SYBASE
An SAP Company

Developer Guide: Mobile Workflow Packages

Sybase Unwired Platform 2.1 ESD #3

DOCUMENT ID: DC01218-01-0213-04
LAST REVISED: January 2013

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

Contents

Introduction to Developer Guide for Mobile Workflow Packages

This developer guide provides information about using Sybase® Unwired Platform features to create Mobile Workflow packages. The audience is Mobile Workflow developers.

This guide describes requirements for developing a Mobile Workflow package, how to generate Mobile Workflow package code, and how to deploy the Mobile Workflow package to the device or simulator.

Companion guides include:

- *Sybase Unwired WorkSpace – Mobile Business Object Development*
- *Sybase Unwired WorkSpace – Mobile Workflow Package Development*
- *Tutorial: Mobile Workflow Package Development*
- *Troubleshooting for Sybase Unwired Platform*

Documentation Roadmap for Unwired Platform

Sybase® Unwired 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. *Fundamentals* is available on the Sybase Product Documentation Web site.

Check the Sybase Product Documentation Web site regularly for updates: access *http://sybooks.sybase.com/nav/summary.do?prod=1289*, then navigate to the most current version.

Introduction to Developing Mobile Workflow Applications With Sybase Unwired Platform

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

Within Sybase Unwired Platform, development tools enable both aspects of Mobile Workflow application development:

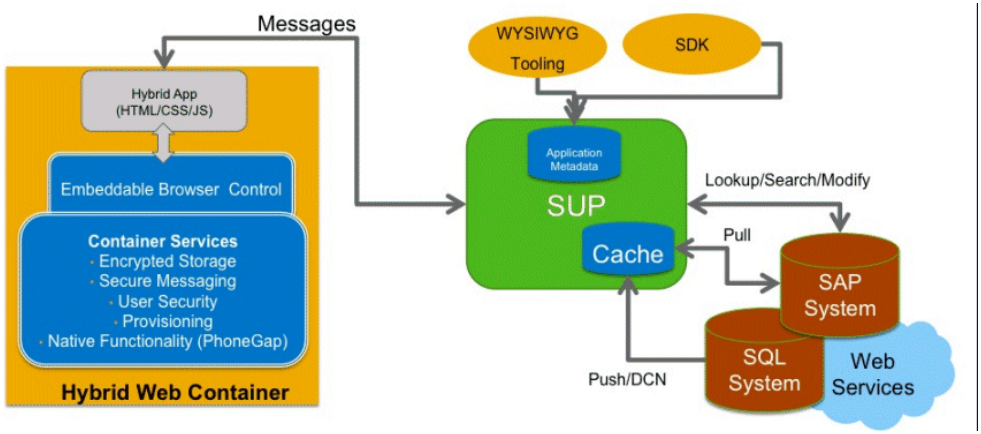
- The data aspects of the Mobile Workflow application are called mobile business objects (MBO), and "MBO development" refers to defining object data models with back-end enterprise information system (EIS) connections, attributes, operations, and relationships. Mobile Workflow applications can reference one or more MBOs and can include load parameters, personalization, and error handling.
- Once you have developed MBOs and deployed them to Unwired Server, develop device-resident presentation and logic for your Mobile Workflow application using the Mobile Workflow Forms Editor.

Note: See *Sybase Unwired WorkSpace – Mobile Business Object Development* for procedures and information about creating and deploying MBOs.

Hybrid Web Container Architecture

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

The Hybrid Web Container is a native application that 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. The Hybrid Web Container enables the rapid development of mobile workflows, in which you can extend existing enterprise business processes, to a mobile device so that business process decisions can be made on a mobile device.



Mobile Workflow Forms Editor

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

Mobile Workflow package files are generated using the Mobile Workflow Package generation wizard in the Mobile Workflow Forms editor. The generated Mobile Workflow 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 Mobile Workflow 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.

Mobile Workflow packages can be deployed to Unwired Server and assigned to users using the Mobile Workflow Forms Editor in Eclipse.

Customization

You can modify certain files in the generated Mobile Workflow package to customize application behavior.

