return errors
    if (errorCB && (typeof errorCB !== "function")) {
        console.log("HttpsConnection Error: errorCB is not a function");
        return false;
    }

    try {
        // First check whether it is an object
        if (typeof certSource !== "object") {
            errorCB({errorCode : HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description : HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "certSource"]);
            return false;
        }

        if (certSource.Source === "FILE") {
            if (!certSource.Path) {
                errorCB({errorCode : HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description : HttpsConnection.MSG_MISSING_PARAMETER + "keystore path"]);
                return false;
            }

            if (typeof certSource.Path !== "string") {
                // errorCB required.
                // First check this one. We may need it to return errors
                return false;
            }

            if (certSource.Path !== "FILE") {
                if (!certSource.Path) {
                    errorCB({errorCode : HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description : HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "certSource"]);
                    return false;
                }

                if (typeof certSource.Path !== "string") {
                    // errorCB required.
                    // First check this one. We may need it to return errors
                    return false;
                }
            }
        }
    }
}
errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "keystore path"});

return false;

if (!certSource.Password) {
  errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_MISSING_PARAMETER + "keystore password"});
  return false;
}

if (typeof certSource.Password !== "string") {
  errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "keystore password"});
  return false;
}

if (!certSource.CertificateKey) {
  errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_MISSING_PARAMETER + "certificate key"});
  return false;
}

if (typeof certSource.CertificateKey !== "string") {
  errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "certificate key"});
  return false;
}

} else if (certSource.Source === "SYSTEM") {
if (!certSource.CertificateKey) {
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_MISSING_PARAMETER + "certificate key"});
    return false;
}

if (typeof certSource.CertificateKey !== "string") {
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "certificate key"});
    return false;
}

} else if (certSource.Source === "AFARIA") {
    if (!certSource.CN) {
        errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_MISSING_PARAMETER + "common name"});
        return false;
    }

    if (typeof certSource.CN !== "string") {
        errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "common name"});
        return false;
    }

    if (!certSource.ChallengeCode) {
        errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_MISSING_PARAMETER + "Afaria challenge code"});
        return false;
    }

    if (!certSource.ChallengeCode) {
        errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_MISSING_PARAMETER + "Afaria challenge code"});
        return false;
    }
}

if (typeof certSource.ChallengeCode !== "string") {
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "Afaria challenge code"});
    return false;
}

}
if (typeof certSource.ChallengeCode !== "string") {
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "Afaria challenge code");
    return false;
} else {
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "certSource");
    return false;
}
return true;
} catch (ex) {
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "certSource");
}
};

/**
 * Send a HTTP(S) request to a remote server.
 * @memberOf HttpsConnection
 * @public
 * @param {string} method Standard HTTP request method name.
 * @param {string} url The http url with format http(s):// [user:password]@hostname[:port]/path.
 * @param {Object} header HTTP header to be sent to server. This is an Object. Can be null.
 * @param {string} requestBody Data to be sent to server with the request. It's a string value. Can be null.
 * @param {anonymous.sendRequestSuccessCB} successCB Callback method upon success.
* @param {anonymous.sendRequestErrorCB} errorCB Callback method upon failure.

* @param {string} [user] User ID for basic authentication.

* @param {string} [password] User password for basic authentication.

* @param {number} [timeout] Timeout setting in seconds.

* @param {Object} [certSource] Certificate description object. It can be one of {@link HttpsConnection.CertificateFromFile},

* {@link HttpsConnection.CertificateFromStore}, or {@link HttpsConnection.CertificateFromAfaria}.

* @returns {anonymous.abort} A JavaScript function object to cancel the operation.

* @example

* // To send a post request to server, call the method

* HttpsConnection.sendRequest("POST", "http://www.google.com", null, "THIS IS THE BODY", function (data) {

*     alert("Status: " +
*     JSON.stringify(data.status));

*     alert("Headers: " +
*     JSON.stringify(data.headers));

*     alert("Response: " +
*     JSON.stringify(data.response));

* }, function (data) {

*     alert("Failed: " + JSON.stringify(data));});

* // To send a post request to server with headers, call the method

* HttpsConnection.sendRequest("POST", url, {HeaderName: "Header value"}, "THIS IS THE BODY", successCB, errorCB);

* // To send a post request to server with basic authentication, call the method

* HttpsConnection.sendRequest("POST", url, headers, "THIS IS THE BODY", successCB, errorCB, "username", "password");

* // To send a post request to server with mutual authentication, call the method

* HttpsConnection.sendRequest("POST", "https://hostname", headers, "THIS IS THE BODY", successCB, errorCB, null,
null, 0, new CertificateFromFile("/mnt/sdcard/my.keystore", "password", "mykey");

HttpsConnection.sendRequest = function (method, url, header, requestBody, successCB, errorCallback, user, password, timeout, certSource) {

    // errorCallback required.
    // First check this one. We may need it to return errors
    if (!errorCallback || (typeof errorCallback !== "function")) {
        console.log("HttpsConnection Error: errorCallback is not a function");
        // if error callback is invalid, throw an exception to notify the caller
        throw new Error("HttpsConnection Error: errorCallback is not a function");
        return;
    }

    // method required
    if (!method) {
        console.log("HttpsConnection Error: method is required");
        errorCallback({errorCode: HttpsConnection.ERR_MISSING_PARAMETER, description: HttpsConnection.MSG_MISSING_PARAMETER + "method"});
        return;
    }

    // We only support GET, POST, HEAD, PUT, DELETE method
    if (method !== "GET" && method !== "POST" && method !== "HEAD" && method !== "PUT" && method !== "DELETE") {
        console.log("Invalid Parameter Value for parameter: " + method);
    }

    //
errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "method"});

return;

// url required
if (!url) {
    console.log("HttpsConnection Error: url is required");
    errorCB({errorCode: HttpsConnection.ERR_MISSING_PARAMETER, description: HttpsConnection.MSG_MISSING_PARAMETER + "url"});
    return;
}

// successCB required
if (!successCB) {
    console.log("HttpsConnection Error: successCB is required");
    errorCB({errorCode: HttpsConnection.ERR_MISSING_PARAMETER, description: HttpsConnection.MSG_MISSING_PARAMETER + "successCB"});
    return;
}

if (typeof successCB !== "function") {
    console.log("HttpsConnection Error: successCB is not a function");
    errorCB({errorCode: HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description: HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "successCB"});
    return;
}

if (user && typeof user !== "string") {
errorCB({errorCode : HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description : HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "user"});

return;

if (password && typeof password !== "string") {
    errorCB({errorCode : HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description : HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "password"});
    return;
}

if (timeout && typeof timeout !== "number") {
    errorCB({errorCode : HttpsConnection.ERR_INVALID_PARAMETER_VALUE, description : HttpsConnection.MSG_INVALID_PARAMETER_VALUE + "timeout"});
    return;
}

if (!HttpsConnection.validateCertSource(certSource, errorCB)) {
    return;
}

try {
    var client = new HttpsConnection.Client(method, url, header, requestBody, successCB, errorCB, user, password, timeout, certSource);
    return client.send();
} catch (ex){
    errorCB({errorCode : HttpsConnection.ERR_UNKNOWN, description : ex.message});
}
};
/**
 * Send a HTTP(S) GET request to a remote server.
 * @memberOf HttpsConnection
 * @public
 * @param {string} url The http url with format http(s):// [user:password]@hostname[:port]/path.
 * @param {Object} header HTTP header to be sent to server. This is an Object. Can be null.
 * @param {anonymous.sendRequestSuccessCB} successCB Callback method upon success.
 * @param {anonymous.sendRequestErrorCB} [errorCB] Callback method upon failure.
 * @param {string} [user] User ID for basic authentication.
 * @param {string} [password] User password for basic authentication.
 * @param {number} [timeout] Timeout setting in seconds.
 * @param {Object} [certSource] Certificate description object. It can be one of {@link HttpsConnection.CertificateFromFile},
 * {@link HttpsConnection.CertificateFromStore}, or {@link HttpsConnection.CertificateFromAfaria}.
 * @returns {anonymous.abort} A JavaScript function object to cancel the operation.
 * @example
 * // To send a get request to server, call the method
 * HttpsConnection.get("http://www.google.com", null, function (data) {
 *     alert("Status: " + JSON.stringify(data.status));
 *     alert("Headers: " + JSON.stringify(data.headers));
 *     if (data.responseText){
 *         alert("Response: " + JSON.stringify(data.responseText));
 *     }
 * },

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps 553
384 * function (error) {
385   *     alert("Failed: " +
386   *         JSON.stringify(error));
387   * }
388
389   * // To send a get request to server with headers, call
390   * // the method
391   * HttpsConnection.get(url, {HeaderName : "Header value"},
392   * successCB, errorCB);
393
394   * // To send a get request to server with basic
395   * // authentication, call the method
396   * HttpsConnection.get(url, headers, successCB, errorCB, "username", "password");
397
398   * // To send a get request to server with mutual
399   * // authentication, call the method
400   * HttpsConnection.get("https://hostname", headers, successCB, errorCB, null, null, 0,
401   *     new CertificateFromFile("/mnt/sdcard/my.p12", "password", "mykey");
402
403   */
404
405   HttpsConnection.get = function (url, header, successCB, errorCB, user, password, timeout, certSource){
406     return HttpsConnection.sendRequest("GET", url, header, null, successCB, errorCB, user, password, timeout, certSource);
407   }
408
409   /**
410    * Delete cached certificate from keychain. iOS client
411    * will always try the cached certificate first if it is available
412    * before requesting the certificate from
413    * afaria server or loading the certificate from file
414    * system. In case the cached certificate is no longer valid, use this
415    * method to delete it from keychain
416
417    * <b> Only supported by iOS platform </b>
418
419    * @memberOf HttpsConnection
420    * @public
421
422    * @param {anonymous.sendRequestSuccessCB} successCB
423    * Callback method upon success.
424
425    * @param {anonymous.sendRequestErrorCB} [errorCB]
426    * Callback method upon failure.
* @param {string} certificateKey The key of the certificate to be deleted.

/**
 * @private
 * @param {string} method Standard HTTP request method name.
 * @param {string} url The http url with format http(s)://[user:password]@hostname[:port]/path.
 * @param {Object} header HTTP header to be sent to server. This is an Object. Can be null.
 * @param {string} requestBody Data to be sent to server with the request. It’s a string value. Can be null.
 * @param {anonymous.sendRequestSuccessCB} successCB Callback method upon success.
 * @param {anonymous.sendRequestErrorCB} errorCB Callback method upon failure.
 * @param {string} [user] User ID for basic authentication.
 * @param {string} [password] User password for basic authentication.
 * @param {number} [timeout] Timeout setting in seconds.
 * @param {Object} [certSource] Certificate description object. It can be one of {@link HttpsConnection.CertificateFromFile},
 * {@link HttpsConnection.CertificateFromStore}, or {@link HttpsConnection.CertificateFromAfaria}.
 * @returns {anonymous.abort} A JavaScript function object to cancel the operation.
 */
HttpsConnection.Client = function (method, url, header, requestBody, successCB, errorCB, user, password, timeout, certSource)
{
    // ios plugin parameter does not support object type, convert Header and CertSource to JSON string
    if (device.platform === "iOS" || (device.platform && device.platform.indexOf("iP") === 0 ))
    {
        if (header) {
            header = JSON.stringify(header);
        }
        if (certSource) {
            certSource = JSON.stringify(certSource);
        }
    }

    this.Method = method;
    this.Url = url;
    this.Header = header;
    this.RequestBody = requestBody;
    this.SuccessCB = successCB;
    this.ErrorCB = errorCB;
    this.User = user;
    this.Password = password;
    this.Timeout = timeout;
    this.CertSource = certSource;
    this.IsAbort = false;

    this.abort = function ()
    {
        this.IsAbort = true;
    };
}
this.send = function ()
{
    var args = [this.Method, this.Url, this.Header,
                this.RequestBody, this.User, this.Password, this.Timeout,
                this.CertSource];

    var me = this;

    var successCallBack = function(data)
    {
        if (me.IsAbort === true)
        {
            return;
        }

        successCB(data);
    };

    var errorCallBack = function(data)
    {
        if (me.IsAbort === true)
        {
            return;
        }

        errorCB(data);
    };

    cordova.exec(successCallBack, errorCallBack,
                 "HttpsProxy", "sendRequest", args);

    return this.abort;
};
Anonymous

Used to group anonymous objects and callback functions used as method parameters. Methods and fields in this namespace cannot be instantiated. Used for API docs generation only.

@namespace

Anonymous = (typeof anonymous === "undefined" || !anonymous) ? {} : anonymous;  // SUP 'namespace'

/**
 * Callback function that will be invoked when
 * the HttpsConnection.get() / sendRequest() succeeded.
 */

* @name anonymous.sendRequestSuccessCB

* @param {anonymous.sendRequestSuccessCBParameter} data The response data object.

* @function

*/

/**
 * Callback function that will be invoked when
 * the HttpsConnection.get() / sendRequest() failed.
 */

* @name anonymous.sendRequestErrorCB

* @param {anonymous.sendRequestErrorCBParameter} data The error object.

* @function

*/


/**
 * Object used in {@link anonymous.sendRequestSuccessCB} function.
 *
 * @class
 * @name anonymous.sendRequestSuccessCBParameter
 *
 * @property {number} status The HTTP status code
 * @property {object} headers An object that contains header_key = value pairs.
 * @property {string} [responseText] The text response. This parameter is present only if the response is a text response.
 * @property {string} [responseBase64] Base64 encoded representation of the binary response. This parameter is included only if the response is a binary response.
 * @property {object} [clientError] An optional object that contains the authentication error. It is an object of {@link anonymous.sendRequestErrorCBParameter}.
 */

/**
 * Object used in {@link anonymous.sendRequestErrorCB} function.
 *
 * @class
 * @name anonymous.sendRequestErrorCBParameter
 *
 * @property {number} errorCode Predefined error code
 * @property {string} description The description of the error
 * @property {number} [nativeErrorCode] The native error code reported from Afaria, device, etc (optional)
 */

/**
 * JavaScript function to abort the HTTP(S) request
 *
 * @name anonymous.abort
 */
SUPStorage.js

/* Sybase Hybrid App version 2.3.4 */

* SUPStorage.js

* This file will not be regenerated, so it is possible to modify it, but it
* is not recommended.

* Copyright (c) 2012 Sybase Inc. All rights reserved.

*/

/**
* The namespace for the Hybrid Web Container javascript
* @namespace
*/

hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc; // SUP 'namespace'

/**
* Access the storage functions, which allow you to specify a cache that stores results from online requests.
* These functions give you the ability to:
* Name the cached result sets
* Enumerate the cached result sets
* Read, delete, and modify cached contents individually for each cached result set
Cached result sets must be stored as strings (before deserialization to an xmlWorkflowMessage structure).

* Creates a SUPStorage with the specified storeName. Provides encrypted storage of name value pairs. Results from online requests are one example.

Strings stored in SUPStorage are encrypted and persisted to survive multiple invocations of the mobile workflow application.

@desc Storage
@memberOf hwc
@constructor
@param {string} store the store name
@example
var store1 = new hwc.SUPStorage("one");

hwc.SUPStorage = function(store) {
    this.bForSharedStorage = false;
    this.store = store ? store : "";
};

/*
* Gets the number of available keys in this object. The keys themselves may be
* retrieved using key().
* @desc Storage
* @memberOf hwc.SUPStorage
* @public
* @example
*/

// Create the SUP Storage
var store = new hwc.SUPStorage("one");
* store.setItem("foo", "bar"); // add an item.
* store.setItem("foo1", "bar"); // add an item.
* store.setItem("foo2", "bar"); // add an item.
  
* var result = store.length; // result = 3
*/

  
hwc.SUPStorage.prototype.length = function() {
    var response;
    hwc.traceEnteringMethod("hwc.SUPStorage.length");
    try {
      if (hwc.isWindowsMobile() || hwc.isIOS()) {
        response = hwc.getDataFromContainer("workflowstorage",
            
            
            "&command=length&shared=" + this.bForSharedStorage + "&store=" +
            
            encodeURIComponent(this.store));

        return parseInt(response, 10);
      }
    } catch (e) { 
    }

    return _SharedStorage.length(hwc.versionURLParam);
  }

  
  
  
  

} finally {
  hwc.traceLeavingMethod("hwc.SUPStorage.length");
}

/**
 * Returns the key at the supplied index. Keys are guaranteed to remain
at the same index until a modification is made.
*
* @desc Storage
* @public
* @memberOf hwc.SUPStorage
* @param {Integer} index 0-based index to the key. Must be less than the value retrieved
* @returns {string} The key, or null if the index is invalid.
* @example
* // Create the SUP Storage
* var store = new hwc.SUPStorage("one");
* store.setItem("foo", "bar"); // add an item.
* var result = store.key(0); // will returns "foo".
*/

hwc.SUPStorage.prototype.key = function(index) {
    var key, isExist;
    hwc.traceEnteringMethod("hwc.SUPStorage.key");
    try {
        if (null === index) {
            return null;
        }
        if (hwc.isWindowsMobile() || hwc.isIOS()) {
            key = hwc.getDataFromContainer("workflowstorage", 
                
            );
            if (key === null || typeof key === 'undefined' || key === ") {
                isExist = 
            hwc.getDataFromContainer("workflowstorage", 
                        
                    
                    
                
        } else {
            hwc.getDataFromContainer("workflowstorage", 
                
                    
                
            
            
            
        }
    }
};
"&store=" + encodeURIComponent(this.store) + 
"&index=" + encodeURIComponent(index));

//WM returns empty string if an item does not exist or if the value is empty string

//call exist to distinguish this

if (isExist == "true") {
  key = "";
}

else {
  key = null;
}

else {
  if (this.bForSharedStorage) {
    key = _SharedStorage.key(index,
    hwc.versionURLParam);
  }
  else {
    key = SUPStorage.key(this.store, index);
  }

  if (key === null || typeof key === 'undefined') {
    return null;
  } else {
    return key + "";
  }
}

} finally {
  hwc.traceLeavingMethod("hwc.SUPStorage.key");
}

};
function checkNull(input) {
    if (null === input) {
        input = "";
    }
    return input;
}

hwc.SUPStorage.prototype.getItem = function(key) {
171         var value, isExist;
172         key = key ? key : "";
173
174         hwc.traceEnteringMethod("hwc.SUPStorage.getItem");
175         try {
176             if (hwc.isWindowsMobile() || hwc.isIOS()) {
177                 value = hwc.getDataFromContainer("workflowstorage",
178                         ";" + this.bForSharedStorage +
179                         ";store=" + encodeURIComponent(this.store) +
180                         ";key=" + encodeURIComponent(key));
181             }
182             else {
183                 if (this.bForSharedStorage) {
184                     value = _SharedStorage.getItem(key,
185                         hwc.versionURLParam);
186                 }
187             }
188         }
189         catch (e) {
190             isExist = hwc.getDataFromContainer("workflowstorage",
191                         ";" + this.bForSharedStorage +
192                         ";store=" + encodeURIComponent(this.store) +
193                         ";key=" + encodeURIComponent(key));
194             //WM returns empty string if an item does not
195             //exist or if the value is empty string
196             //call exist to distinguish this
197             if (isExist == "true") {
198                 value = "";
199             }
200             else {
201                 value = null;
202             }
203         }
204         }
205         }
else {
    value = SUPStorage.getItem(this.store, key);
}

if (value === null || typeof value === 'undefined') {
    return null;
} else {
    return value + "";
}

} finally {
    hwc.traceLeavingMethod("hwc.SUPStorage.getItem");
}

/**
 * A constant for the maximum length for a string being stored on BB7
 * BB7 cannot handle strings with length longer than 524000
 * This restriction applies to real devices as well as simulators.
 */

hwc.SUPStorage.BB7_MAX_STRING_STORAGE_LENGTH = 524000;

/**
 * Sets the value associated with a specified key. This replaces the key's
 * previous value, if any.
 */

@desc Storage

@memberOf hwc.SUPStorage
*param {string} key String key corresponding to the value.