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

- HTML – HTML files are generated in the Mobile Workflow Forms editor. The files that are generated 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 Mobile Workflow deployment tool. The Mobile Workflow Forms editor 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 `customBeforeWorkflowLoad()`. Additional custom JavaScript files can be added to the Mobile Workflow package in the Eclipse WorkSpace.
- CSS – the Hybrid Web Container uses a 3rd-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 Mobile Workflow Packages in Eclipse and manage them through the Sybase Control Center console. No device interaction is required from the administrator. Once a Mobile Workflow package is deployed into an existing installation, the administrator can configure the Mobile Workflow package and assign it to any active user in the system.

Offline Capabilities

Server-initiated notifications extract data from the backend and Unwired 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 Workflow 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.

Hybrid Web Container Development Task Flow

Developing a Hybrid Web Container includes these basic tasks.

1. Open or import a mobile application project with predefined mobile business objects (MBOs).
2. Deploy the Mobile Application Project:
   b. On the Target Server page, select the server and connect to it.

c. On the Server Connection Mapping page, map the database connection profile to the server.

3. Create the application connection in Sybase Control Center (SCC).

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

4. Use the Mobile Workflow Forms Editor to create a new Hybrid Web application.

Note: Optionally, you can create a Hybrid Web application manually, however, using the Mobile Workflow Forms Editor, automates many tasks and provides integration across different device platforms.

5. Use the Mobile Workflow Forms Editor 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. Use the Mobile Workflow Package Generation wizard to generate the Hybrid Web Container files.
8. (Optional) Customize the generated `Custom.js` file.
9. (Optional) If you customized the Hybrid Web application files, re-generate the files using the Mobile Workflow Package Generation Wizard.
10. Deploy the Mobile Workflow package to Unwired Server.
11. Install and configure the Hybrid Web Container on the device or simulator.
12. In SCC, assign the Hybrid Web application to the device user.
13. On the device or simulator, run, test and debug the Hybrid Web application.

Note: See *Sybase Unwired WorkSpace – Mobile Business Object Development* for procedures and information about creating and deploying MBOs.

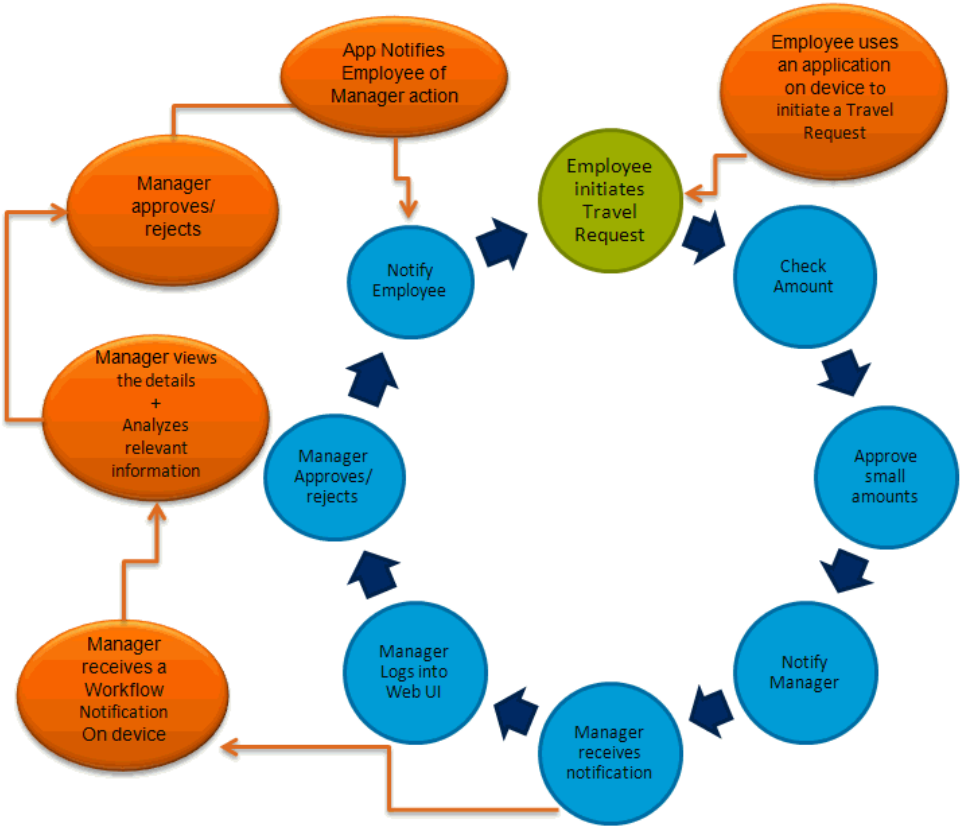
Identify a Business Process for Workflow Development

The first step is identifying whether a workflow package can implement a decision point in a particular business process.