*param {string} value String value to store.

// Create the SUP Storage

var store = new hwc.SUPStorage("one");

store.setItem("foo", "bar"); // add an item.

```
233 hwc.SUPStorage.prototype.setItem = function(key, value) {
234     var result;
235     hwc.traceEnteringMethod("hwc.SUPStorage.setItem");
236     key = key ? key : "";
237     value = value ? value : "";
238     try {
239         if (hwc.isWindowsMobile() || hwc.isIOS()) {
240             hwcpostDataToContainer("workflowstorage", "command=setItem&store=" + encodeURIComponent(this.store) + 
241                 "&shared=" + this.bForSharedStorage + 
242                 
243     throw new hwc.SUPStorageException(hwc.SUPStorageException.MAX_SIZE_REACHED, "SUP storage maximum size reached - maximum length of string to store on BB7 is 524000 but attempted to store string of length " + value.length);
244         } else {
245             if (hwc.isBlackBerry7() && value.length > hwc.SUPStorage.BB7_MAX_STRING_STORAGE_LENGTH) {
246             throw new hwc.SUPStorageException(hwc.SUPStorageException.MAX_SIZE_REACHED, "SUP storage maximum size reached - maximum length of string to store on BB7 is 524000 but attempted to store string of length " + value.length);
247         }
248         }
249     } else {
250         result = SUPStorage.setItem(this.store, key, value);
```
252                 }
253                 if (result !== 0) {
254                     throw new hwc.SUPStorageException(result, "SUP storage maximum size reached");
255                 }
256             }
257         } finally {
258             hwc.traceLeavingMethod("hwc.SUPStorage.setItem");
259         }
260     }
261     
262     /**
263     * Removes the key and its associated value from this object. If the
264     * key does not exist, has no effect.
265     *
266     * @desc Storage
267     * @memberOf hwc.SUPStorage
268     * @param {string} key String key to remove.
269     * @example
270     * // Create the SUP Storage
271     * var store = new hwc.SUPStorage("one");
272     * store.setItem("foo", "bar"); // add an item.
273     * store.removeItem("foo");
274     * result = store.getItem("food"); // will be null.
275     */
276     hwc.SUPStorage.prototype.removeItem = function(key) {
277         hwc.traceEnteringMethod("hwc.SUPStorage.removeItem");
278         try {
279             key = key ? key : "";
280             if (hwc.isWindowsMobile() || hwc.isIOS()) {
281                 hwc.getDataFromContainer("workflowstorage", "&command=removeItem&shared=" + this.bForSharedStorage +
282                         "&store=" + encodeURIComponent(this.store) + 
283                         "&key=" + encodeURIComponent(key));
284             } else {
285                 if (this.bForSharedStorage) {
286                     _SharedStorage.removeItem(key,
287                     hwc.versionURLParam);
288                 }
289                 else {
290                     SUPStorage.removeItem(this.store, key);
291                 }
292             } finally {
293             hwc.traceLeavingMethod("hwc.SUPStorage.removeItem");
294         }
295     }
296
297     /**
298     * Removes all key/value pairs from this object.
299     * @desc Storage
300     * @memberOf hwc.SUPStorage
301     */
302     hwc.SUPStorage.prototype.clear = function() {
303         hwc.traceEnteringMethod("hwc.SUPStorage.clear");
304         try {
305             if (hwc.isWindowsMobile() || hwc.isIOS()) {
306                 hwc.getDataFromContainer("workflowstorage",
307                     ",command=clear&shared=" + this.bForSharedStorage + 
308                     "&store=" +
309                     encodeURIComponent(this.store));
310             } else {
311                 if (this.bForSharedStorage) {
312                     
313                 }
314             }
315         } finally {
316             hwc.traceLeavingMethod("hwc.SUPStorage.clear");
317         }
318     }
_SharedStorage.clear(hwc.versionURLParam);
}
else {
    SUPStorage.clear(this.store);
}
}
} finally {
    hwc.traceLeavingMethod("hwc.SUPStorage.clear");
}
};
/**
 * Exception thrown when Storage space is exceeded.
 * @desc Storage
 * @constructor
 * @memberOf hwc
 * @param {Integer} code the error code
 * @param {string} message the error message.
 */
hwc.SUPStorageException = function(code, message) {
    this.code = code;
    this.message = message;
};
hwc.SUPStorageException.UNKNOWN_ERROR = 1;
hwc.SUPStorageException.MAX_SIZE_REACHED = 2;
hwc.SUPStorageException.SHARED_STORAGE_DISABLED = 3;
// shared storage key.
hwc.sharedStorageKey = undefined;
/**
Method to return the shared storage key defined for the hybrid app by designer. An empty string is returned if the shared storage function is disabled.

@desc Storage

@memberOf hwc

@returns {string} the shared storage key.

hwc.getSharedStorageKey = function() {
    if (hwc.sharedStorageKey === undefined) {
        var key = hwc.getQueryVariable("sharedStorageKey");
        hwc.sharedStorageKey = (key === undefined) ? "" : key;
    }
    return hwc.sharedStorageKey;
};

/**
 * Indicates whether the shared storage is enabled for the hybrid app.
 *
 * @desc Storage
 *
 * @memberOf hwc
 *
 * @returns {boolean} true if the shared storage is enabled; false otherwise.
 *
 */

hwc.isSharedStorageEnabled = function() {
    var key = hwc.getSharedStorageKey();
    if (key === undefined || key === "") {
        return false;
    }
    else {
        return true;
    }
};
/**
 * Constructs a new SUP shared storage. You can use the returned value to access the shared storage data with the exising SUPStorage interface, however, the operation only affects the items belonging to the specified shared storage key.
 * @classdesc
 * @memberOf hwc
 * @desc Storage
 */

hwc.SharedStorage = function() {
    if (hwc.isSharedStorageEnabled() === false ) {
        throw new hwc.SUPStorageException(hwc.SUPStorageException.SHARED_STORAGE_DISABLED, "Shared storage is disabled");
    }
    this.bForSharedStorage = true;
    this.store = "";
};

hwc.SharedStorage.prototype = new hwc.SUPStorage();

timezone.js
/*
 * Sybase Hybrid App version 2.3.4
 * Timezone.js
 */
Develop Hybrid Apps Using Third-party Web Frameworks

5        * This file will not be regenerated, so it is possible to
6        * modify it, but it
7        *
8        * Copyright (c) 2012 Sybase Inc. All rights reserved.
9        */
10
11/**
12* The namespace for the Hybrid Web Container javascript
13* @namespace
14*/
15
hwc = (typeof hwc === "undefined" || !hwc) ? {} : hwc;  // SUP 'namespace'
16
17(function(hwc, window, undefined) {
18
19/**
20* Returns the current locale. The platform's locale string
21* should be available. However, if it is
22* missing the function queries available JavaScript APIs for
23* a suitable value.
24* @desc Timezone
25* @memberOf hwc
26* @public
27* @returns {string} Returns a string containing the current
28* locale, or null if it is not available.
29* @example
30* var sLocale = hwc.getCurrentLocale();
31* /
32
hwc.getCurrentLocale = function() {
33    try {
if (hwc.lang) {
    return hwc.lang;
} else {
    if (navigator) {
        if (navigator.language) {
            if (hwc.isAndroid()) {
                return navigator.userAgent.match(/Android \d+(?:\d+){1,2}; [a-z]{2}-[a-z]{2}/).toString().match(/[a-zA-Z]{2}-[a-zA-Z]{2}/).toString();
            } else {
                return navigator.language;
            }
        } else if (navigator.browserLanguage) {
            return navigator.browserLanguage;
        } else if (navigator.systemLanguage) {
            return navigator.systemLanguage;
        } else if (navigator.userLanguage) {
            return navigator.userLanguage;
        }
    } finally {
        hwc.traceLeavingMethod("hwc.getCurrentLocale");
    }
}
64 * Returns a localized representation of the given Date object. Queries the platform OS for a locale-
65 * formatted date/time string.
66 * @desc Timezone
67 * @memberOf hwc
68 * @public
69 * @param {Date} date Date to be localized, initialized to some valid time.
70 * @returns {string} Returns a localized date/time string, or undefined if platform is unsupported.
71 * @example
72 * var sDT = hwc.getLocalizedDateTime( date );
73 *
74 */
75 hwc.getLocalizedDateTime = function( date ) {
76     var result, dMilliseconds, sTzId, response;
77     hwc.traceEnteringMethod("hwc.getLocalizedDateTime");
78     try {
79         if (hwc.isAndroid()) {
80             dMilliseconds = Date.UTC(date.getFullYear(),
81                 date.getMonth(), date.getDate(), date.getHours(), date.getMinutes(),
82                 date.getSeconds() );
83             sTzId = _HWC.getLocalizedDateTime( dMilliseconds )+ '';
84             result = sTzId;
85         }
86     } else if (hwc.isWindowsMobile()) {
87         // Feature was not needed on this platform
88         result = undefined;
89     } else if (hwc.isIOS()) {
90         dMilliseconds = Date.UTC(date.getFullYear(),
91             date.getMonth(), date.getDate(), date.getHours(), date.getMinutes(),
92             date.getSeconds() );
93         response = hwc.getDataFromContainer("tz",
94             ";" + dMilliseconds);
result = (response);

else if (hwc.isBlackBerry()) {

    dMilliseconds = Date.UTC(date.getFullYear(),
    date.getMonth(), date.getDate(), date.getHours(), date.getMinutes(),
    date.getSeconds() );

    sTzId = TimeZone.tzdatetime( dMilliseconds );

    result = sTzId;
}

else {

    result = undefined;
}

return result;

} finally {

    hwc.traceLeavingMethod("hwc.getLocalizedDateTime");

}

/**
 * Returns a localized representation of the given Date object. Queries the platform OS for a locale-
 * formatted date string.
 */

@desc Timezone

@memberOf hwc

@public

@param {Date} date Date to be localized, initialized to some valid time.

@returns {string} Returns a localized date string, or undefined if platform is unsupported.

@example

var sD = hwc.getLocalizedDate( date );

*/

hwc.getLocalizedDate = function( date ) {
var dMilliseconds, sTzId, response, result;

hwc.traceEnteringMethod("hwc.getLocalizedDate");

try {
    if (hwc.isAndroid()) {
        dMilliseconds = Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(), 12, 0, 0);
        sTzId = _HWC.getLocalizedDate( dMilliseconds ) + '';
        result = sTzId;
    }

    else if (hwc.isWindowsMobile()) {
        // Feature was not needed on this platform
        result = undefined;
    }

    else if (hwc.isiOS()) {
        dMilliseconds = Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(), 12, 0, 0);
        response = hwc.getDataFromContainer("tz", ";
        "&command=tzdate&time=" + dMilliseconds);
        result = (response);
    }

    else if (hwc.isBlackBerry()){
        dMilliseconds = Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(), 12, 0, 0);
        sTzId = TimeZone.tzdate( dMilliseconds );
        result = sTzId;
    }

    else {
        result = undefined;
    }

    return result;
}

finally {
    hwc.traceLeavingMethod("hwc.getLocalizedDate");
}
149     }
150
151     /**
152      * Returns a localized representation of the given Date
153      * object. Queries the platform OS for a locale-
154      * formatted time string.
155      * @desc Timezone
156      * @memberOf hwc
157      * @public
158      * @param {Date} date Date to be localized, initialized to
159      * some valid time.
160      * @returns {string} Returns a localized time string, or
161      * undefined if platform is unsupported.
162      * @example
163      * var sT = hwc.getLocalizedTime( date );
164      *
165      */
166     hwc.getLocalizedTime = function( date ) {
167         var dMilliseconds, sTzId, response, result;
168         hwc.traceEnteringMethod("hwc.getLocalizedTime");
169         try {
170             if (hwc.isAndroid()) {
171                 dMilliseconds = Date.UTC(date.getFullYear(),
172                                 date.getMonth(), date.getDate(),
173                                 date.getHours(), date.getMinutes(),
174                                 date.getSeconds() );
175                 sTzId = _HWC.getLocalizedTime( dMilliseconds ) + 
176                      ";
177                 result = sTzId;
178             }
179             else if (hwc.isWindowsMobile()) {
180                 // Feature was not needed on this platform
181                 result = undefined;
182             }
183             else if (hwc.isIOS()) {
184                 // Feature was necessary on this platform
185                 result = _HWC.getLocalizedTime( dMilliseconds ) + 
186                      ";
187                 result = sTzId;
188             }
189         }
190     }
```javascript
    dMilliseconds = Date.UTC(date.getFullYear(), 
    date.getMonth(), date.getDate(), date.getHours(), date.getMinutes(), 
    date.getSeconds() );

    response = hwc.getDataFromContainer("tz", 
    "&command=tztime&time=" + dMilliseconds);

    result = (response);

    } else if (hwc.isBlackBerry()){ 

        dMilliseconds = Date.UTC(date.getFullYear(), 
        date.getMonth(), date.getDate(), date.getHours(), date.getMinutes(), 
        date.getSeconds() );

        sTzId = TimeZone.tztime( dMilliseconds );

        result = sTzId;

    } else if (hwc.isWindows()){ 

        // For debugging on a browser of windows platform

        result = date.toString();

    } else { 

        result = undefined;

    }

    return result;

} finally {

    hwc.traceLeavingMethod("hwc.getLocalizedTime");

};

/**
 * Converts the given Date object to the device's local time, 
 * and returns the new Date.
 *
 * @desc Timezone
 * @memberOf hwc
 * @public
```
@param {Date} date Date to be converted, initialized to some valid UTC time.

* @returns {Date} Returns the converted Date object.

* @example

var localDate = hwc.convertUtcToLocalTime( date );

*/

hwc.convertUtcToLocalTime = function( date )
{
    hwc.traceEnteringMethod("hwc.convertUtcToLocalTime");
    try {
        var iMilliseconds, totalOffsetInMinutes, time, localDate;
        iMilliseconds = date.valueOf();
        totalOffsetInMinutes = hwc.getOffsetFromUTC( date );
        totalOffsetInMinutes = totalOffsetInMinutes * 60000;
        time = iMilliseconds + totalOffsetInMinutes;
        localDate = new Date();
        localDate.setTime( time );
        return localDate;
    } finally {
        hwc.traceLeavingMethod("hwc.convertUtcToLocalTime");
    }
};

/**
* Converts the given Date object to UTC time, and returns the new Date.
* @desc Timezone
* @memberOf hwc
* @public

* @param {Date} date Date to be converted, initialized to some valid local time.
* @returns {Date} Returns the converted Date object.
* @example
* var utcDate = hwc.convertLocalTimeToUtc( date );
* */

    hwc.convertLocalTimeToUtc = function( date )
{
    hwc.traceEnteringMethod("hwc.convertLocalTimeToUtc");
    try {
        var iMilliseconds, totalOffsetInMinutes, time, utcDate;
        iMilliseconds = date.valueOf();
        totalOffsetInMinutes =
        hwc.getOffsetFromUTC( date );
        totalOffsetInMinutes = totalOffsetInMinutes * 60000;
        time = iMilliseconds - totalOffsetInMinutes;
        utcDate = new Date();
        utcDate.setTime( time );
        return utcDate;
    } finally {
    hwc.traceLeavingMethod("hwc.convertLocalTimeToUtc");
    }
}

/**
* Returns the total offset (difference) between the given "local" time and UTC including any daylight
* savings offsets if applicable. Example: if the device was in London timezone (Gmt +1) and it is
* currently practicing DST, the function would return "120": 60 minutes normal offset plus 60 minutes
for its daylight savings offset.
* @desc Timezone
* @memberOf hwc
* @public
* @param {Date} date Date at which time to determine offset, initialized to some valid time.
* @returns {int} Returns the GMT offset in minutes.
* @example

```javascript
var totalOffset = hwc.getOffsetFromUTC(date);
```

```javascript
hwc.getOffsetFromUTC = function( date ) {
    var lMilliseconds, iMilliseconds, iMinutesOffset, response, dt,
        year, month, day, hour, minute, second, request, d,
        dMilliseconds, result;

    hwc.traceEnteringMethod("hwc.getOffsetFromUTC");
    try {
        if (hwc.isAndroid()) {
            lMilliseconds = date.getTime();
            iMinutesOffset = _HWC.getOffsetFromUTC(lMilliseconds);
            result = iMinutesOffset;
        }
        else if (hwc.isWindows())
            dt = new Date();
            iMinutesOffset = dt.getTimezoneOffset() * (-1);
            result = iMinutesOffset;
        else if (hwc.isWindowsMobile())
            {
```
// JavaScript's Date and WM's DateTime objects
differs in their base starting time

// and definition. It was necessary to pass a
"time" to the OS - see below comment

lMilliseconds = date.getTime();

// Rather than pass a date string (which might be in
a different locale format)

// the raw parameters of the particular "date" are
sent

// this also avoids a date string parse on the OS
side.

year = date.getFullYear();
month = date.getMonth() + 1;
day = date.getDate();
hour = date.getHours();
minute = date.getMinutes();
second = date.getSeconds();

request = "utcoffset=utcoffset&";
request += "year=";
request += year.toString();
request += "&";
request += "month=";
request += month.toString();
request += "&";
request += "day=";
request += day.toString();
request += "&";
request += "hour=";
request += hour.toString();
request += "&";
request += "minute=";
request += minute.toString();
request += "&";
request += "second=";
request += second.toString();

response = hwc postDataToContainer("tz", request);

d = response * 1;
iMinutesOffset = d;

result = iMinutesOffset;

}  
else if (hwc.isBlackBerry()){

  dMilliseconds = date.getTime();

  iMinutesOffset = TimeZone.totaloffset(dMilliseconds);

  result = iMinutesOffset;

}  
else if (hwc.isIOS()) {

  lMilliseconds = date.getTime();

  result = hwc.getDataFromContainer("tz", "&command=utcoffset&time=" + lMilliseconds);

}  
else {

  result = undefined;

}

return result;

} finally {

hwc.traceLeavingMethod("hwc.getOffsetFromUTC");

}

/**
   * Returns whether daylight savings rules are in effect for
   * the current timezone at the given time.
   *
   * @desc Timezone
   * @memberOf hwc
   */


348 * @public  
349 * @param {Date} date Date at which to determine whether  
350 * daylight savings is in effect.  
351 * @returns {boolean} Returns true iff daylight savings rules  
352 * are in effect at the given time in the  
353 * current timezone.  
354 * @example  
355 */  
356 hwc.isDstActiveAtGivenTime = function( date )  
357 {  
358     var lMilliseconds, iMilliseconds, iMinutesOffset,  
359         response, dt,  
360         year,month, day, hour, minute, second, request, d,  
361         dMilliseconds, result;  
362     hwc.traceEnteringMethod("hwc.isDstActiveAtGivenTime");  
363     try {  
364         if (hwc.isAndroid()) {  
365             iMilliseconds = date.getTime();  
366             result =  
367                 _HWC.isDstActiveAtGivenTime(iMilliseconds);  
368         }  
369         else if (hwc.isWindowsMobile())  
370         {  
371             // JavaScript's Date and WM's DateTime objects  
372             // differs in their base starting time  
373             // and definition. It was necessary to pass a  
374             // "time" to the OS - see below comment  
375             lMilliseconds = date.getTime();  
376             // Rather than pass a date string (which might be in  
377             // a different locale format)  
378             // the raw parameters of the particular "date" are  
379             // this also avoids a date string parse on the OS  
380             // side.