Workflow packages 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 Workflow package allows creating a mobile business object (MBO) and sending it to the Unwired Server, or retrieving an MBO from the Unwired Server and displaying that information in a decision step. A more complex Workflow package 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 mobile workflow 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.

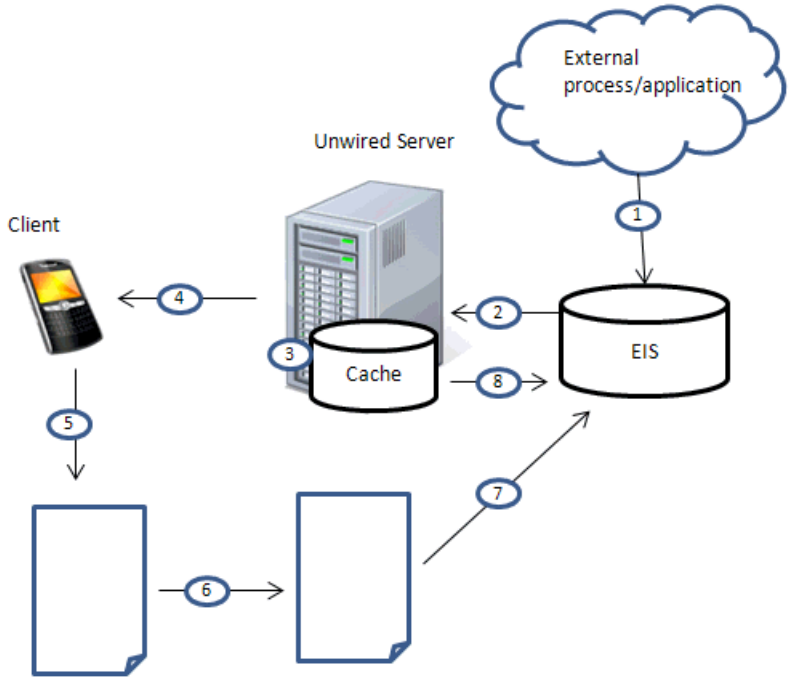


Hybrid Web Container Patterns

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

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

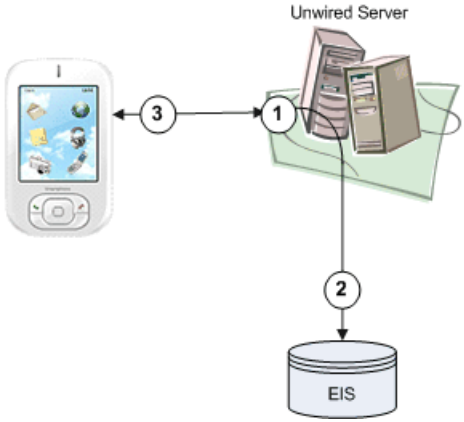
These patterns are not mutually exclusive. You can create applications that combine patterns in various ways to meet business needs. For example:



1. An external process or application updates EIS data.
2. The changed data triggers a data change notification (DCN), which is sent to Unwired Server, or a message from another workflow client updates mobile business object (MBO) data contained on Unwired Server.
3. The DCN could be programmed to update MBO data.
4. Unwired 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 Unwired Server.
8. Unwired Server updates the EIS.

Online Lookup

This pattern provides direct interaction between the data requester (workflow client) and the enterprise information system (EIS), supplying real-time EIS data rather than 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 workflow 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, Unwired 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 workflow client is triggered when the user selects the **Submit** menu item that calls the findByParameter object query.

Implementing Online Lookup for Workflow Clients

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 workflow application that calls the findByParameter object query to return real-time results from the EIS.

Defining Mobile Workflow 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 Unwired WorkSpace, create an MBO that has at least one load argument. For example, you could define an Emp MBO as:

```
SELECT   id,
  empName,
```

```
  empDeptId FROM sampledb.dba.employee
WHERE empDeptId = :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 Mobile Workflow 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 Unwired 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.

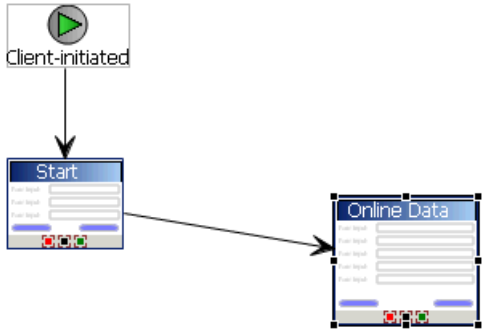
Prerequisites

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

Task

1. From Unwired WorkSpace, launch the Mobile Workflow Forms Editor.
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**.

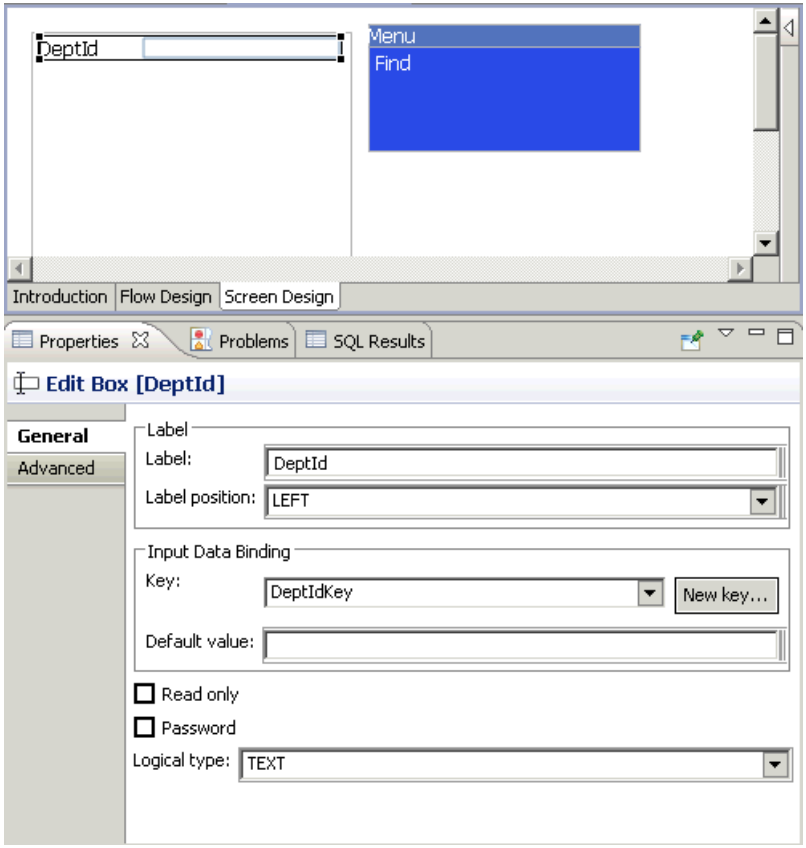
      If you select the Parameter Mappings tab, you see all the load parameters defined for the MBO and used to generate the findByParameter object query. In addition to Key, you can map parameters to BackEndPassword, BackEndUser, DeviceId, DeviceName, DeviceType, UserName, MessageId, ModuleName, ModuleVersion,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

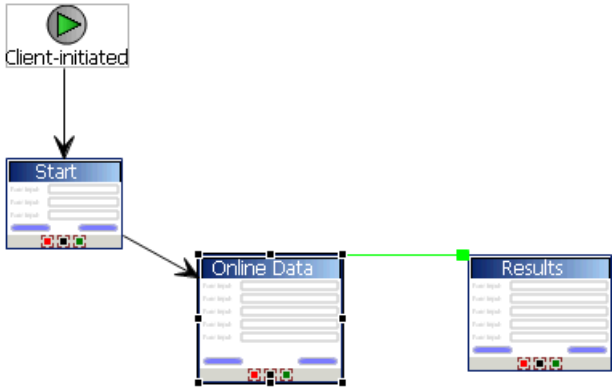
Add a control to pass the load argument to Unwired Server. Define a screen that displays the results returned from the EIS.

1. Define a control that passes the load argument to Unwired 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.