Develop Hybrid Apps Using Third-party Web Frameworks

SAP Mobile Platform
request = "indst=indst&";
response = undefined;
year = date.getFullYear();
month = date.getMonth() + 1;
day = date.getDate();
hour = date.getHours();
minute = date.getMinutes();
second = date.getSeconds();

request += "year=";
request += year.toString();
request += "&";
request += "month=";
request += month.toString();
request += "&";
request += "day=";
request += day.toString();
request += "&";
request += "hour=";
request += hour.toString();
request += "&";
request += "minute=";
request += minute.toString();
request += "&";
request += "second=";

response = hwc postDataToContainer("tz", request);
result = (response === 'true');
else if (hwc.isBlackBerry()){
    dMilliseconds = date.getTime();
    result = TimeZone.indst(dMilliseconds);
}
else if (hwc.isIOS()) {
    lMilliseconds = date.getTime();
    response = hwc.getDataFromContainer("tz", "&command=indst&time=" + lMilliseconds);
    result = (hwc.parseBoolean(response));
}
else {
    result = false;
}
return result;
}

/**
 * Returns the daylight savings offset in minutes for the current timezone at the given time.
 * Example: for Mountain Standard Time, at March 31st (currently is practicing DST), the returned offset is 60.
 * Example: for Mountain Standard Time, at November 31st (currently is not practicing DST), the returned offset is 0.
 * @desc Timezone
 * @memberOf hwc
 * @public
 * @param {Date} date Date at which to determine daylight savings offset.
 * @returns {int} Returns the number of minutes offset for daylight savings for the current
* timezone and at the given Date, or 0 if the current timezone doesn't practice daylight savings.

* @example

* var iDstOffsetAtTime = hwc.getDstOffsetAtGivenTimeInMinutes(date);

*/

hwc.getDstOffsetAtGivenTimeInMinutes = function (date) {

    var lMilliseconds, iMilliseconds, iMinutesOffset, response, dt,
        year, month, day, hour, minute, second, request, d,
        dMilliseconds, result;

    hwc.traceEnteringMethod("hwc.getDstOffsetAtGivenTimeInMinutes");
    try {
        if (hwc.isAndroid()) {
            iMilliseconds = date.getTime();
            iMinutesOffset = _HWC.getDstOffsetAtGivenTimeInMinutes(iMilliseconds);
            result = iMinutesOffset;
        }
        else if (hwc.isWindowsMobile())
            {
                // JavaScript's Date and WM's DateTime objects differs in their base starting time
                // and definition. It was necessary to pass a "time" to the OS - see below comment
                lMilliseconds = date.getTime();

                // Rather than pass a date string (which might be in a different locale format)
                // the raw parameters of the particular "date" are sent
                // this also avoids a date string parse on the OS side.

                // HWC.getDstOffsetAtGivenTimeInMinutes(lMilliseconds);
            }
    }
request = "dstoffset=dstoffset&";

year = date.getFullYear();
month = date.getMonth() + 1;
day = date.getDate();
hour = date.getHours();
minute = date.getMinutes();
second = date.getSeconds();

request += "year=";
request += year.toString();
request += ";
request += "month=";
request += month.toString();
request += ";
request += "day=";
request += day.toString();
request += ";
request += "hour=";
request += hour.toString();
request += ";
request += "minute=";
request += minute.toString();
request += ";
request += "second=";
request += second.toString();

response = hwc.postDataToContainer("tz", request);

d = response * 1;
iMinutesOffset = d;

result = iMinutesOffset;
else if (hwc.isBlackBerry()) {
    dMilliseconds = date.getTime();
    iMinutesOffset = TimeZone.dstoffset(dMilliseconds);
    result = iMinutesOffset;
}
else if (hwc.isIOS()) {
    lMilliseconds = date.getTime();
    response = hwc.getDataFromContainer("tz", "&command=dstoffset&time=" + lMilliseconds);
    result = parseInt(response, 10);
}
else {
    result = undefined;
}
return result;
};

/**
 * Returns a string containing the current Timezone's standard name. The name will not change based
 * on daylight savings periods. The native OS returns the string in the current locale where applicable.
 * Currently this string is derived from using available platform OS APIs. The values for the same
 * timezone will be different among platforms.
 * @desc Timezone
 * @memberOf hwc
 * @public
*/
518      * @returns {string} Returns a string containing the current
519      * Timezone's standard name.
520      * @example
521      * var sTzId = hwc.getTimezoneId();
522      *
523      */
524
525     hwc.getTimezoneId = function () {
526         var sTzId, request, response, result;
527
528         hwc.traceEnteringMethod("hwc.getTimezoneId");
529         try {
530             if (hwc.isAndroid()) {
531                 sTzId = _HWC.getTimezoneId() + ''; 
532                 result = sTzId;
533             }
534             else if (hwc.isWindowsMobile())
535             {
536                 request = "tzid=tzid";
537                 response = hwc.postDataToContainer("tz", request);
538                 result = response;
539             }
540             else if (hwc.isIOS()) {
541                 response = hwc.getDataFromContainer("tz","&command=tzid");
542                 result = (response);
543             } else if (hwc.isBlackBerry()){
544                 sTzId = TimeZone.tzid();
545                 result = sTzId;
546             }
547             else {
548                 result = undefined;
549             }
548             }
549         } finally {
550             hwc.traceLeavingMethod("hwc.getTimezoneId");
551         }
552     }
553 }
554
555     /**
556      * Returns whether the device's current timezone practices daylight savings. If a device's current timezone never practices daylight savings, this function returns "false". If a device's current timezone practices DST, but DST rules are not currently in effect, function returns "true".
557      * @desc Timezone
558      * @memberOf hwc
559      * @public
560      * @returns {boolean} Returns true iff the device's current timezone practices daylight savings, irrespective of whether daylight savings is currently in effect.
561      * @example
562      * var isDstAware = hwc.getUsesDST();
563      *
564      */
565      hwc.getUsesDST = function () {
566          var date, lMilliseconds, request, response, result;
567          hwc.traceEnteringMethod("hwc.getUsesDST");
568          try {
569              if (hwc.isAndroid()) {
570                  result = _HWC.useDaylightTimeCurrently();
571              }
572          } finally {
573          if (hwc.isWindowsMobile())
574              result = _HWC.useDaylightTimeCurrently();
575          }
576          return result;
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.
2. Set the values for all the fields. The fields names map to the Department MBO create operation’s parameter name.

```javascript
var dep1 = new Department();
dep1.dept_id = "800";
dep1.dept_name = "Dept";
dep1.dept_head_id = "888";
```

3. Call the create online request function.

```javascript
department_create_onlineRequest(dep1,
    "supusername=supAdmin&suppassword=s3pAdmin",
    function() { alert("error occurred")});
```

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

```javascript
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_paramKey').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:

```javascript
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 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() == 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";
2. Create a new employee:

```javascript
var emp1 = new Employee();
emp1.emp_id = "1";
emp1.manager_id = "2";
emp1.emp_fname = "Yan";
emp1.emp_lname = "Gong";
emp1.street = "King Street";
emp1.city = "Waterloo";
emp1.state = "ON";
emp1.zip_code = "n2v3l4";
emp1.phone = "518-8836863";
emp1.status = "A";
emp1.ss_number = "024601768";
emp1.salary = "324234";
emp1.start_date = "1996-12-30";
emp1.termination_date = "1999-12-20";
emp1.birth_date = "1956-12-20";
emp1.bene_health_ins = "Y";
emp1.bene_life_ins = "Y";
emp1.bene_day_care = "Y";
emp1.sex = "F";
```

3. Create a second employee:

```javascript
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:

```javascript
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:

```javascript
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.

If the delete operation has old value arguments, you also need to set old values for parent and child MBOs. This example assumes the delete operation has old value arguments, and the data (1 department and 2 employee) has been inserted into back end:

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

```javascript
function deleteDepartment() {
  var dep = new Department();
  dep.dept_id="2";

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

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

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

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

  if ( workflowMessage.getValues().getData('Department_dept_id_attribKey').getValue()== '2'){
    var dep = new Department();
    dep.dept_id=workflowMessage.getValues().getData('Department_dept_id_attribKey').getValue();
    dep.dept_head_id=workflowMessage.getValues().getData('Department_dept_head_id_attribKey').getValue();
    dep.dept_name=workflowMessage.getValues().getData('Department_dept_name_attribKey').getValue();

    var oldDep = new Department();
    oldDep.dept_id=workflowMessage.getValues().getData('Department_dept_id_attribKey').getValue();
    oldDep.dept_name=workflowMessage.getValues().getData('Department_dept_name_attribKey').getValue();
    oldDep.dept_head_id=workflowMessage.getValues().getData('Department_dept_head_id_attribKey').getValue();
```
for( var i = 0; i < employees.length ; i++ ) {
    var emp = new Employee();
    emp.emp_id = employees[i].getData('Employee_emp_id_attribKey').getValue();
    emp.manager_id = employees[i].getData('Employee_manager_id_attribKey').getValue();
    emp.emp_fname = employees[i].getData('Employee_emp_fname_attribKey').getValue();
    emp.emp_lname = employees[i].getData('Employee_emp_lname_attribKey').getValue();
    emp.dept_id = employees[i].getData('Employee_dept_id_attribKey').getValue();
    emp.street = employees[i].getData('Employee_street_attribKey').getValue();
    emp.city = employees[i].getData('Employee_city_attribKey').getValue();
    emp.state = employees[i].getData('Employee_state_attribKey').getValue();
    emp.zip_code = employees[i].getData('Employee_zip_code_attribKey').getValue();
    emp.phone = employees[i].getData('Employee_phone_attribKey').getValue();
    emp.status = employees[i].getData('Employee_status_attribKey').getValue();
    emp.ss_number = employees[i].getData('Employee_ss_number_attribKey').getValue();
    emp.salary = employees[i].getData('Employee_salary_attribKey').getValue();
    emp.start_date = employees[i].getData('Employee_start_date_attribKey').getValue().substr(0, 10);
    emp.termination_date = employees[i].getData('Employee_termination_date_attribKey').getValue().substr(0, 10);
    emp.birth_date = employees[i].getData('Employee_birth_date_attribKey').getValue().substr(0, 10);
    emp.bene_health_ins = employees[i].getData('Employee_bene_health_ins_attribKey').getValue();
    emp.bene_life_ins = employees[i].getData('Employee_bene_life_ins_attribKey').getValue();
    emp.bene_day_care = employees[i].getData('Employee_bene_day_care_attribKey').getValue();
    emp.sex = employees[i].getData('Employee_sex_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:

```javascript
department_delete_onlineRequest( dep, oldDep, function( error)
{ alert(error));
}
........
}
```

### MediaCache Examples

```javascript
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 `MessageValueCollection` 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:

```html
<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:

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

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

```javascript
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:

  ```html
  <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.

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

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

  ```html
  <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:

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

**set screentitle**

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.

**remove all menu items**

Removes all the menu items from the Hybrid Web Container.
clearrequestcache
Clears the entire Online Request cache for the current Hybrid App.

clearrequestcachecacheitem
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.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Size Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>Large attachments can produce longer processing times.</td>
</tr>
<tr>
<td>Android</td>
<td>Large attachments can produce longer processing times. There is a 1MB limit for attachments on Android devices.</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>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.</td>
</tr>
<tr>
<td>BlackBerry 5.0 and BlackBerry 6.0</td>
<td>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.</td>
</tr>
</tbody>
</table>
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>/MobileSDK23/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 App Package 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>\MobileSDK23\HybridApp\API\Container` and `<SMP_HOME>\MobileSDK23\HybridApp\API\AppFramework` directories.
- Use `WorkflowMessage.js` to view and manipulate the Hybrid App messages. This file is located in `<SMP_HOME>\MobileSDK23\HybridApp\API\AppFramework`
**HTML Format**

This is a commonly used HTML format.

```html
<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>
</head>
<body onload="onHybridAppLoad()">
<div id="Screen1KeyScreenDiv" smp_screen_title="Screen1Title" smp_menuitems="NativeMenu1Key,NativeMenu1DisplayName,NativeMenu2Key,NativeMenu2DisplayName" smp_okaction="myOKAction()">
    [...]
</div>
</body>
<script>
$(document).ready( function(){
    [...]
});
</script>
</html>
```

**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.

```xml
<manifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="">
</manifest>
```
Develop Hybrid Apps Using Third-party Web Frameworks

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>
      </HTMLWorkflow>
    </Android>
  </ClientWorkflows>
</Module>
<Module>…</Module>
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:

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

**ProcessUpdates**

```xml
<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**

```xml
<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**

```xml
<RequiresActivation key="shared_activation_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**

```xml
<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**

```xml
<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.

```xml
<Class>Sybase.UnwiredPlatform.WorkflowClient.Transformer</Class>
```

The .NET Type in the assembly that implements the IMailProcessor interface.

**ResponsePlugin**

```xml
<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**

```xml
<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.

```xml
<Class>Sybase.UnwiredPlatform.WorkflowClient.Responder</Class>
```

The .NET Type in the assembly that implements the IResponseProcessor interface.

**ClientWorkflows**

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

Develop Hybrid Apps Using Third-party Web Frameworks

Developer Guide: Hybrid Apps 613
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
<?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.

```xml
<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**

```xml
<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.

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

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**

```xml
<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.
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.

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.

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 client variables 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.

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.

```
<?xml version="1.0" encoding="utf-8"?>
  <Globals>
    <DefaultScreens activation="..." credentials="..."/>
  </Globals>
  <Triggers>
    <Actions>
      <Action name="..." sourcescreen="..." targetscreen="..." errorscreen="..."/>
      <Method type="replay" mbo="..." package="..."/>
    </Actions>
  </Triggers>
</Workflow>
```
Develop Hybrid Apps Using Third-party Web Frameworks
<Methods>
</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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the action, which typically corresponds to the key of the menuitem that invoked the action.</td>
</tr>
</tbody>
</table>
### Table 2. Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>The type of method to invoke. For object queries, this must be <code>search</code>. For operations, it must be <code>replay</code>.</td>
</tr>
<tr>
<td><strong>mbo</strong></td>
<td>The name of the mobile business object (MBO).</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>optype</td>
<td>The type of MBO operation to invoke. Must be one of these types:</td>
</tr>
<tr>
<td></td>
<td>• none</td>
</tr>
<tr>
<td></td>
<td>• create</td>
</tr>
<tr>
<td></td>
<td>• update</td>
</tr>
<tr>
<td></td>
<td>• delete</td>
</tr>
<tr>
<td></td>
<td>• other</td>
</tr>
<tr>
<td>opname</td>
<td>The name of the MBO operation to invoke.</td>
</tr>
<tr>
<td>generatedOld</td>
<td>A boolean that indicates whether or not to generate old value keys.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceType</td>
<td>The source of the data. Must be one of these types:</td>
</tr>
<tr>
<td></td>
<td>• Key</td>
</tr>
<tr>
<td></td>
<td>• BackEndPassword</td>
</tr>
<tr>
<td></td>
<td>• BackEndUser</td>
</tr>
<tr>
<td></td>
<td>• DeviceId</td>
</tr>
<tr>
<td></td>
<td>• DeviceName</td>
</tr>
<tr>
<td></td>
<td>• DeviceType</td>
</tr>
<tr>
<td></td>
<td>• UserName</td>
</tr>
<tr>
<td></td>
<td>• MessageId</td>
</tr>
<tr>
<td></td>
<td>• ModuleName</td>
</tr>
<tr>
<td></td>
<td>• ModuleVersion</td>
</tr>
<tr>
<td></td>
<td>• QueueId</td>
</tr>
<tr>
<td></td>
<td>• ContextVariable</td>
</tr>
<tr>
<td>workflowKey</td>
<td>If the sourceType is <strong>Key</strong>, the name of the key in the Hybrid App message from which to obtain the value.</td>
</tr>
<tr>
<td>contextVariable</td>
<td>If the sourceType is <strong>ContextVariable</strong>, the name of the context variable from which to obtain the value.</td>
</tr>
<tr>
<td>paramName</td>
<td>If present, the name of the parameter the value is supplying.</td>
</tr>
<tr>
<td>pkName</td>
<td>If present, the name of the personalization key the value is supplying.</td>
</tr>
<tr>
<td>attribName</td>
<td>If present, the name of the attribute name the value is supplying.</td>
</tr>
<tr>
<td>parentMBO</td>
<td>The name of the parent MBO, if any.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>relationShipName</td>
<td>The name of the relationship, if any.</td>
</tr>
<tr>
<td>mboType</td>
<td>The type of the value in MBO terms. Must be one of these types:</td>
</tr>
<tr>
<td></td>
<td>• string</td>
</tr>
<tr>
<td></td>
<td>• char</td>
</tr>
<tr>
<td></td>
<td>• date</td>
</tr>
<tr>
<td></td>
<td>• datetime</td>
</tr>
<tr>
<td></td>
<td>• time</td>
</tr>
<tr>
<td></td>
<td>• int</td>
</tr>
<tr>
<td></td>
<td>• byte</td>
</tr>
<tr>
<td></td>
<td>• short</td>
</tr>
<tr>
<td></td>
<td>• long</td>
</tr>
<tr>
<td></td>
<td>• decimal</td>
</tr>
<tr>
<td></td>
<td>• boolean</td>
</tr>
<tr>
<td></td>
<td>• binary</td>
</tr>
<tr>
<td></td>
<td>• float</td>
</tr>
<tr>
<td></td>
<td>• double</td>
</tr>
<tr>
<td></td>
<td>• list</td>
</tr>
<tr>
<td></td>
<td>• integer</td>
</tr>
<tr>
<td></td>
<td>• structure</td>
</tr>
<tr>
<td>array</td>
<td>A boolean that indicates whether or not the value is an array. The default is false.</td>
</tr>
<tr>
<td>length</td>
<td>The length of the parameter/attribute/personalization key.</td>
</tr>
<tr>
<td>precision</td>
<td>The precision of the parameter/attribute/personalization key.</td>
</tr>
<tr>
<td>scale</td>
<td>The scale of the parameter/attribute/personalization key.</td>
</tr>
<tr>
<td>convertToLocalTime</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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

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

Table 5. Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generatedOld</td>
<td>A boolean that indicates whether or not to generate old value keys.</td>
</tr>
</tbody>
</table>

Inner tags

Contains the description of how to map the results of the object query to a key in the Hybrid App message.