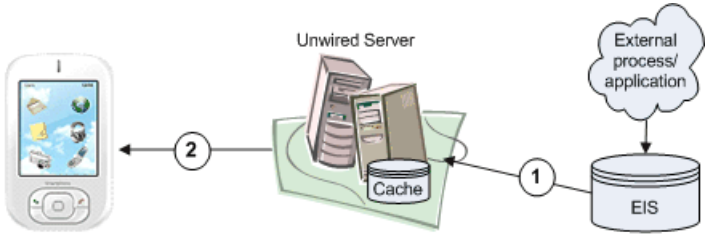
4. Configure the Results screen to display the results. In this example, the Emp MBO, contains three attributes: Id, empName, and empDeptId. Create a Listview with a cell for each attribute to display the results returned from the EIS as a list:
   a) From the Flow Design window, double-click the **Results** 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 workflow 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 Unwired Server. Any data changes matching these rules trigger a notification from Unwired Server to the workflow 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/mobile workflow package, Unwired Server sends a notification to the client.

Implementing Server Notification for Workflow Clients

Set up Unwired Server to send notifications to workflow clients when matching rules are encountered.

Defining the Mobile Business Object for Server Notification

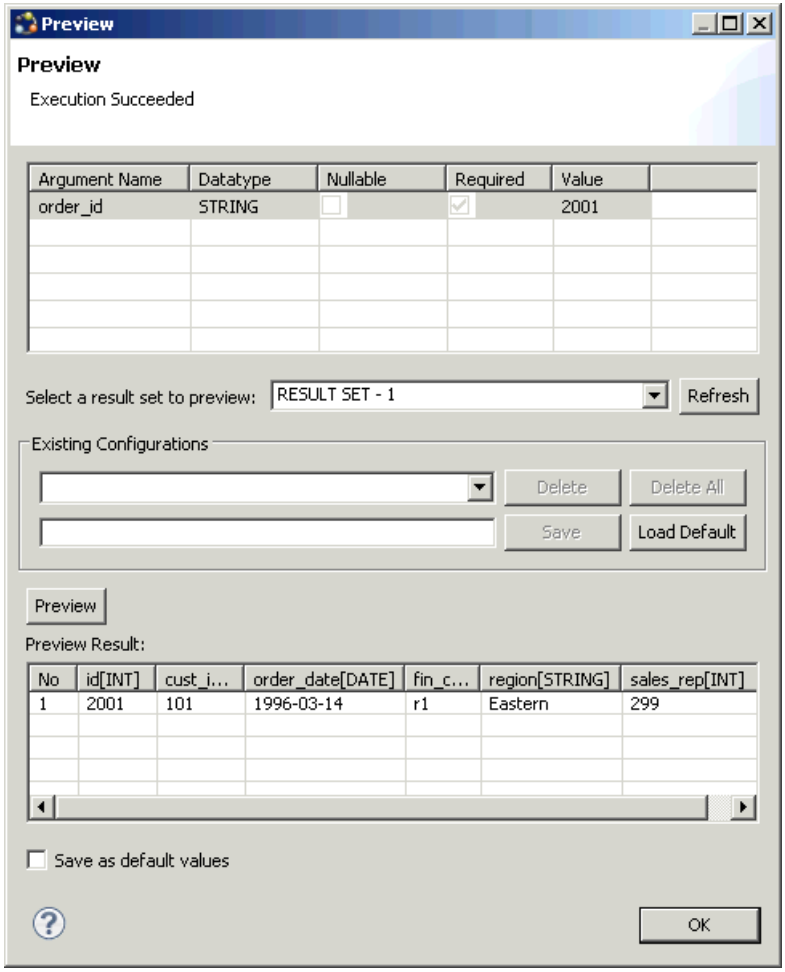
The server notification pattern supports any number of MBO definitions. For this example, create an MBO with one load argument, assign the load argument a propagate-to attribute value, then assign the MBO to a cache group that uses an Online policy.

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

1. In Unwired 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:

```
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 Unwired Server.

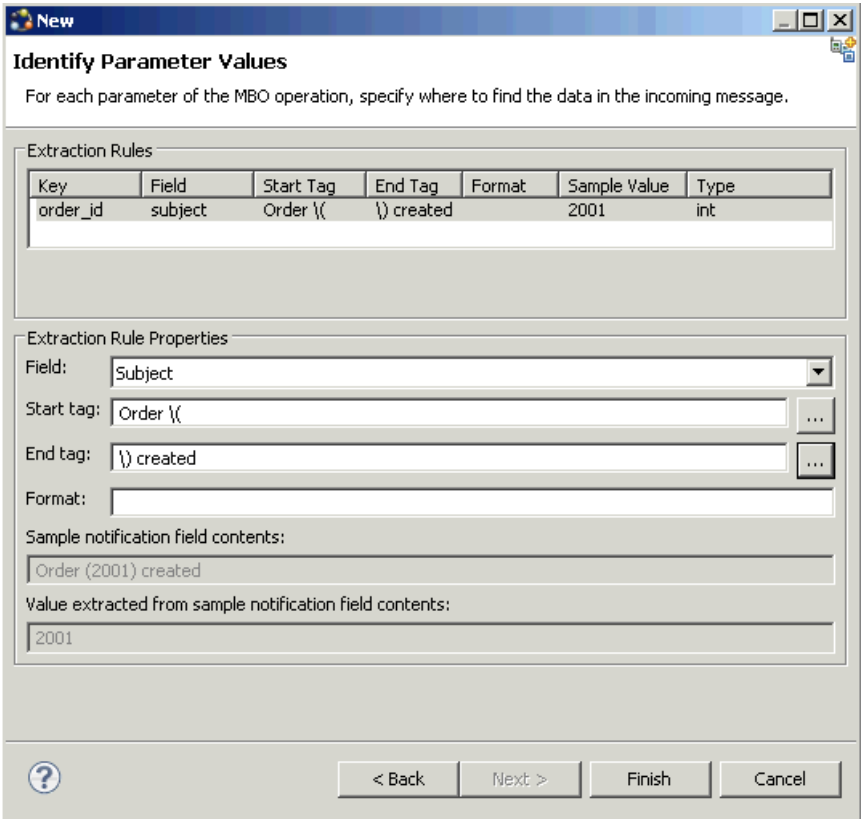
Creating the Server-Driven Notification Starting Point

Create a new workflow application with a server-initiated starting point.

1. From Unwired WorkSpace, select **File** > **New** > **Mobile Workflow Forms Editor**.
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.

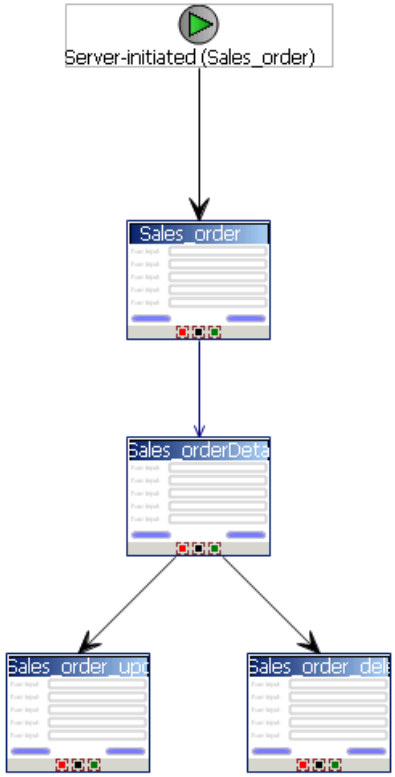
      The order_id parameter appears in the Parameters field. Click **Next**.
   d) Specify a sample notification. Enter `Order (2001) created` in the Subject line. Click **Next**.
   e) Click and drag to select "`Order (`", while this phrase is highlighted, right-click and select **Select as Matching Rule**:
   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 workflow 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 workflow package to Unwired Server.

Sending an Order Notification to the Device

Use the mobile workflow "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 a device user and assign it to the workflow package in Sybase Control Center (SCC).
2. Download and configure the Sybase messaging client on the device or emulator to match those performed in SCC.

   See your Sybase documentation for details.

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.