Mapping

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

Table 6. Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflowKey</td>
<td>The name of the key in the Hybrid App message to fill with the results of the object query.</td>
</tr>
<tr>
<td>workflowType</td>
<td>The type of the data in the Hybrid App message. Must be one of these types:</td>
</tr>
<tr>
<td></td>
<td>• text</td>
</tr>
<tr>
<td></td>
<td>• number</td>
</tr>
<tr>
<td></td>
<td>• boolean</td>
</tr>
<tr>
<td></td>
<td>• datetime</td>
</tr>
<tr>
<td></td>
<td>• date</td>
</tr>
<tr>
<td></td>
<td>• time</td>
</tr>
<tr>
<td></td>
<td>• list</td>
</tr>
<tr>
<td>attribName</td>
<td>If present, the name of the attribute name to which the key is mapped.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hardCodedValue</td>
<td>If the workflowType is not choice, and attributeName is not present, the hard-coded value to which the key is mapped.</td>
</tr>
<tr>
<td>keyWorkflowKey</td>
<td>If the workflowType is choice, the key to which to map the dynamic display names of the choice.</td>
</tr>
<tr>
<td>valueWorkflowKey</td>
<td>If the workflowType is choice, the key to which to map the dynamic values of the choice.</td>
</tr>
<tr>
<td>assumeLocalTime</td>
<td>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.</td>
</tr>
<tr>
<td>array</td>
<td>A boolean that indicates whether or not the value is an array. The default is false.</td>
</tr>
<tr>
<td>mboType</td>
<td>The type of the value in MBO terms. Must be one of these types:</td>
</tr>
<tr>
<td></td>
<td>• string</td>
</tr>
<tr>
<td></td>
<td>• char</td>
</tr>
<tr>
<td></td>
<td>• date</td>
</tr>
<tr>
<td></td>
<td>• datetime</td>
</tr>
<tr>
<td></td>
<td>• time</td>
</tr>
<tr>
<td></td>
<td>• int</td>
</tr>
<tr>
<td></td>
<td>• byte</td>
</tr>
<tr>
<td></td>
<td>• short</td>
</tr>
<tr>
<td></td>
<td>• long</td>
</tr>
<tr>
<td></td>
<td>• decimal</td>
</tr>
<tr>
<td></td>
<td>• boolean</td>
</tr>
<tr>
<td></td>
<td>• binary</td>
</tr>
<tr>
<td></td>
<td>• float</td>
</tr>
<tr>
<td></td>
<td>• double</td>
</tr>
<tr>
<td></td>
<td>• list</td>
</tr>
<tr>
<td></td>
<td>• integer</td>
</tr>
<tr>
<td></td>
<td>• structure</td>
</tr>
<tr>
<td>relationShipName</td>
<td>The name of the relationship, if any.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The type of the notification. Must be onEmail-Triggered.</td>
</tr>
<tr>
<td>targetscreen</td>
<td>The screen to which the client will be opened if the object query succeeds.</td>
</tr>
<tr>
<td>errorscreen</td>
<td>The screen to which the client will be opened, by default, if the object query fails.</td>
</tr>
<tr>
<td>• errorlogskey</td>
<td></td>
</tr>
<tr>
<td>• errorlogmessagekey</td>
<td></td>
</tr>
<tr>
<td>• errorlogmessageaslistkey</td>
<td></td>
</tr>
</tbody>
</table>

Inner tags

<Transform> ... </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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The type of the rule. Must be regex-extract.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| source        | The source of the data to be extracted. Must be one of these sources:  
• 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  | The type of the data in the Hybrid App message. Must be one of these data types:  
• 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
<?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.

For iOS Hybrid Apps that do not fully display buttons on the bottom of the screen, use the `<AdjustWebviewFrame>`1</AdjustWebviewFrame> element after the `</screens>` entry. For example:

```xml
<?xml version="1.0" encoding="utf-8"?>
<widget>
    <screens src="html/NonDefaultMetadataTest.html" default ="" >
        <screen key="html/NonDefaultMetadataTest.html">
        
    </screens>
    </widget>
    <AdjustWebviewFrame>1</AdjustWebviewFrame>
    <zoom>0</zoom>
</widget>
```

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:

```xml
<BlackBerry>
    <HTMLWorkflow>
        <File>TokenSI_CustomLookAndFeel.xml</File>
    </HTMLFiles>
    <HTMLFiles>
        <HtmlFile>html/css/bb/some-3rd-part.css</HtmlFile>
        <HtmlFile>html/css/bb/checkbox.css</HtmlFile>
    </HTMLFiles>
</BlackBerry>
```
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.
Develop Hybrid Apps Using Third-party Web Frameworks

- Uninstalls the original package.
- Installs the new package with the same name and version.
- 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.

You cannot run Hybrid Apps created in a newer version of the Hybrid App Designer on an older version of the Hybrid Web Container. The newly generated code could call native functionality that previous Hybrid Web Containers are unaware of.

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:

   ```sql
   SELECT emp_id, emp_fname, emp_lname, dept_id
   FROM sampledb.dba.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.
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 sampledb database that has at least one load argument. For example, you could define a Sales_order MBO as:

   ```sql
   SELECT id,
          cust_id,
          order_date,
          fin_code_id,
          region FROM sampledb.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.
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:

   ```sql
   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:

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

```javascript
/**
 * Returns relative URL to the html directory
 */
```
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.

```html
<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>
```

- The menu has one anchor (a) element for each menu item:
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.

```html
<div data-role="content" class="wrapper">
  <div data-role="scroller">
    <form name="Department_createForm" id="Department_createForm">
      <div class="customTopOfFormStyle" id="Department_createForm_help" class="help">
      </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"/>
      </div>
    </form>
  </div>
</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:

```javascript
/*
#include "Custom.js"

var screenKey = getCurrentScreen();
var form = document.forms[screenKey + "Form"];  
if (form) {
  var topOfFormElem = document.getElementById("topOfFormKey" + screenKey + "Form");
  topOfFormElem.innerHTML = "Use this screen to ...";
  var bottomOfFormElem = document.getElementById("bottomOfFormKey" + 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
<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>
    </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
<html>
  <body onload="hwc.onHybridAppLoad();">
    <div id="Department_createScreenDiv">
      <form name="Department_createForm" id="Department_createForm">
        <div class="customTopOfFormStyle" id="Department_createForm_help" class="help"></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"/>
          <div class="Department_create_depart_id_paramKey_help"></div>
        </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, `<ul>`, a child element for the menu. The menu has one list item, `<li>`, for each menu item.
• 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 <span> 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
<html>
  <body onload="onHybridAppLoad();">
    <div id="Department_createScreenDiv">
      <ul id="Department_createScreenDivMenu" class="menu">
        <li><a class="nav" name="Create" onclick="menuItemCallbackDepartment_createSubmit_Workflow();">Create</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&amp;c=firsttab&amp;a=0&amp;p=categories, then navigate to the current version of this topic.

These JavaScript files are also included:

* Util.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
AndUpdateHandler object for that screen. That UpdateHandler 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:

```javascript
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;
}
}

try {
    customAfterWorkflowLoad();
    //Setup UIHandler to draw our Listview Screen
    UIUpdateHandlers[0] = new MyListViewUpdateHandler();
}

} // function callback

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>
Your activation was successful, the newly created Hybrid App requests will automatically be pushed to you.<br>
For more information contact your administrator or visit us
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:
   ```javascript
   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 `getValues()` or `getValues()` if you do not need to modify the data structure.

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_para
```
handling individual items
Develop a Hybrid App Using the Hybrid App Designer

```javascript
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:

```javascript
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, `<img src="URI from getPicture" width=50 height=50 />`. 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

```javascript
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 the following example. Note that this example is specific to top level screens. See Example 3 for a more general code example:

```javascript
var options = { destinationType: PictureOption.DestinationType.IMAGE_URI, sourceType: PictureOption.SourceType.PHOTOLIBRARY };
getPicture(onGetPictureError, onGetPictureSuccess, options);
```
function onGetPictureSuccess(fileName, imageUri) {
// Set file for upload
var fileDataKey = "Picture_create_fileData_paramKey";

// Code for calling from top level screen
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);
    // Code for calling from top level screen
    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);

When uploading the image to the server for a URI, the JavaScript looks similar to the following example. This example is more general compared to Example 1, since it is invoked from a lower level screen:

// invoke from a lower level screen
var messageValue =
getCurrentMessageValueCollection().getData(contentDataKey);

if (messageValue) {
  // Update file for upload
  messageValue.setValue(base64String);
} else {
  // Add file for upload
  messageValue = new MessageValue();
  messageValue.setKey(contentDataKey);
  messageValue.setValue(base64String);
  messageValue.setType(MessageValueType.TEXT);
  // invoke from a lower level screen
  getCurrentMessageValueCollection().add(contentDataKey,
  messageValue);
}

**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 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/>";
        }
    }
    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:

```javascript
//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").getValue();
              var orderId = currItem.getData("PersonalWorkQueue_orderId_attribKey").getValue();
              var operationNumber = currItem.getData("PersonalWorkQueue_operationNumber_attribKey").getValue();
              var description = currItem.getData("PersonalWorkQueue_description_attribKey").getValue();

              try {
                  var promDate = currItem.getData("PersonalWorkQueue_datePromised_attribKey").getValue();
              } catch (err) {
                  var promDate = "";
              }

              try {
                  var planDate = currItem.getData("PersonalWorkQueue_dateStartPlan_attribKey").getValue();
              } catch (err) {
                  var planDate = "";
              }

```

Develop a Hybrid App Using the Hybrid App Designer
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>'
i++;
}

htmlOutput += '</li></ul>'

//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.

```javascript
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
Develop a Hybrid App Using the Hybrid App Designer

- 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:

```javascript
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:

```javascript
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
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
```javascript
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:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Encryption policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackBerry</td>
<td><code>PersistentStore</code>, which adheres to the Content Protection BES IT policy</td>
</tr>
<tr>
<td>Android</td>
<td>Encrypted before storing into the SQLite database</td>
</tr>
<tr>
<td>iOS</td>
<td>Stored in SQLite Encryption Extensions database</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>Unencrypted SQLite—security is deferred to Afaria Security Manager</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Platform</th>
<th>Data storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackBerry</td>
<td>Amount of free PersistentStore.</td>
</tr>
<tr>
<td>iOS and Android</td>
<td>Amount of free file system for the SQLite database, and/or the SQLite database size limit</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>Amount of free file system, and the SQLite database size limit.</td>
</tr>
</tbody>
</table>

## 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 Attachment Viewer 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.

- 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.

![Conditional Navigation Example](image)

2. Go to the Flow Design page to see the conditional navigation paths in the flow.
3. In the `Custom.js` file, add the custom code for conditional navigation.

```javascript
//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.
5. Add you custom code to the Custom.js file. For example:

```javascript
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:

```javascript
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_ss1', 'Screen_Start_One' );
    customNavigations[1] = new customNavigationEntry( 'Fred_second_screen', 'Screen_Start_Two' );
    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:
   ```javascript
   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.
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.)

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.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Key Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cc_username</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>cc_password</td>
<td>string</td>
</tr>
</tbody>
</table>
Develop a Hybrid App Using the Hybrid App Designer

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.

   ![Diagram](image)

   **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.

```java
InputStream is = getClass().getResourceAsStream("sybase101.p12");
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:

<table>
<thead>
<tr>
<th>Content protection strength setting</th>
<th>ECC encryption key length (in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>160</td>
</tr>
<tr>
<td>Stronger</td>
<td>283</td>
</tr>
<tr>
<td>Strongest</td>
<td>571</td>
</tr>
</tbody>
</table>

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 if 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.
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.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Select the language.</td>
</tr>
<tr>
<td>Country</td>
<td>Select the country.</td>
</tr>
<tr>
<td>Variant</td>
<td>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.</td>
</tr>
<tr>
<td>Overwrite existing file</td>
<td>Overwrite an existing localization file.</td>
</tr>
<tr>
<td>Automatically create default locale</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**.
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.
Develop a Hybrid App Using the Hybrid App Designer

Specify the name of the key and, optionally, the input data binding for the key as well.

Name: AprobacionDeViaje
Type: string

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

OK Cancel
• 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:
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.

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:
1. In the Programs screen, open the **Hybrid Web Container**.

**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.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Represents the default configuration. The default values are used if optional fields are left blank in a module trace configuration. Required.</td>
</tr>
<tr>
<td>Device Management</td>
<td>Miscellaneous functions related to device registration, event notification, and device administration. Enable tracing for problems in these areas.</td>
</tr>
<tr>
<td>JMSBridge</td>
<td>This module handles communications from the SAP Mobile Server to the messaging server. Enable tracing to view the detailed messaging exchange.</td>
</tr>
<tr>
<td>MO</td>
<td>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.</td>
</tr>
<tr>
<td>SUPBridge</td>
<td>This module handles communications from the messaging server to the SAP Mobile Server. Enable tracing to view the detailed messaging exchange.</td>
</tr>
<tr>
<td>Module</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TM</td>
<td>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.</td>
</tr>
<tr>
<td>WorkflowClient</td>
<td>The WorkflowClient module.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>Display only. Default, module name, or list of module names selected.</td>
</tr>
<tr>
<td>Description</td>
<td>(Optional) Custom description of the server module.</td>
</tr>
<tr>
<td>Level</td>
<td>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.</td>
</tr>
<tr>
<td>Max trace file size</td>
<td>(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.</td>
</tr>
<tr>
<td>User name</td>
<td>(Optional) Only data for the specified user name is traced.</td>
</tr>
<tr>
<td>Application Connection ID</td>
<td>(Optional) Only data for the specified Application ID is traced.</td>
</tr>
</tbody>
</table>

b) Click **OK**.

Log files for each module are stored in folders of the same name located in: `SMP_HOME\Servers\UnwiredServer\logs`. 

---

**Developer Guide: Hybrid Apps** 715
Develop a Hybrid App Using the Hybrid App Designer
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.
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.
7. 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.

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Guidelines</th>
</tr>
</thead>
</table>
| Hybrid App                  | • 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      | • 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.

<table>
<thead>
<tr>
<th>Context Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SupUser</td>
<td>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.</td>
</tr>
<tr>
<td>SupDomain</td>
<td>The name of the domain that the Hybrid App package is deployed to.</td>
</tr>
<tr>
<td>SupUnrecoverableErrorRetryTimeout</td>
<td>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.</td>
</tr>
<tr>
<td>SupThrowCredentialRequestOn401Error</td>
<td>The default is true, which means that an error code 401 throws a CredentialRequestException, 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.</td>
</tr>
<tr>
<td>Context Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SupThrowBadHttpHeadersOn412Error</td>
<td>The default is <strong>true</strong>, which means that an error code 412 throws a <code>BadHttpHeadersException</code>. If this property is set to <strong>false</strong>, error code 412 is treated as a normal recoverable exception.</td>
</tr>
<tr>
<td>SupRecoverableErrorRetryTimeout</td>
<td>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.</td>
</tr>
<tr>
<td>SupPassword</td>
<td>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.</td>
</tr>
<tr>
<td>SupPackages</td>
<td>The name and version of the MBO packages that are used in the Hybrid App. This cannot be changed.</td>
</tr>
<tr>
<td>SupMaximumMessageLength</td>
<td>Use this property to increase the allowed maximum Hybrid App message size. Limitations vary depending on device platform:</td>
</tr>
<tr>
<td></td>
<td>• For BlackBerry 5, the limit is 512 bytes.</td>
</tr>
<tr>
<td></td>
<td>• For Windows Mobile the limit is 500 bytes.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>SupWorkflowVersion</td>
<td>The version number of the Hybrid App package.</td>
</tr>
</tbody>
</table>

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 `hwce2eTrace` 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]
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 the device 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.
Manage a Hybrid App Package
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.

You must first Build a Customized Hybrid Web Container Using the Provided Source Code before creating a prepackaged Hybrid App.

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

- 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.

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](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.
You can use the provided BlackBerry Hybrid Web Container template to build a custom user interface and configure other resources.

**Prerequisites**
- 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](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.
   If there is no key file and believe it is not needed, return NULL from `getCodeSignerId()`. The key file is used to protect data in the persistent store. If there is no key file and you want to create one, install and use `BlackBerry Signing Authority Tool`.
   Once you have created a key file, add it to your project, so it is included in the `.cod` file. `getCodeSignerId()` then needs to return the name of the key file without an extension.
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 \MobileSDK\version\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_2.3.2.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.

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.
   b) Download the Android SDK (for example, `installer_r21-windows.exe`).
Install and Configure the Hybrid Web Container On the Device

**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.

   ![Android SDK Manager](image)

   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`. 

---

736 SAP Mobile Platform
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) window](image)

      c) Click **OK** to add the instance to the AVD Manager.

      ![Create new Android Virtual Device (AVD) window](image)
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*. 
Install and Configure the Hybrid Web Container On the Device

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.
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**.
Install and Configure the Hybrid Web Container On the Device

- 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.
Installing Microsoft Synchronization Software

Install and configure Microsoft synchronization software so you can deploy and run an application on a Windows Mobile emulator.


1. Download Microsoft ActiveSync:
   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".
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:
   - Automatic (Password) – enter the password for automatic registration.
   - 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.
• 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.
• 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
Use Afaria to get an X.509 certificate on to an iOS device.
See the topic Provisioning with Afaria in Mobile Application Life Cycle for details.

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 *Apple Local and Push Notifications in Depth* for details.


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.
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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>The push notification server.</td>
</tr>
<tr>
<td>Port</td>
<td>Push notification server port.</td>
</tr>
<tr>
<td>Feedback server</td>
<td>If a feedback service is enabled, the server to which APNS routes feedback information.</td>
</tr>
<tr>
<td>Feedback port</td>
<td>The feedback service port.</td>
</tr>
</tbody>
</table>
Install and Configure the Hybrid Web Container On the Device

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate (encoded)</td>
<td>The security certificate used for authentication.</td>
</tr>
<tr>
<td>Certificate password</td>
<td>The security certificate password.</td>
</tr>
</tbody>
</table>

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.
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.
Install and Configure the Hybrid Web Container On the Device
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\MobileSDK23\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.

   ```xml
   <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\car.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 the 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:
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:


  **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.

<table>
<thead>
<tr>
<th>Touch Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ANDROID_CUSTOMIZATION_POINT_COLORS</code></td>
<td>Use custom colors for the Hybrid Web Container.</td>
</tr>
<tr>
<td><code>ANDROID_CUSTOMIZATION_POINT_FONTS</code></td>
<td>Use custom fonts in the Hybrid Web Container.</td>
</tr>
<tr>
<td><code>ANDROID_CUSTOMIZATION_POINT_BRAND</code></td>
<td>Change application name, copyright, and developer information</td>
</tr>
<tr>
<td><code>ANDROID_CUSTOMIZATION_POINT_SPLASHSCREEN</code></td>
<td>Add a splash screen to the Hybrid Web Container.</td>
</tr>
<tr>
<td>Touch Point</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_DEFAULTSETTINGS</td>
<td>Set the defaults for the Settings screen.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_PRESETSETTINGS</td>
<td>Hard code settings for the Settings screen so they do not show up on the device. This prevents the user from changing the settings.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_PREPACKAGED_APP</td>
<td>Run the Hybrid Web Container as a single Hybrid App.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_PIN</td>
<td>Use for PIN screen customizations, or to remove the PIN screen.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_SORTING</td>
<td>Sort Hybrid App messages based on different criteria.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_FILTERING</td>
<td>Filter the list of Hybrid App messages so only messages meeting certain criteria are shown.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSORT</td>
<td>Customize the criteria for how the Hybrid App list is sorted.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH</td>
<td>Make the list of Hybrid App packages searchable.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST</td>
<td>Customize the Hybrid App package list appearance.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS</td>
<td>Create categorized views of the Hybrid App packages.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_HTTPHEADERS</td>
<td>Set HTTP headers for the Android Hybrid Web Container to include authentication tokens.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_PUSH_NOTIFICATION</td>
<td>Customize how the Hybrid Web Container handles the push notification.</td>
</tr>
<tr>
<td>ANDROID_CUSTOMIZATION_POINT_ANONYMOUS_USER</td>
<td>Returns whether or not anonymous user support is being used. Change to YES to allow clients to register anonymously.</td>
</tr>
</tbody>
</table>

**Note:** For this to work, the HWC application connection template must be configured to use the anonymous security configuration. See Application Connection Templates in SAP Control Center for SAP Mobile Platform.
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:
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:
```java
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.setTextColor.”

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
<?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:

```java
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;
    }

    public void setText( String text )
    {
        super.setText( text );
    }

    public void setCustomFont( Context oContext, AttributeSet oAttrs, 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!" );
        }
    }

    public void setCustomFont( Context oContext, AttributeSet oAttrs, int iFontId, int iStyle )
    {
        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!" );
        }
    }

    public void setCustomFont( Context oContext, AttributeSet oAttrs, int iFontId, int iStyle, int iWeight )
    {
        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!" );
        }
    }

    public void setCustomFont( Context oContext, AttributeSet oAttrs, int iFontId, int iStyle, int iWeight, int iStyleSet )
    {
        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!" );
        }
    }

    public void setCustomFont( Context oContext, AttributeSet oAttrs, int iFontId, int iStyle, int iWeight, int iStyleSet, int iStyleInSet )
    {
        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!" );
        }
    }
}

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.

Open the `workflowmessages.xml` file for editing and add this attribute to the `RelativeLayout` tag:

```xml
```

Find the `TextView` tag with the "ID msg_from" and change the tag from a `TextView` tag to a "com.sybase.hwc.CustomFontTextView" tag.

Add this attribute to the `com.sybase.hwc.CustomFontTextView` tag:

```xml
custom:customFont="<NAME_OF_YOUR_FONT_FILE.TTF>"
```

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.

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`:

```java
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:

   ```java
   // Display high priority messages only.
   ArrayList<Message> filteredMessages = MessageDb.getMessages( 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
<?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>
```
   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:

```java
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
```
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());
    startActivityForResult(oIntentHybridAppContainer, Consts.INTENT_ID_HYBRIDAPP_CONTAINER);
}

@Override
public void onActivityResult(int iRequestCode, int resultCode, Intent relaunchData)
{
    super.onActivityResult(iRequestCode, resultCode, relaunchData);
    if (iRequestCode == Consts.INTENT_ID_HYBRIDAPP_CONTAINER && resultCode == 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_PROGRESS_TEXT));
        startActivityForResult(oIntentHybridAppContainer, Consts.INTENT_ID_HYBRIDAPP_CONTAINER);
    }
}

public class ImageAdapter extends BaseAdapter
{
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 HybridApps
    mHybridApps = new Vector<HybridApp>(Arrays.asList(HybridAppDb.getInvocableHybridApps(false)));
    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.
      (Android only) That class must extend Activity.
7. (Android only) 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
<?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="6dp">
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="fill_parent">
        <TextView
            android:id="@+id/category"
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:singleLine="true"
            android:ellipsize="marquee"
            android:gravity="center_vertical"/>
    </LinearLayout>
</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:

```java
public static final String[] m_asHybridAppCategories = {
    "Financial", "Utilities", "Miscellaneous"};
```

4. Replace the HybridAppAdapter class with:

```java
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( oHybridApp.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 ) {
            HybridWeb Container Customization
SAP Mobile Platform
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;
                    }
                }
                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) {
                    HybridWebContainerCustomization.
                }
            }
        }
    });
}
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(false) ) ), m_asHybridAppCategories );

6. Modify the onListItemClick method as shown in the example code (changes are shown in bold):
   public void onListItemClick(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, ((HybridAppState) 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**):

```java
public void onCreateContextMenu( ContextMenu oMenu, View v, ContextMenu.ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu( oMenu, v, menuInfo );

    AdapterContextMenuInfo oInfo = (AdapterContextMenuInfo) 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**):

```java
public boolean onContextItemSelected( MenuItem oItem )
{
    if ( oItem.getItemId() == CONTEXT_MENU_DELETE )
    {
        AdapterContextMenuInfo oInfo = (AdapterContextMenuInfo) oItem.getMenuInfo();

        // 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:

```java
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 ) {
                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
    ImageView ic = (ImageView)
    v.findViewById( R.id.msg_icon );
    if ( oMsg.isMsgProcessed() )
        ic.setImageResource( UiIconIndexLookup.getProcessedIconIdForIndex( oMsg.getIconIndex()));
    else
        ic.setImageResource( UiIconIndexLookup.getNormalIconIdForIndex( oMsg.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 );
     }
}
}

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 iMessageCategoryIndex = getCategoryIndex( oMessage );
int iCategoryIndex = getCategoryIndex( sString );
if( iCategoryIndex <= iMessageCategoryIndex )
{
    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].toLowerCase() ) >= 0 )
                {
                    return index;
                }
            }
        }
    }
}
11. In the **onResume** method, make these changes (changes are shown in **bold**):

```java
try {
    // ANDROID_CUSTOMIZATION_POINT_FILTERING
    ArrayList<Message> alMessages = MessageDb.getMessages();
    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**):

```java
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 sqlite database
            try {
                oMsg.getTransformData();
            } catch ( Exception ex ) {
                MocaLog.getAmpHostLog().logMessage( "Failed to read transform data", MocaLog.eMocaLogLevel.Normal );
            }
        }
    }
```
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);
}

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 \textbf{Activity}.

\textbf{14. (Android only) Update the manifest.xml file to include the new activity you create.}

\textit{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 \texttt{ANDROID\_CUSTOMIZATION\_POINT\_HYBRIDAPPSEARCH}.

1. Add an XML layout called \texttt{emptyview.xml}, and do not add anything to the resulting autogenerated XML file.

2. Open the \texttt{hybridapps\_list.xml} file for editing and add the following tag above the \texttt{ListView} tag:

   \begin{verbatim}
   <EditText
       android:hint="@string/SEARCH_HINT"
       android:id="@+id/EditTextSearchHybridAppList"
       android:layout_width="match_parent"
       android:layout_height="47dp" />
   \end{verbatim}

3. Open \texttt{...\Values\Strings.xml} and, between the \texttt{<resource>} and \texttt{</resource>} tags, add:

   \begin{verbatim}
   <string name="SEARCH_HINT">search</string>
   \end{verbatim}

4. Copy the \texttt{UiHybridAppScreen.java} file to your own class name, for example, \texttt{SearchableAppScreen.java} and open it for editing.

   a) Add these import statements:

   \begin{verbatim}
   import android.widget.EditText;
   import android.textEditable);
   import android.text.TextWatcher;
   \end{verbatim}

   b) Add the following code to the end of the \texttt{onCreate} method:

   \begin{verbatim}
   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
       before, int count) {}
       public void onTextChanged( CharSequence s, int start, int
       before, int count) {}
   });
   \end{verbatim}

   c) Add this member variable to the \texttt{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:

```java
public void notifyDataSetChanged()
{
    search();
    super.notifyDataSetChanged();
}
public void setSearch( String sSearchFor )
{
    m_sToSearchFor = sSearchFor;
}
```

h) Add this member variable to the UiHybridAppScreen class:

```java
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.

   (Android only) That class must extend Activity.

6. (Android only) 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) Open the Android SDK Manager. If you are using Eclipse choose **Window > AVD Manager**.
   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.

Upgrading the PhoneGap Library Used by the Android Hybrid Web Container

SAP Mobile Platform includes the Cordova (PhoneGap) 2.0 libraries. Follow these steps if you want to upgrade the Android Hybrid Web Container to a more recent version of the Cordova library.

This procedure describes upgrading the Cordova library from version 2.0.0 to version 2.9.0. The steps to upgrade to other versions differ slightly. Since the Hybrid Web Container template project does not include the source code for building **HWCLib.jar**, the ability to upgrade Cordova to newer versions is limited, and certain new Cordova features may not work properly in Hybrid Web Container project.
Note: Upgrading the Hybrid Web Container container to use Cordova 3.0.0 is not supported because the Hybrid Web Container project does not work with Cordova 3.0.0 CLI.

1. Download phonegap 2.9.0 from phonegap.com, and unzip it to a local folder.
2. Open Eclipse and import the HWC template project.
3. Expand the HWC template project, and delete the cordova-2.0.0.jar file from the libs folder. Copy the cordova-2.9.0.jar file from the unzipped phonegap2.9.0 \lib\android\ folder, and copy it to the libs folder.
4. Right click the Hybrid Web Container project and click the Properties menu. Select Java Build Path > Libraries.
5. Select the cordova-2.0.0.jar file, then select Remove, to remove the old jar file.
6. Select Add JARs… and expand the HybridWebContainer\libs node. Select the new cordova-2.9.0.jar file, and click OK to confirm the selection.
7. Select OK to close the “Properties” dialog.
8. Update the private void initWebView() method.
   private void initWebView() calls the super.loadUrlWithData(sBaseURL, abData) Cordova method, which no longer exists in cordova-2.9.0. Change this method to call the super.loadUrl(sBaseURL) method instead.

   Open the UiHybridAppContainer.java class and navigate to the private void initWebView() method and make this change:

   // PhoneGap Change: We must call through PhoneGap to load the URL
   if ( USE_PHONEGAP )
   {
       // PhoneGap may timeout loading the web page
       super.setIntegerProperty( "loadUrlTimeoutValue", 300000 );

       // PhoneGap will load the URL
       super.loadUrl(sBaseURL);
   }
   else
   {
       m_oWebView.loadDataWithBaseURL( sBaseURL, new String( abData ), null, "utf-8", null );
   }

9. Clean the HWC project and have Eclipse build the HWC project.
10. If the cordova.js file is used in your HybridApp.js app, you must also update cordova.js to the one provided with the new cordova library.

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.

<table>
<thead>
<tr>
<th>Touch Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_COLORS</td>
<td>Use custom colors for the Hybrid Web Container.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_FONTS</td>
<td>Use custom fonts in the Hybrid Web Container.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_BRAND</td>
<td>Change application name, copyright, and developer information.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_SPLASHSCREEN</td>
<td>Add a splash screen to the Hybrid Web Container.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_DEFAULTSETTINGS</td>
<td>Set the defaults for the Settings screen.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_PRESETSETTINGS</td>
<td>Hard-code Settings screen options so they do not show up on the device, preventing the user from changing the settings.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_PIN</td>
<td>Use for PIN screen customizations, or to remove the PIN screen.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_SORTING</td>
<td>Sort application messages based on a variety of criteria.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_FILTERING</td>
<td>Filter the message list so only messages meeting certain criteria are shown.</td>
</tr>
<tr>
<td>Touch Point</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSORT</td>
<td>Customize the criteria for sorting the Hybrid App list.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPSearch</td>
<td>Make the list of Hybrid App packages searchable.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPLIST</td>
<td>Customize the Hybrid App package list appearance.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS</td>
<td>Create categorized views of the Hybrid App packages.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_HTTPHEADERS</td>
<td>Set HTTP headers for the BlackBerry Hybrid Web Container to include authentication tokens.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_MULTIHWC</td>
<td>Install more than one Hybrid Web Container on one device.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_PREPACKAGEN_APP</td>
<td>Run the Hybrid Web Container as a single Hybrid App.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_PUSH_NOTIFICATION</td>
<td>Customize the way the Hybrid Web Container handles push notifications.</td>
</tr>
<tr>
<td>BLACKBERRY_CUSTOMIZATION_POINT_ANONYMOUS_USER</td>
<td>Returns whether or not anonymous user login is supported. Change to YES to allow clients to register anonymously.</td>
</tr>
</tbody>
</table>

**Note:** For this to work, the HWC application connection template must be configured to use the anonymous security configuration. See Application Connection Templates in SAP Control Center for SAP Mobile Platform.

**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):
   ```java
   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.
   g) Close the BlackBerry_App_Descriptor.xml file.
   h) Open the HybridWebContainer.java file for editing.
   i) Add this line at the beginning of the postEvent method:
   ```java
   Brand.OEM_ENGINE_EXE_NAME = "HybridWebContainer";
   ```
   Replace HybridWebContainer with the name you used in step 4d.
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:
   ```java
   return SplashScreen.class;
   ```

   Your class must extend `MainScreen`, call `pushScreen` on itself so that it appears, then `popScreen` on itself when it is finished.

```java
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;
        }
    }
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).

   ```java
   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 );
}
```
3. To change the background color of the Hybrid Web Container:
   a) Add these import statements to the AppScreen.java file:

```java
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:

```java
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 HWCScreen.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:

```java
// Connection Header
m_oConnection = null;
M_oConnection = new LabelField( m_res.getString( HybridWebContainerResource.IDS_CONNECTION ),
    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.

   ```java
   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:

   ```java
   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:

   ```java
   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;

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!" );
    }
3. To change the font for the names of the Hybrid Apps in the list of Hybrid Apps:
   a) Add the font file (.ttf file) to the resources folder of the HybridWebContainer project.
   b) Open the AppScreen.java file for editing.
   c) 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.

   ```java
   // 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.getDisplayName(), 2 * iMargin + iBitmap.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. Double-click on the file BlackBerry_App_Descriptor.xml to open it.
2. In the Application 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 may 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:

   ```java
   public class CustomizationHelper {
   ....
   public final String getApplicationIndicatorIconName() {
     //return HWCMessagesScreen.INDICATOR_PNG; return "icon.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:

   ```java
   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 App Messages
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.
Hybrid Web Container Customization

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.

```java
/*
 * 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
   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().getHy
bridAppComparator() );

       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 );

       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(); )
   {
      }
private void updateScreen()
{
    // have to do stuff to the UI on a separate thread
    UiApplication.getUiApplication().invokeLater()
    
    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 ) );

private MenuItem m_mniSettings =
    new MenuItem( m_res.getString(HybridWebContainerResource.IDS_SETTINGS),
        100001, 10)
{
    public void run()
    {
        XmlHybridApp.startHybridAppSettings(false);
    }
};

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() + ".");
    }
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:
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:

```java
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:

```java
public void drawListRow(ListField listField, Graphics graphics, int index, int y, int width) {
    // y parameter is already offset to center
    int iOffset = (listField.getRowHeight() -
        getFont().getHeight()) >> 1;
    // HybridApp oApp = ( HybridApp )
    m_oApps.elementAt( index );
    // BLACKBERRY_CUSTOMIZATION_POINT_HYBRIDAPPLIST
    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;
7. Replace the `onHybridAppAdded` method in the `HybridAppsListener` with this modified version:

```java
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:

```java
private String m_sSearchFor;
```

2. Add the following code in the constructor of `AppScreen`, before the line that says `// Add list field to screen`:

```java
//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:

```java
// 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 )
```

---

Developer Guide: Hybrid Apps 825
Hybrid Web Container Customization

```java
// 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:

```java
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.

**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.

**Upgrading the PhoneGap Library Used by the BlackBerry Hybrid Web Container**

SAP Mobile Platform includes the Cordova (PhoneGap) 2.0 libraries. Follow these steps if you want to upgrade the BlackBerry Hybrid Web Container to a more recent version of the Cordova library.

This procedure describes upgrading the Cordova library from version 2.0.0 to version 2.9.0. The steps to upgrade to other versions differ slightly. Since the Hybrid Web Container template project does not include the source code for building `HybridAppLib.jar`, the ability to upgrade Cordova to newer versions is limited, and certain new Cordova features may not work properly in Hybrid Web Container project.

**Note:** Upgrading the Hybrid Web Container container to use Cordova 3.0.0 is not supported because the Hybrid Web Container project does not work with Cordova 3.0.0 CLI.

1. Download phonegap 2.9.0 from [phonegap.com](http://phonegap.com), and unzip it to a local folder.
2. After unzipping the `phonegap2.9.0` zip file, go into the `\blackberry\bbos\framework\ext` folder, and import the project to Eclipse blackberry plugin to compile/package the source code. The Cordova.jar can be found under the `deliverables` directory, in your project.
3. Open Eclipse and import the HWC template project.
4. Expand the HWC template project, and delete the `PhoneGapExtension.jar` file from the `libs` folder. Copy the `cordova.jar` file built above and copy it to the `libs` folder.
5. Right-click the Hybrid Web Container project and click the *Properties* menu. Select *Java Build Path > Libraries*. Select the `PhoneGapExtension.jar` file, then remove the old jar file.
6. Select *Add JARs...* and expand the `HybridWebContainer\libs` node. Select the new `cordova-2.9.0.jar` file, and click *OK* to confirm the selection.
7. Right-click the Hybrid Web Container project and click the *Properties* menu. Select *Java Build Path > Order and Export*. Select the checkbox for `Cordova.jar` and *OK* to close the “Properties” dialog.
8. Update the `com.sybase.hwc.amp.HWCBrowserFieldListener.java` class, since the constructor of class `CordovaExtension` was changed in 2.9.0.

Open the `com.sybase.hwc.amp.HWCBrowserFieldListener.java` class and make this change:

```
***************line 323**************
if( extension instanceof CordovaExtension )
```
{ 
    extension = new 
    CordovaExtension(getClass().getResourceAsStream("/xml/ 
    plugins.xml")); 
}

Change to:

if( extension instanceof CordovaExtension )
{
    extension = new CordovaExtension();
}

***************************************

9. Update the com.sybase.hwc.amp.HWCWidgetConfigImpl.java class:

**********line 175**********************
widgetExtensions.addElement( new 
    CordovaExtension(this.getClass().getResourceAsStream("/xml/ 
    plugins.xml")) );
Change to:
widgetExtensions.addElement( new CordovaExtension() );

***************************************

10. Clean the HWC project and have Eclipse build the HWC project.

11. If the cordova.js file is used in your HybridApp.js app, you must also update 
cordova.js to the one provided with the new cordova library.

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.

<table>
<thead>
<tr>
<th>Touch Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOS_CUSTOMIZATION_POINT_PRESET-SETTINGS</td>
<td>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.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_DEFAULT-SETTINGS</td>
<td>Set the defaults for the Settings screen.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_PREPACKAGED_APP</td>
<td>Include a prepackaged Hybrid App that launches automatically when the Hybrid Web Container starts.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_PIN</td>
<td>Use for PIN screen customizations, or to remove the PIN screen.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_SORTING</td>
<td>Sort Hybrid Apps or messages based on different criteria.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_FILTERING</td>
<td>Filter the list of Hybrid Apps or messages so only items meeting certain criteria are shown.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_HTTPHEADERS</td>
<td>Set HTTP headers for the iOS Hybrid Web Container to include authentication tokens.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_FONTS</td>
<td>Customize fonts in the Hybrid Web Container.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_SPLASH-SCREEN</td>
<td>Change the splash screen, or the length of time for which it is shown.</td>
</tr>
</tbody>
</table>
### 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.
2. Open the HWC.xcodeproj project with XCode 5 or above.
3. In the left project panel, select HWC > Resources > Images.xcassets.
4. Select AppIcon in the opening main view.
5. Select new images, and drag and drop the app icons you want to replace.

   **Note:** The new icon 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.

6. Rebuild the HWC.xcodeproj project.
7. From the Xcode menu, select Product > Clean.
8. Select Product > Build.
9. Click Run.

---

<table>
<thead>
<tr>
<th>Touch Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOS_CUSTOMIZATION_POINT_PUSH_NOTIFICATION</td>
<td>Customize how the Hybrid Web Container handles the push notification.</td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_ANONYMOUS_USER</td>
<td>Returns whether or not anonymous user support is being used. Change to YES to allow clients to register anonymously.</td>
</tr>
<tr>
<td><strong>Note:</strong> For this to work, the HWC application connection template must be configured to use the anonymous security configuration. See Application Connection Templates in SAP Control Center for SAP Mobile Platform.</td>
<td></td>
</tr>
<tr>
<td>IOS_CUSTOMIZATION_POINT_HTTPS_CLIENT_CERT_LISTENER</td>
<td>Customize how to handle client certificate authentication challenge.</td>
</tr>
</tbody>
</table>
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.
Hybrid Web Container Customization

- 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.

   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.
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`. 

---

SAP Mobile Platform
You do not need to explicitly add the new directory to the project, but you should verify it is there.


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.xcodeproj 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:
      ```bash
      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.
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.

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 update the application
bundle ID in the plist file. Finally, if your application needs to communicate with the Afaria client for provisioning your application or retrieving a certificate, you need to specify a unique URL scheme in the plist file. If your application does not need to communicate with the afaria client, then you should delete the “URL types” item from the same plist file.

1. Change the project name:
   a) In the Xcode Project Navigator, click on the root Hybrid Web Container element.
   b) With the Hybrid Web Container element highlighted click on the Hybrid Web Container text to rename.
   c) Change the name of the Hybrid Web Container element to your new project name.
   d) A window to rename project content items appears. Click Rename.

2. 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.

3. 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.

The container template project has a URL schema setting in the project plist file, which is used to communicate with Afaria client.

4. To avoid multiple container applications from interfering with each other when communicating with the Afaria client, the URL schema must be unique among all container applications that are installed on the device, otherwise, the application may be launched by the afaria client by mistake, or fail to launch altogether.
   a) In Xcode Project Navigator, find and open the HWC-Info.plist file, which is located in the Resources group folder.
   b) Expand the URL types item > Item 0 > URL Schemes item.
   c) Select Item 0, and change its value to a unique value among all other applications.
   d) 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 Documention 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**.

**Handling Client Certificate Challenge**
Customize the client certificate authentication challenge.

To customize the client certificate authentication challenge, uncomment the `onClientCertificateChallenge` method in the source `CustomizationHelper.m` file, then implement the logic to get a certificate identity as described by the comment within the method.

**Note:** It is not safe to display a model view within this method, as there may be another model view already displayed on the screen. Refer to the `[HWCApDelegete onClientCertificateChallenge]` method in `HWCApDelegate.m` for details of this model view issue.

**Upgrading the PhoneGap Library Used By the iOS Hybrid Web Container**

SAP Mobile Platform includes the Cordova 2.0 libraries. Follow these steps if you want to upgrade the iOS Hybrid Web Container to a more recent version of the Cordova library.

The Cordova library used by the Hybrid Web Container uses source code that has been modified from the original source, primarily because the original source does not support some Hybrid Web Container features. This procedure describes upgrading the Cordova library from version 2.0.0 to version 2.9.0. The steps to upgrade to other versions differ slightly. Since the Hybrid Web Container template project does not include the source code for building HWCLib.a, the ability to upgrade Cordova to newer versions is limited, and certain new Cordova features may not work properly in Hybrid Web Container project.
Hybrid Web Container Customization
Note: Upgrading the Hybrid Web Container container to use Cordova 3.0.0 is not supported
because the Hybrid Web Container project does not work with Cordova 3.0.0 CLI.
1. From a browser on your Mac, download phonegap 2.9.0 from phonegap.com, and unzip it
to a local folder.
2. Open terminal app, go to the unzipped folder phonegap-2.9.0/lib/ios/bin, and
create a Cordova prototype project using this command:
$ bash create /<path>/hello hello.example.com hello
3. From Xcode, open the generated project hello/hello.xcodeproj.
The cordovalib project will be used to update the Hybrid Web Container (HWC) project.
4. Open the HWC template project. In the HWC project's build setting, “Linking\Other Link
Flag” section, delete all references to cordovalib.a.
5. From Finder, delete the Cordovalib folder under the HWC template project folder.
Delete all cordovalib.a files from the HWC project's lib folder.
6. In the XCode HWC project, add the cordovalib project created in the hello project as a
dependent project. Update the HWC project's setting to include the cordova project in the
HWC’s “Target Dependencies” and “Link Binary With Libraries” section.
7. In the sub project cordovalib Navigator, find and open the CDVViewController.h
file located in the Classes/Cleaver group folder (Make sure only the cordovalib project
opens in HWC Xcode, otherwise these files are invisible in Navigator).
8. Add a UIViewController property declaration:
@property (nonatomic, strong) UIViewController* viewController;

9. Add these function declarations in CDVViewController:
-(void)setTheWebView: (UIWebView*) theWebView;
-(void)setTheViewController: (UIViewController*)
theViewController;

10. In the Xcode Project Navigator, find and open the CDVViewController.m file,
synthesizing the viewController property:
@synthesize viewController = _viewController;

11. Use #if 0 to comment out these portions of the viewDidLoad function in the
CDVViewController.m file:
#if 0
NSURL* appURL = nil;
NSString* loadErr = nil;
if ([self.startPage rangeOfString:@"://"].location != NSNotFound)
{ appURL = [NSURL URLWithString:self.startPage]; } else if
([self.wwwFolderName rangeOfString:@"://"].location !=
NSNotFound) { appURL = [NSURL URLWithString:[NSString
stringWithFormat:@"%@/%@", self.wwwFolderName,
self.startPage]]; } else {
NSString* startFilePath = [self.commandDelegate
pathForResource:self.startPage];

844

SAP Mobile Platform


12. Update the two registerPlugin functions in the CDVViewController.m file by replacing:

```
[plugin setViewController:self];
```

with:

```
[plugin setViewController: self.viewController];
```

13. Add the implementations for the two new methods mentioned previously in the CDVViewController.m file:

```
-(void) setTheWebView: (UIWebView*) theWebView {
    self.webView = theWebView;
}

-(void) setTheViewController: (UIViewController*)theViewController {
    self.viewController = (CDVViewController*)theViewController;
}
```

14. Add this line to the dealloc function in the CDVViewController.m file:

```
self.whitelist = nil;
```

15. Use `#if 0` to comment out the CreateGapView function implementation from the CDVViewController.m file.

```
-(void)createGapView
{
    #if 0
    CGRect webViewBounds = self.view.bounds;
    webViewBounds.origin = self.view.bounds.origin;
    if (!self.webView) { self.webView = [self newCordovaViewWithFrame:webViewBounds];
        self.webView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
        UIViewAutoresizingFlexibleHeight); [self.view addSubview:self.webView];
        [self.view sendSubviewToBack:self.webView]; _webViewDelegate = [
        [CDVWebViewDelegate alloc] initWithDelegate:self]; self.webView.delegate = _webViewDelegate; #endif
```

...
16. In the Xcode Project Navigator, find and open the CDVPlugin.m file, which is in the Classes/Commands group folder, and add this code at the very top of the dealloc function:

self.viewController = nil;

17. At the beginning of initializeAppAfterKeyVaultUnlocked method in the HWCAppDelegate.m file, call:

[super viewDidLoad];

18. Add Jsonkit back into the new Cordova library to avoid link errors. You can copy the jsonkit.h and jsonkit.m files from the PhoneGap 2.0.0 library project. Beginning with Cordova 2.4.0, the JsonKit is no longer included in cordova library, however, HWCLib.a uses some functions provided by Jsonkit.

19. Add Jsonkit.h and Jsonkit.m as source files in the Cordova library project.

20. Set the compile flag “-fno-objc-arc” for the jsonkit.m file in the “build phase”/“compile source” section. The new cordova library project is compiled with ARC enabled, but jsonkit.m does not support ARC.

21. Add jsonkit.h as a public header file in the Xcode cordovalib project’s “build phase”/“copy header”/“public file” section, so it can be found by the HWC project.

22. Delete cordova.plist from hwc project, and add config.xml created for the Hello project into the HWC project. Beginning with Cordova 2.3.0, config.xml replaces cordova.plist.

23. Delete www and capture.bundle in the HWC project to avoid build errors. Replace the file VERSION with the one created by the new Hello project.

24. Edit the config.xml file in the HWC project to enable httpproxy and applog plugin by adding these lines:

```xml
<feature name="AppLog">
  <param name="ios-package" value="AppLogPlugin"/>
</feature>
<feature name="HttpsProxy">
  <param name="ios-package" value="HttpsProxyPlugin"/>
</feature>
```

25. Replace this code in the HWCAppDelegate.m file in the HWC project:

```c
#ifdef CORDOVA_FRAMEWORK
#import <Cordova/CDVViewController.h>
#import <Cordova/CDVContacts.h>
#else
#import "CDVViewController.h"
#import "CDVContacts.h"
#endif
```
26. Add the AssetsLibrary, CoreMotion and imageIO frameworks to the “build phase”/“Link Binary With Libraries” section in the HWC project to avoid link errors.

27. Delete this line from the `initializeAppAfterKeyVaultUnlocked` function in the `HWCAppDelegate.m` file:

```swift
[CDVContacts setContactsAccessDelegate:self];
```

28. Delete these lines from the `HWCAppDelegate.m` file:

```swift
static BOOL s_bContactsChallengeInProgress = NO;
+(BOOL) isContactsChallengeInProgress{
    return s_bContactsChallengeInProgress;
}
(void) requestContactsAccess {
    #if __IPHONE_OS_VERSION_MAX_ALLOWED >= 60000
    NSArray* versionCompatibility = [[UIDevice currentDevice].systemVersion componentsSeparatedByString:@"."];
    NSInteger iMajorVersion = [[versionCompatibility objectAtIndex:0] intValue];
    if (iMajorVersion >= 6)
    {
        if (ABAddressBookGetAuthorizationStatus() == kABAuthorizationStatusNotDetermined)
        {
            s_bContactsChallengeInProgress = YES;
            ABAddressBookRef addressBookRef = ABAddressBookCreateWithOptions(NULL, NULL);
            ABAddressBookRequestAccessWithCompletion(addressBookRef, ^(bool granted, CFErrorRef error){
                s_bContactsChallengeInProgress = NO; });
            CFRelease(addressBookRef);
        }
    }
    while (s_bContactsChallengeInProgress)
    {
        NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
        [[NSRunLoop currentRunLoop] runUntilDate:[NSDate dateWithTimeIntervalSinceNow:1]]; [pool release];
    }
    #endif
}
```

29. Delete this line in the `applicationWillResignActive` method from the `HWCAppDelegate.m` file:

```swift
bPresentingContactsChallenge = [HWCAppDelegate isContactsChallengeInProgress];
```

30. Delete this line from the `HWCAppDelegate.h` file:
31. Delete the protocol “MissingFeaturesProvider” from the CDVViewController interface definition from the HWCAppDeleage.h file to avoid build errors.

32. Update cordova.js to the one provided with the new cordova library if cordova.js is used in your HybridApp.js app.

## 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`.

<table>
<thead>
<tr>
<th>Touch Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM_CUSTOMIZATION_POINT_BRAND</td>
<td>Change application name, copyright, and developer information in the About form.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_HYBRID-APPSEARCH</td>
<td>Make the list of Hybrid App packages searchable.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_HYBRID-APPLIST</td>
<td>Change the appearance of the Hybrid App package list.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS</td>
<td>Create categorized views of the Hybrid App packages.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_HYBRID-APPSORTING</td>
<td>Customize the criteria for sorting the Hybrid App package list.</td>
</tr>
<tr>
<td>Touch Point</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_MESSAGE-SORTING</td>
<td>Customize the criteria for sorting the message list.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_MESSAGE-FILTERING</td>
<td>Change the filter used to sort the list of messages.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_ANONYMOUS_USER</td>
<td>Indicates if the login mode is anonymous.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_DEFAULT-SETTINGS</td>
<td>Change default server settings.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_PRESET-SETTINGS</td>
<td>Hard-code settings for the Settings screen so they do not appear on the device. This prevents the user from changing the settings.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_HTTPHEADERS</td>
<td>Set HTTPS headers for the Windows Mobile Hybrid Web Container to include authentication tokens.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_HTTPERRORHANDLERS</td>
<td>Change the handling of HTTP errors.</td>
</tr>
<tr>
<td>WM_CUSTOMIZATION_POINT_TOKENERROR</td>
<td>Change how the client engine handles authentication token errors (for example, when a token expires).</td>
</tr>
</tbody>
</table>

**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:

- `ampiconindex.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 `ampiconindex.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:
   ```csharp
   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, MessageBoxIcon.DefaultButton.Button1);
   }
   ``

4. Rebuild and test your project.

**Adding aSplash 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.rex 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:

```csharp
public override ServerSettings DefaultServerSettings
{
    get
    {
        if (m_ServerSettings == null)
```csharp
private ServerSettings m_ServerSettings {

    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;
}
```

**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:
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:
   ```csharp
   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:
   ```csharp
   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:
   ```csharp
   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)
   ```
```csharp
{  
iCompareResult = x.ReceiveDate.compareTo(y.ReceiveDate);
}  
return iCompareResult;
}
3. Open the CustomizationHelper class in the CustomCode folder.
4. Override the MessageComparator using:
```csharp
public override IComparer<Message> MessageComparator  
{
    get { return new MessageComparer(); }
}
```csharp
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:
```csharp
public bool Select(Message subject)  
{  
    if (subject.Priority ==  
        MessageConsts.EMAIL_STATUS_IMPORTANCE_HIGH)  
    {  
        return false;  
    }  
    return true;
}
```csharp
3. Open the CustomizationHelper class in the CustomCode folder.
4. Override the MessageFilter method using:
```csharp
public override IFilter<Message> MessageFilter  
{
    get  
    {  
        return new MessageFilter();
    }
}
```csharp
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:

```csharp
internal virtual void OnClosingMessageListForm( MessageListForm form )
{
}
```

```csharp
internal virtual void OnClosingApplicationListForm( HybridWebAppListForm form )
{
}
```

```csharp
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:
   ```bash
   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 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.
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.

   ```csharp
   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:
   
   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:

   ```plaintext
   [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:

   ```plaintext
   [SourceDisksFiles.ARM]
   CMessagingClient.2.2.0.dll=1
   OBSetup.dll=1
   HWCA.exe=1
   HWCEngine.lnk=1
   ```
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,js,prepackage,
prepackage.css,prepackage.default,prepackage.en,prepackage.en_US,
prepackage.icon,prepackage.images,prepackage.js,prepackage.html

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:

```javascript
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>
<thead>
<tr>
<th>API</th>
<th>Object and Function</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>accelerometer</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• getCurrentAcceleration</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• watchAcceleration</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• clearWatch</td>
<td></td>
</tr>
<tr>
<td>Note:</td>
<td>On iOS, this function must be called after watchAcceleration.</td>
<td></td>
</tr>
<tr>
<td>Acceleration</td>
<td>• x</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• y</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• z</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• timeStamp</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• getPicture (Camera.PictureSourceType.CAMERA)</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• getPicture (Camera.PictureSourceType.PHOTOLIBRARY)</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• getPicture (Camera.PictureSourceType.SAVEDPHOTOALBUM)</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td>CameraOptions</td>
<td>• quality</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• dedestinationType.DATA_URL</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• dedestinationType.FILE_URI</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>FILE_URI is the default.</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>• allowEdit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• encodingType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• targetWidth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• targetHeight</td>
<td></td>
</tr>
<tr>
<td>Capture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Object and Function</td>
<td>Platform</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Capture</td>
<td>captureAudio</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• captureImage</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• captureVideo</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 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.</td>
<td></td>
</tr>
<tr>
<td>MediaFile</td>
<td>getFormatData</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td>Compass</td>
<td>compass</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• getCurrentHeading</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• watchHeading</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• clearWatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• watchHeadingFilter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>compass.Heading</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• magneticHeading</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• trueHeading</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• headingAccuracy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• timestamp</td>
<td></td>
</tr>
<tr>
<td>Connection</td>
<td>network.connection.type</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td>Contacts</td>
<td>contacts.create</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td>API</td>
<td>Object and Function</td>
<td>Platform</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>contacts.find</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>contact.clone</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Contacts.remove</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>On Android, there is a issue with contacts not being fully removed. See <a href="https://issues.apache.org/jira/browse/CB-75">https://issues.apache.org/jira/browse/CB-75</a>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contacts.save</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td><strong>Device</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Device.name</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Device.phonegap</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Device.platform</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Device.uuid</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
</tbody>
</table>
## Hybrid Web Container Customization

<table>
<thead>
<tr>
<th>API</th>
<th>Object and Function</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Device.version</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td>Events</td>
<td>Deviceready</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Pause</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>Resume</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>Online</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Offline</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Batterycritical</td>
<td>iOS</td>
</tr>
<tr>
<td></td>
<td>Batterylow</td>
<td>iOS</td>
</tr>
<tr>
<td></td>
<td>Batterystatus</td>
<td>iOS</td>
</tr>
<tr>
<td>Note:</td>
<td>On Android, PhoneGap 1.4.1, this does not work due to a known issue. See <a href="https://issues.apache.org/jira/browse/CB-173">https://issues.apache.org/jira/browse/CB-173</a>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Menubutton</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>Searchbutton</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>File</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Object and Function</td>
<td>Platform</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>DirectoryEntry</td>
<td>• copyTo</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• moveTo</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• toURI</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• remove</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>• removeRecursively</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• getParent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• createReader</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• getDirectory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• getFile</td>
<td></td>
</tr>
<tr>
<td>FileEntry</td>
<td>• copyTo</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• moveTo</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• toURI</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• remove</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>• getParent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• createWriter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• file</td>
<td></td>
</tr>
<tr>
<td>FileReader</td>
<td>• abort</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• readAsDataURL</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• readAsText</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td>FileWriter</td>
<td>• abort</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• seek</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• truncate</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• write</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td>DirectoryReader</td>
<td>• readEntries</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
</tbody>
</table>
### API

<table>
<thead>
<tr>
<th><strong>API</strong></th>
<th><strong>Object and Function</strong></th>
<th><strong>Platform</strong></th>
</tr>
</thead>
</table>
| LocalFileSystem | • requestFileSystem  
• resolveLocalFileSystemURI | • Android  
• BlackBerry  
• iOS  
• Windows Mobile |
| FileTransfer | • upload  
• download | • Android  
• BlackBerry  
• iOS  
• Windows Mobile |
| Geolocation | geolocation | • Android  
• BlackBerry  
• iOS  
• Windows Mobile |
|         | • getCurrentPosition | **Note**: This function does not work on the Android Galaxy Tab P1000 device. |
|         | • watchPosition  
• clearWatch | |
| Position | • coords  
• timestamp | • Android  
• BlackBerry  
• iOS  
• Windows Mobile |
<table>
<thead>
<tr>
<th>API</th>
<th>Object and Function</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coordinates</td>
<td>• latitude</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>• longitude</td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td>• altitude</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>• accuracy</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• altitudeAccuracy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• heading</td>
<td>• Android only. The returned heading property is always null.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• speed</td>
<td>• Android, the returned speed property is always null.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media</td>
<td>Media.play</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Media.pause</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Media.stop</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td>Media.release</td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Media.record</td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Media.startRecord</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td>API</td>
<td>Object and Function</td>
<td>Platform</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td></td>
<td>Media.stopRecord</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Media.getCurrentPosition</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Media.seekTo</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Media.getDuration</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td>Note:</td>
<td>On Android, this function</td>
<td></td>
</tr>
<tr>
<td></td>
<td>returns a value without</td>
<td></td>
</tr>
<tr>
<td></td>
<td>an error but always returns</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1, which indicates duration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is not available.</td>
<td></td>
</tr>
<tr>
<td>Notification</td>
<td>Notification.beep</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Notification.confirm</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td>Notification.alert</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td></td>
<td>Notification.vibrate</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows Mobile</td>
</tr>
<tr>
<td>Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Object and Function</td>
<td>Platform</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>window</td>
<td>• OpenDatabase</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td>Database</td>
<td>• transaction</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
<tr>
<td>SQLTransaction</td>
<td>• executeSQL</td>
<td>• Android</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BlackBerry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iOS</td>
</tr>
</tbody>
</table>

**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:

```javascript
var db = window.openDatabase("aName1", "1.0", "aName1", 0);
db = window.openDatabase("aName2", "1.0", "aName2", 200000);
```

SQLResultSet

| SQLResultSet | • insertid                           | • Android              |
|              | • rowAffected                         | • BlackBerry           |
|              |                                         | • iOS                  |

**Note:** The returned SQLResultSet object does not contain a `rowAffected` property, as the PhoneGap API states. Instead, use `rowsAffected`.

| SQLResultSetList | • item                              | • Android              |
|                 | • length                            | • BlackBerry           |
|                 |                                     | • iOS                  |

<table>
<thead>
<tr>
<th>SQLError</th>
<th>• code</th>
<th>• message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|           |       |           |
|           |       |           |
### 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\MobileSDK\version\HybridApp\Containers\Platform directories`.

```javascript
function loadPhoneGap() {
    var jsfile = null;
    var pre = "";
    var language = hw.getURLParam("lang");
    if (!language == undefined) && (language.length > 0)){
        pre = "./";
    } else if (hw.isAndroid()) {
        jsfile = pre + "js/android/cordova-2.0.0.javascript";
    } else if (hw.isIOS()) {
        jsfile = pre + "js/ios/cordova-2.0.0.javascript";
    } else if (hw.isBlackberry()) {
        jsfile = pre + "js/blackberry/cordova-2.0.0.javascript";
    } else 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:
  ```javascript
  function phoneGapIsready() {
      showAlertDialog("PhoneGap is ready");
  }
  ```

- This customization detects when the PhoneGap initialization occurs and displays your notification:
  ```javascript
  function hwc.customAfterHybridAppLoad() {
      document.addEventListener("deviceready",
      phoneGapIsReady, false);
  }
  ```

Alternatively, detect the initialization directly in your Hybrid App HTML file:
```html
<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

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\MobileSDK\version\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);
 * }
 * @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());
            } catch (Exception e) {
                Log.i("DirectoryListPlugin", "Error: "
                + e.getMessage());
            }
        }
    }
}
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();
}


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:

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](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:
   
   ```java
   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.


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:

```javascript
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:

```javascript
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:

```javascript
var gActIndicator = true; // global variable

function customAfterMenuItemClick(screen, menuItem) {
    if (screen === "Start" && menuItem === "networkActivityIndicator") {
        window.networkActivityIndicator.set( gActIndicator, aiSuccess, aiFail );
        // Toggle the network activity indicator each time plugin is selected
        if ( gActIndicator )
            gActIndicator = false;
        else
            gActIndicator = true;
        return false;
    }
}

function aiSuccess() {
    alert("Successfully enabled activity indicator");
}

function aiFail() {
```
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.beep 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:

```csharp
using WMGapClassLib.Cordova;
namespace Sybase.Hwc.CustomCode
{
    public class Calculator : PluginBase
    {
        public void sum(Session session,
        {
            try
```
2. Open the file `Plugins.xml`, which is located in the HybridWebContainer project, and add the custom plug-in:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<plugins>
  <plugin id="showcertpicker"
    class="Sybase.Hwc.CertificationPickerPlugin"/>
  <plugin id="Calculator"
</plugins>
```

3. Open the `Custom.js` file for editing and add this method:

```javascript
function calculateSum(x, y, successCb, errorCb) {
  cordova.require('cordova/exec') (successCb, errorCb, "Calculator", "sum", {
    x: document.getElementById('x').value, y: document.getElementById('y').value });
}
```

4. Call this JavaScript method somewhere else to get the result:

```javascript
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.

**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
   <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 Hybrid Web Container, 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
   \MobileSDK23\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:

   ```java
   @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.

Using the HTTPS Proxy Exposed by the PhoneGap Plugin

PhoneGap JavaScript application users that want to send an AJAX request to a HTTPS server, where the embedded browser does not support SSL, can use the HTTPS proxy exposed by the PhoneGap plugin.

The HTTPS proxy supports both HTTP and HTTPS connections. HTTPS connections are supported for both server side certificate validation and client side certificate, or either, or none. A keystore/certificate from either system, file, or Afaria is loaded when first required. Compared to the lifecycle of certificates, the lifecycle of certificates in memory is very short. The user cannot refresh newly loaded keystores/certificates. If a new certificate is added or updated, you must restart the application to include it.

PhoneGap HTTPS proxy connection properties

The JavaScript API definition of the HTTPS proxy is: Namespace: HttpsConnection.

This table describes the platform dependent methods, error codes, and object definitions. Its implementation is platform dependent so every platform should provide its own version of the JavaScript code.

<table>
<thead>
<tr>
<th>Method/definition/error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CertificateFromFile (path, password, certificateKey)</td>
<td>Create a certificate descriptor for the certificate from a file. Calling this method does not immediately load the certificate.</td>
</tr>
<tr>
<td>CertificateFromAfaria (cn, challengeCode)</td>
<td>Create a certificate descriptor for the certificate from the Afaria server. Calling this method does not immediately load the certificate.</td>
</tr>
<tr>
<td></td>
<td>• path – path of the keystore file</td>
</tr>
<tr>
<td></td>
<td>• password – password of the keystore</td>
</tr>
<tr>
<td></td>
<td>• certificateKey – certificate key of the certificate in the keystore, which is the alias in the Java keytool</td>
</tr>
<tr>
<td></td>
<td>• cn – common name of the certificate</td>
</tr>
<tr>
<td></td>
<td>• challengeCode – challenge code to the Afaria server</td>
</tr>
<tr>
<td>Method/definition/error code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| CertificateFromStore (certificateKey) | Create a certificate descriptor for certificate from files. Calling this method does not immediately load the certificate.  
  - certificateKey – certificate key of the certificate in the system keystore, which is the alias in the Java keytool |
| deleteCertificateFromStore(certificateKey) | (iOS only) Delete a cached certificate from the keychain. The iOS client always tries the cached certificate first if available, before requesting the certificate from the Afaria server or loading the certificate from the file system.  
  In cases where the cached certificate is no longer valid, use this method to delete it from the keychain. |
<table>
<thead>
<tr>
<th>Method/definition/error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get(url, header, successCB, errorCB, userId, password, timeout, certSource)</td>
<td>Send a HTTP request of the GET method.</td>
</tr>
<tr>
<td>• url – the full URL in format https://...[:port]</td>
<td></td>
</tr>
<tr>
<td>• header – header of the request in JSON Object</td>
<td></td>
</tr>
<tr>
<td>• successCB – callback method upon success. Its parameter is a string encoded JSON object with these fields:</td>
<td></td>
</tr>
<tr>
<td><code>{status: number; headers: JSON object with string fields; responseText: Optional if the requested data is text; responseBase64: Optional if the requested data is binary}</code></td>
<td></td>
</tr>
<tr>
<td>Callers must provide this method otherwise an exception is thrown.</td>
<td></td>
</tr>
<tr>
<td>• errorCB – callback method upon failure. Its parameter is an object with these fields:</td>
<td></td>
</tr>
<tr>
<td><code>{errorCode: number; description: string; nativeErrorCode: number}</code></td>
<td></td>
</tr>
<tr>
<td>Callers must provide this method otherwise an exception is thrown.</td>
<td></td>
</tr>
<tr>
<td>• userID – (Optional) for basic authentication</td>
<td></td>
</tr>
<tr>
<td>• password – (Optional) for basic authentication</td>
<td></td>
</tr>
<tr>
<td>• timeout – (Optional) in seconds</td>
<td></td>
</tr>
<tr>
<td>• certSource – (Optional) The JavaScript certificate description object</td>
<td></td>
</tr>
<tr>
<td>Returns a JavaScript object that contains an abort() method to abort the current connection</td>
<td></td>
</tr>
<tr>
<td>Method/definition/error code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>sendRequest(method, url, header, requestBody, successCB, errorCB, userId, password, timeout, certSource)</td>
<td>Send a generic HTTP request to the server.</td>
</tr>
<tr>
<td>• url – the full URL in format https://...[:port]</td>
<td></td>
</tr>
<tr>
<td>• header – header of the request in JSON Object</td>
<td></td>
</tr>
<tr>
<td>• requestBody – data as a string value to be sent to server with the request.</td>
<td></td>
</tr>
<tr>
<td>• successCB – callback method upon success. Its parameter is a string encoded JSON object with these fields:</td>
<td></td>
</tr>
<tr>
<td>{status: number; headers: JSON object with string fields; responseText: Optional if the requested data is text; responseBase64: Optional if the requested data is binary}</td>
<td></td>
</tr>
<tr>
<td>Callers must provide this method otherwise an exception is thrown.</td>
<td></td>
</tr>
<tr>
<td>• errorCB – callback method upon failure. Its parameter is an object with these fields:</td>
<td></td>
</tr>
<tr>
<td>{errorCode: number; description: string; nativeErrorCode: number}</td>
<td></td>
</tr>
<tr>
<td>Callers must provide this method otherwise an exception is thrown.</td>
<td></td>
</tr>
<tr>
<td>• userID – (Optional) for basic authentication</td>
<td></td>
</tr>
<tr>
<td>• password – (Optional) for basic authentication</td>
<td></td>
</tr>
<tr>
<td>• timeout – (Optional) in seconds</td>
<td></td>
</tr>
<tr>
<td>• certSource – (Optional) The JavaScript certificate description object</td>
<td></td>
</tr>
<tr>
<td>Returns a JavaScript object that contains an abort() method to abort the current connection</td>
<td></td>
</tr>
<tr>
<td>ERR_UNKNOWN</td>
<td>The operation failed with an unknown error.</td>
</tr>
<tr>
<td>ERR_INVALID_PARAMETER_VALUE</td>
<td>The operation has an invalid parameter.</td>
</tr>
<tr>
<td>ERR_MISSING_PARAMETER</td>
<td>The operation failed because of a missing parameter.</td>
</tr>
</tbody>
</table>
### Method/definition/error code

<table>
<thead>
<tr>
<th>Method/definition/error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_NO_SUCH_ACTION</td>
<td>There is no such Cordova action for the current service.</td>
</tr>
<tr>
<td>ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED</td>
<td>Certificate from file keystore is not supported on the current platform.</td>
</tr>
<tr>
<td>ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED</td>
<td>Certificate from system keystore is not supported on the current platform.</td>
</tr>
<tr>
<td>ERR_AFARIA_CERTIFICATE_SOURCE_UNSUPPORTED</td>
<td>Certificate from Afaria server is not supported on the current platform.</td>
</tr>
<tr>
<td>ERR_CERTIFICATE_ALIAS_NOT_FOUND</td>
<td>The certificate with given alias could not be found.</td>
</tr>
<tr>
<td>ERR_CERTIFICATE_FILE_NOT_EXIST</td>
<td>The certificate file could not be found.</td>
</tr>
<tr>
<td>ERR_CERTIFICATE_INVALID_FILE_FORMAT</td>
<td>Incorrect certificate file format.</td>
</tr>
<tr>
<td>ERR_GET_CERTIFICATE_FAILED</td>
<td>Failed in getting certificate.</td>
</tr>
<tr>
<td>ERR_CLIENT_CERTIFICATE_VALIDATION</td>
<td>The provided certificate failed server-side validation.</td>
</tr>
<tr>
<td>ERR_SERVER_CERTIFICATE_VALIDATION</td>
<td>The server certificate failed client-side validation.</td>
</tr>
<tr>
<td>ERR_SERVER_REQUEST_FAILED (-110)</td>
<td>Exception message reported by httpURLConnection. The native code should contain the specific error information.</td>
</tr>
<tr>
<td>ERR_HTTP_TIMEOUT</td>
<td>Timeout error while connecting to the server.</td>
</tr>
</tbody>
</table>

### Requirements and Limitations

Additional requirements and limitations for using the PhoneGap HTTPS proxy.

- Although the embedded browser in BlackBerry supports HTTPS requests, this implementation is required to support Afaria.
- System keystore for Android versions prior to 4.0 is not supported.
- Multiple instances of certificates/keystores are supported for certificate/keystore from files and Afaria.
- The plugin in a production environment denies any trust confirmation for server side certificates.
- The API format for how to call cordova.exec(…) is not defined, because it is transparent to users, and is platform dependent, so implementation is left to plugin developers.
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)

**Note:** Do not send DCNs with the same Message ID from different back-end systems. Make sure to use different Message IDs for different DCNs.

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—for example, if using auto-registration, the user name used to register the device is the "device user name"
or the "application connection user name" (in either case this refers to the user name used to register the device). And the WF-DCN "To" field can use this name to push the message to the device. Additionally, there is another package user name which is established during activation of the Hybrid App. The package user name can also be used as the "To" field for pushing the message.

For manually registered devices, the WF-DCN is pushed based on the package user name established after Hybrid App activation.

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:

```json
{
  "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.

Device Registration

For Security reasons SAP Mobile Server pushes WF-DCN notifications only to auto-registered device users. For example, if there are two application connections with the same name and one is auto-registered and the other is manually registered, SAP Mobile Server pushes the WF-DCN notification only to the auto-registered device. For manually registered devices, the WF-DCN is pushed based on the package user, which is created when the Hybrid App is assigned to the device and the user activates the Hybrid App using EIS user name/password.

See the topic Registering Applications, Devices, and Users in the Security Guide.

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=supPwd&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_request=<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:

```json
{
  "result": [
    {
      "general dcn result"
    },
    {
      "general dcn result"
    }
  
```
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:

```json
```

For example:
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:

```json
{"id":"","op":"","subject":"","to":"","from":"","read":"","priority ":"","body":"","data":{"id":"","pkg":"Package","messages": [{"id":"2","mbo":"MBO","op":"insert","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:

```json
{"id":"1137","op":"insert","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":"insert","cols": [{"WORKITEM":"1470577","USERNAME":"perf0111","DESCRIPTION":"cc","DECISION":"test"} ] }, {"id":"6","mbo":"Alternatives","op":"insert","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.

```java
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 wfdcn_request = "{"id":"dcntest_69","op":"upsert","subject":"dept_id = 1300","to":"perf0111","from":"SAP Leave WorkFlow","read":false,"priority":true,"body":",TaskID:, WIID:000001468382, USER:perf0111#END\n"};
        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 751

A
abortion
  method 72
activationRequired
  method 149, 249
ActiveSync, installing and configuring 743
addAppInstallationListener
  method 149
addAppListener
  method 150
addConnectionListener
  method 151
addLogListener
  method 153
addMenuItem
  method 212
addMenuItemCollection
  method 154
addMessageListener
  method 155
addPushNotificationListener
  method 157
Advanced Encryption Standard 696
AES
  See also Advanced Encryption Standard
  AES-128 698
  AES-256 694
Afaria® Security Manager 699
Alert Message property 757
alertDialogCallbackFunction
  method 73
Alerts property 757
Android emulator
  configuring 735
Android Hybrid Web Container
  installing 735
Android Hybrid Web Container customization
  setting HTTP headers 775
ANDROID_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS 780
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH 794
anonymous
  namespace 68
API functions
  credential functions 661
  general utility functions 661
  Hybrid App native device functions 661
  Hybrid App UI functions 661
  Hybrid App utility functions 661
  Hybrid App validation functions 661
  message data functions 661
API.js 661
APNS 754
APNS Device Token property 757
APP_ADDED
  member 126
APP_REFRESH
  member 126
APP_REMOVED
  member 127
APP_UPDATED
  member 127
AppInstallationListener
  method 73, 74
Apple push notification properties 757
Apple push notification, configuring 756
application 719
application ID
  guidelines 720
ApplicationListener
  method 75
AppLog
  namespace 84
AppLogErrorCallbackParameter
  class 70
arbitrary metadata 608
ATTACHMENT_NOT_DOWNLOADED
  member 122
AttachmentViewer control
  image limitations 604

B
BAD_OPTIONS
  member 142
Badges property 757
Index

BB7_MAX_STRING_STORAGE_LENGTH
  member 117
BlackBerry 739
  configuring the simulator 739, 740
BlackBerry Desktop Manager 738
BlackBerry Hybrid Web Container 739
  adding a new language 804
  adding a splash screen 802
  BLACKBERRY_CUSTOMIZATION_POINT_T_SPLASHSCREEN 802
  default behavior customization 810
  setting HTTP headers 813
  using custom colors 805

C

CACHE_FIRST
  member 133
cached data lookup pattern
data flow diagram 644
  overview 644
Callbacks.js 666
  source file 253
CallbackSet 666
Camera.js
  source file 257
categorized views 780
certificate picker 686
CERTIFICATE_NOT_SELECTED
  member 122
Certificate.js 669
  source file 268
CertificateFromAfaria
  class 98
CertificateFromFile
  class 99
CertificateFromStore
  class 99
certificateLabels
  method 158
certificates
  for context variables 723
CertificateStore
  method 158
class
  AppLogErrorCallbackParameter 70
  CertificateFromAfaria 98
  CertificateFromFile 99
  CertificateFromStore 99
  sendRequestErrorCBParameter 70
  sendRequestSuccessCBParameter 71
SUPStorage 116
SUPStorageException 120
clear
  method 117
clearCache
  method 164, 249
clearCacheItem
  method 164, 249
ClientIconIndex 608
ClientVariables
  method 165
ClientVariablesException
  method 168
close
  method 168
closeWorkflow
  method 250
complete
  method 76
conditional navigation 680
conditional start 682
CONNECTED
  member 127
connection settings
  configuring 746
  default 835
  device 746
  Hybrid Web Container 746
CONNECTION_CONNECTED
  member 128
CONNECTION_DISCONNECTED
  member 128
CONNECTION_ERROR
  member 128
CONNECTION_OTHER
  member 129
CONNECTION_RETRIEVED_ITEMS
  member 129
ConnectionSettings
  method 168
ConnectionStateListener
  method 77
connectToServer
  method 170
content security 694
  Android 694
  BlackBerry 696
  iOS 698
Windows Mobile 699
content type preference, changing 702
context variables 608, 724
configuring 723
convertLocalTimeToUtc
method 171
convertUtcToLocalTime
method 171
credential functions 661
credentials, static and dynamic 684
CredentialsCache 608
Custom.js 653
custom.js file
methods 671
customAfterNavigateForward 671
customAfterReportErrorFromNative 674
customAfterShowScreen 671
customAfterSubmit 671
customAfterWorkflowLoad 671
customBeforeMenuItemActivate 671
customBeforeNavigateBackward 671
customBeforeNavigateForward 671
customBeforeReportErrorFromNative 674
customBeforeShowScreen 671
customBeforeSubmit 671
customBeforeWorkflowLoad 671
CustomIcon
method 172
customization touch points
ANDROID_CUSTOMIZATION_POINT_DE
FAULTSETTINGS 772
customValidateScreen 671

D
data change notification 895, 896
GET 893
JSON format 893
POST 893
request response 896
Data.js 43
data.js library 64
Data.js library 43
DCN 896
debugging 712
default locale, creating 703
DEFAULT_CUSTOM_ICON_INDEX
member 129
defining an MBO
for cached data lookup 645
for real-time data lookup 635
deleteCertificateFromStore
method 100
DeleteProcessedMessages 608
Delivery Threshold property 757
deploy 631
device platforms 631
device users
assigning Hybrid App packages 722
DEVICE_NOT_CONNECTED
member 123
devices
Apple push notification properties 757
DISCONNECTED
member 130
disconnectFromServer
method 175
documentation roadmap 1
Dynamic authentication 686

E
e2eTrace
member 130
editing
locale properties file 706
Enable property 757
encoding type, changing 702
encryption key length 696
encryption policy 677
ERR_UNKNOWN
member 86
errorCallbackFunction
method 77
expireCredentials
method 175, 250
ExternalResource.js
text file 283

FAIL_TO_SAVE_CERTIFICATE
member 123
FAIL_TO_SAVE_CREDENTIAL
member 123
file association 702
FILENAME_NO_EXTENSION
member 123
files
  source 253
filtering 838
findByParameter
  binding to a menu item 635
findByParameter object query 639
functions
  resource 675
  workflow UI 662

G
general application properties 719
general utility functions 661
generated files 654, 655
generateODataHttpClient
  method 101
genericCallbackFunction
  method 78
get
  method 102
getAllMessages
  method 175
getAppByID
  method 177
getAppIconUrl
  method 177
getAddressConnectionID
  method 178
getBuiltInIconUrl
  method 179
getCallbackFromNativeError
  method 180
ggetClientVariables
  method 180
getCodeFromNativeError
  method 181
gGetCurrentApp
  method 182
gGetCurrentLocale
  method 182
gGetCurrentMessageValueCollection() 664
getCustomIconUrl
  method 183
dataMessage() 664
default
  method 159
gGetDstOffsetAtGivenTimeInMinutes
  method 183
ggetExternalResource
  method 184
ggetInstalledApps
  method 185
ggetItem
  method 118
ggetLocalizedDateTime
  method 186
ggetLocalizedTime
  method 187
ggetLogEntries
  method 91, 187
ggetLogEntriesErrorCallback
  method 78
ggetLogEntriesSuccessCallback
  method 78
ggetLoggingAlertDialog
  method 188
ggetLoggingCurrentLevel
  method 188
ggetMessageByID
  method 189
ggetMsgIconUrl
  method 189
ggetNativeMessageFromNativeError
  method 190
goffsetFromUTC
  method 191
ggetErrorMessageFromNativeError
  method 191
ggetPicture 666
  method 192
ggetPublicCertificate
  method 159
ggetQueryVariable
  method 193
ggetServerInitiatedApps
  method 193
ggetSharedStorageKey
  method 194
ggetSignedCertificate
  method 160
ggetSignedCertificateFromAfaria
  method 161
ggetSignedCertificateFromFile
  method 161
ggetSignedCertificateFromServer
  method 162
getTimezoneId
  method 194
getTransformData
  method 195
getUrl
  method 134
getUrlParamFromNativeError
  method 195
getUsesDST
  method 195
getXMLHTTPRequest
  method 196, 250
guid
  method 196, 250

H
hard coded credentials 724
hideProgressDialog
  method 197
HTML format 607
HttpsConnection
  namespace 97
hwc
  namespace 106
hwc-api.js
  source file 294
hwc-comms.js
  source file 432
hwc-utils.js
  source file 490
HWC.xcodeproj 828
Hybrid App
  prepackaged, BlackBerry 858
Hybrid App client
  using credentials 689
Hybrid App clients
  and static SSO2 tokens 692
  and static X.509 certificates 690
  using credentials in 688
  using SSO2 tokens in 691
Hybrid App native device functions 661
Hybrid App package
  generated files 654
Hybrid App Package Generation Wizard 631
Hybrid App packages
  assigning device users 722
  configuring notification mailbox 721
  deploying 629
Hybrid App UI functions 661
Hybrid App utility functions 661
Hybrid App validation functions 661
Hybrid Web Container
  Android 772
  ANDROID_CUSTOMIZATION_POINT_DE
    FAULTSETTINGS 772
    architecture 3
    building using source code 732
    customization 3, 772
    default values for settings screen 772
    installing from App Store 741
    management 3
    offline capabilities 3
    removing 758
    settings screen 772
    settings screen, default values 772
HybridApp
  method 197
hybridapp_Custom.html 655
hybridapp_Custom.xml 628
hybridapp_jQM.html 655
hybridapp_JQM.xml 628
hybridapp.html 656
hybridapp.html generated file 655
HybridApp.js 24
HybridWebContainer.cod 739, 740

I
image
  limitations in Hybrid App messages 604
IN_PROGRESS
  member 143
INSTALLATION_BEGIN
  member 132
INSTALLATION_END
  member 132
installing 741
internationalization
  Hybrid App Designer 707
  on the device 709
INVALID_COMMON_NAME
  member 124
InvokeOnClient 608
iOS 740
iOS Hybrid Web Container
  client certificate challenge 843
  customizations 830
  settings screen 837
iOS Hybrid Web Container customization 833
  filtering 838
  setting HTTP headers 841
  sorting 838
iOS push notification properties 757
IOS_CUSTOMIZATION_POINT 828
iPad
  hiding the listview 842
isAndroid
  method 201
isAndroid3
  method 201
isBlackBerry
  method 201
isBlackBerry5
  method 202
isBlackBerry5WithTouchScreen
  method 202
isBlackBerry6NonTouchScreen
  method 202
isBlackBerry7
  method 203
isClosed
  method 203
isDstActiveAtGivenTime
  method 204
isEnabled
  method 141
isIOS
  method 204
isIOS4
  method 205
isIOS5
  method 205
isIOS6
  method 205
isIOS7
  method 206
isIPad
  method 206
ISO-8859-1 encoding 702
isSharedStorageEnabled
  method 206
isTraceEnabled
  method 118
isTraceEnabled
  method 119
listAvailableCertificatesFromFileSystem
  method 163
load arguments 635
loadSettings
  method 207
locale
  editing 706
  properties file 703, 706
  localization 653, 700
    creating a new locale 703
    Hybrid App package 701
    limitations 701
    task flow 701
    updating the current locale 706
log
  method 208
LogEntry
  method 92, 209
logListener
  method 79
LogListener
  method 79
logToWorkflow
  method 250
look and feel 628
look and feel files 656
manifest.xml 608
manage 717
ITEM_NOT_FOUND
  member 165
iTunes 742
jquery.mobile-1.1.0.css 656
key
  method 118
length
  method 119
locale
  editing 706
  properties file 703, 706
localization 653, 700
  creating a new locale 703
  Hybrid App package 701
  limitations 701
  task flow 701
  updating the current locale 706
log
  method 208
LogEntry
  method 92, 209
logListener
  method 79
LogListener
  method 79
logToWorkflow
  method 250
look and feel 628
look and feel files 656
manifest.xml 608
Developer Guide: Hybrid Apps
Index

showUrlInBrowser 245, 253
shutdown 246
startClient 246
startInteraction 141
startInterval 141
startLogListener 92
startOrStopLogListenerErrorCallback 83
startOrStopLogListenerSuccessCallback 84
startTrace 131
stopInteraction 142
stopInterval 142
stopLogListener 95
stopTrace 131
stringify 215
this.containsName 165
this.getAllVariableNames 166
this.getClientVariables 198
this.getCount 166
this.getCustomIconList 198
this.getDate 210
this.getDefaultCustomIcon 199
this.getDisplayImage 199
this.getMessage 211
this.getMessageId 216
this.getModuleName 217
this.getModuleId 200
this.getModuleVersion 217
this.getName 173
this.getPriority 217
this.getProcessedImagePath 174
this.getMessage 218
this.getMessageId 219
this.getSubject 179
this.getType 174
this.getVariableValueByName 167
this.getVersion 167
this.getWidth 174
this.isProcessed 219
this.isRead 219
this.updateProcessed 220
this.updateRead 220
updateMessageProcessed 248
updateMessageRead 248
uploadTrace 132
Microsoft ActiveSync, installing and configuring 743
ModuleDesc 608
ModuleName 608
ModuleVersion 608
MSG_ADDED
member 135
MSG_PRIORITY_HIGH
member 135
MSG_PRIORITY_NORMAL
member 136
MSG_REFRESH
member 136
MSG_REMOVED
member 136
MSG_UPDATED
member 137

N

namespace
anonymous 68
AppLog 84
HttpsConnection 97
hwc 106
NativeErrorCodes 121
native device functions 663
NativeErrorCodes
namespace 121
NAVIGATION_ERROR
member 124
network edge authentication 688
NO_ERROR
member 143
non HTTP authentication request 895
non-ASCII encoding 702
NOT_SUPPORTED
member 143
notification mailbox 721
NOTIFICATION_CANCEL
member 137
NOTIFICATION_CONTINUE
member 137
notifications
creating 39
null value support 600
object queries
   binding to a menu item 646
object query parameters
   defining a control that passes 647
OData 43
offline capabilities 3
onGetPictureError
   method 81
onGetPictureSuccess
   method 82
OPEN_APP_NOT_EXIST
   member 138
OPEN_APP_OTHER
   member 138
OPEN_APP_SUCCESS
   member 138
OPEN_MSG_APP_NOT_EXIST
   member 139
OPEN_MSG_NOT_EXIST
   member 139
OPEN_MSG_OTHER
   member 139
OPEN_MSG_SUCCESS
   member 140
openApp
   method 221
openMessage
   method 222
Optimize for appearance look and feel 655
Optimize for performance look and feel 655, 659
options
   member 72
OTA 739
over the air 739

P
perf
   member 140
performance agent 725, 726
PersistAppDomain 608
PersistentContent 696
PersistentContentListener 696
PersistentStore 696
PhoneGap 864
   initializing 886
   supported APIs 864
Phonegap HTTPS proxy connection properties 888
PhoneGap plugin
   testing 878
PictureError
   member 142
PictureOptions
   member 72
PIN screens
   CreatePasswordViewController.xib 833
   customizing 833
   EnterPasswordViewController.xib 833
   iOS 833
PlatformIdentification.js
   source file 500
Plugins/AppLog/applog.js
   source file 513
Plugins/HttpsProxy/datajs-https-proxy.js
   source file 532
Plugins/HttpsProxy/https-proxy.js
   source file 538
Policy
   member 133
preferences
   appearance 702
   content types 702
   general 702
processDataMessage
   method 251
ProcessUpdates 608
processWorkflowMessage
   method 252
properties
   push notification for iOS 757
propogate to attributes 635
proxy connection 44
PurchaseOrderSample 703
push notification properties for iOS 757
Q
query types
   addAllmenuitems 602
   addMenuItem 602
   alert 602
   clearrequestcache 602
   clearrequestcachefile 602
   close 602
   downloadattachment 602
   formredirect 602
   loadtransformdata 602
   logtoworkflow 602
removeAllMenuItems 602
rmi 602
setScreenTitle 602
showAttachment 602
showCertPicker 602
showInBrowser 602
showLocalAttachment 602
submit 602

R
real-time lookup pattern
data flow diagram 634
overview 634
REG_ERR_AUTO_REG_NOT_ENABLED
member 144
REG_ERR_AUTO_REG_TEMPLATE_NOT_F O UND
member 145
REG_ERR_AUTO_REG_USER_NAME_TOO_L ONG
member 145
REG_ERR_AUTO_REG_WRONG_USER_FOR_ DEVICE
member 145
REG_ERR_COULD_NOT_REACH_MMS_SER VER
member 146
REG_ERR_INVALID_USER_NAME
member 146
REG_ERR_MMS_AUTHENTICATION_FAILED
member 146
REGISTRATION_METHOD_AFARIA
member 147
REGISTRATION_METHOD_AUTOMATIC
member 147
REGISTRATION_METHOD_CERTIFICATE
member 147
REGISTRATION_METHOD_MANUAL
member 148
REGISTRATION_METHOD_NO_PREFERENCES
member 148
removeAllMenuItems
method 223
removeAppInstallationListener
method 223
removeAppListener
method 224
removeConnectionListener
method 226
removeItem
method 119
removeLogListener
method 227
removeMessage
method 229
removeMessageListener
method 229
removePushNotificationListener
method 231
REQUIRED_PARAMETER_NOT_AVAILABLE
member 124
RequiresActivation 608
resource functions 675
RESPONSE_TOO_LARGE
member 125
rmi.xml 712
RSA algorithm 751

S
sample_AppListener
method 232
sample_ConnectionListener
method 233
sample_InstallationAppListener
method 234
sample_LogListener
method 234
sample_MessageListener
method 235
sample_PushNotificationListener
method 235
SAP passport 725
saveLoginCertificate
method 236, 252
saveLoginCredentials
method 236, 252
saveSettings
method 237
send a notification 711
sending server notification to a device 643
sendRequest
method 104
sendRequestErrorCB
method 82
sendRequestErrorCBParameter
class 70
sendRequestSuccessCB
method 83
SupPassword 724
for context variables 723
SUPStorage 677
class 116
SUPStorage.js 676
source file 560
SUPStorageException
class 120
SupUser 724, 725
for context variables 723
synchronization software 743

T

task flow 7
testing
X.509 certificates 751
this.containsName
method 165
this.getAllVariableNames
method 166
this.getClientVariables
method 198
this.getCount
method 166
this.getCustomIconList
method 198
this.getDate
method 210
this.getDefaultCustomIcon
method 199
this.getDisplayName
method 199
this.getEventID
method 210
this.getHeight
method 173
this.getIconIndex
method 200, 216
this.getIconUrl
method 200, 216
this.getImagePath
method 247
this.getMessage
method 211
this.getMessageId
method 216
this.getModuleId
method 217
this.getModuleID
  method 200
this.getModuleVersion
  method 217
this.getName
  method 173
this.getPriority
  method 217
this.getProcessedImagePath
  method 174
this.getReceivedDate
  method 218
this.getSender
  method 218
this.getSubject
  method 219
this.getType
  method 174
this.getVariableValueByName
  method 167
this.getVersion
  method 167, 200
this.getWidth
  method 174
this.isProcessed
  method 219
this.isRead
  method 219
this.updateProcessed
  method 220
this.updateRead
  method 220
Timezone.js
  source file 573
TOO_LARGE
  member 143
touch point 828
trace 725
U
UNAVAILABLE
  member 144
UNKNOWN_ERROR
  member 125
UNKNOWN_MIME_TYPE
  member 125
UNSUPPORTED_ATTACHMENT_TYPE
  member 126
updateMessageProcessed
  method 248
updateMessageRead
  method 248
upgrading the PhoneGap library 797, 827
uploadTrace
  method 132
URL parameters 42
USER_REJECT
  member 144
UTF-8 encoding 702
V
variables, context
  configuring 723
viewing Hybrid App messages
  Android 711
  BlackBerry 711
  iOS 711
  Windows Mobile 711
W
whitelisting 44
WorkflowClient.xml 39, 617
X
X.509 certificate 754