1. In the Flow Design of the Mobile Workflow Forms Editor, 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 SCC and assigned to the workflow package.
3. In the Subject field, enter a sales order that meets the matching rules criteria defined for the Sales_order workflow application. For example:

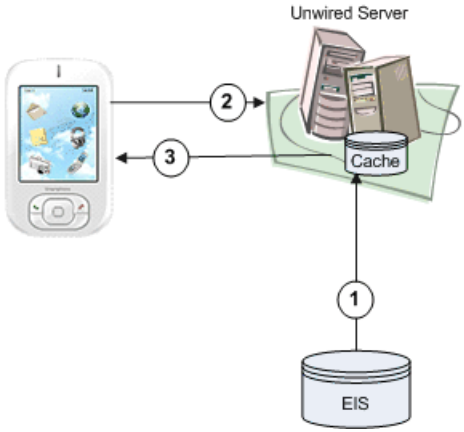
   ```
   Order (2001) created
   ```

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 workflow client 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 workflow application 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 Unwired Server cache (also called the CDB) to the client.

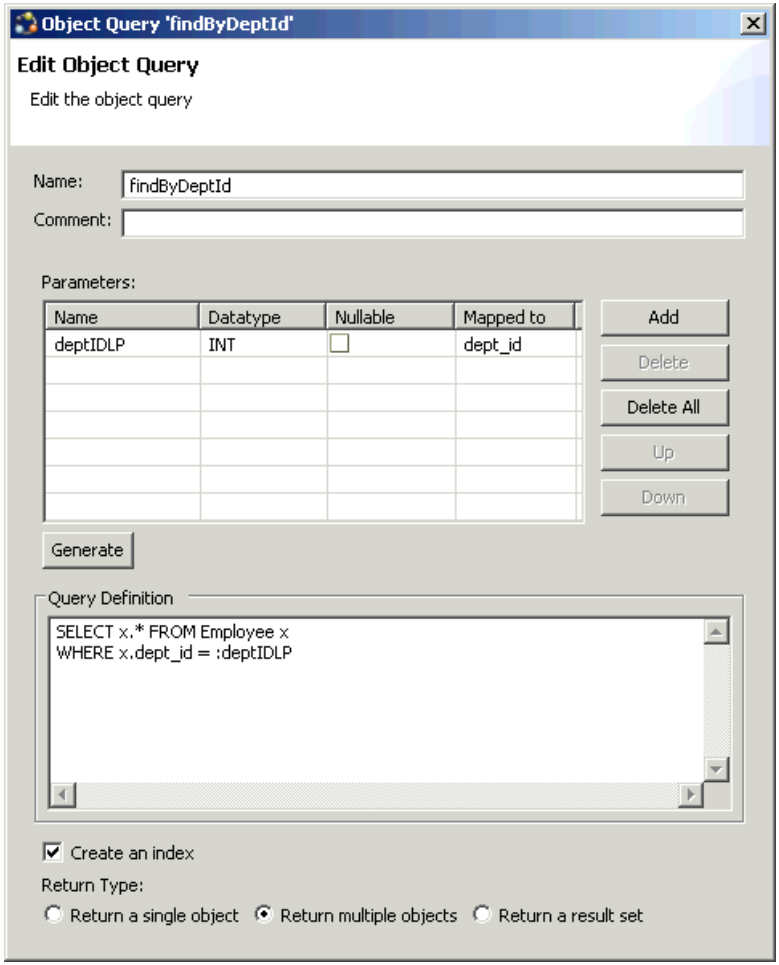
This example defines an MBO that retrieves employee benefit information for all employees of a given department based on the dept_id attribute using the findByDeptId object query.

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

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

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

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



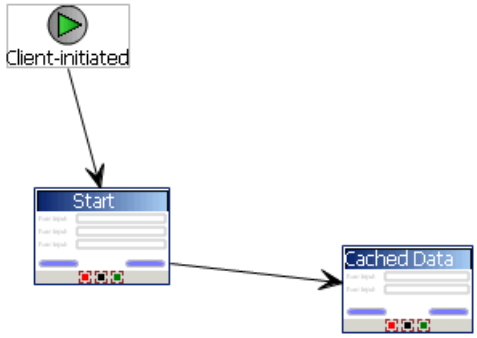
4. Deploy the project that contains the MBO to Unwired Server.

Binding the findByDeptId Object Query to a Menu Action

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

1. From Unwired WorkSpace, launch the Mobile Workflow Forms Editor.
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.



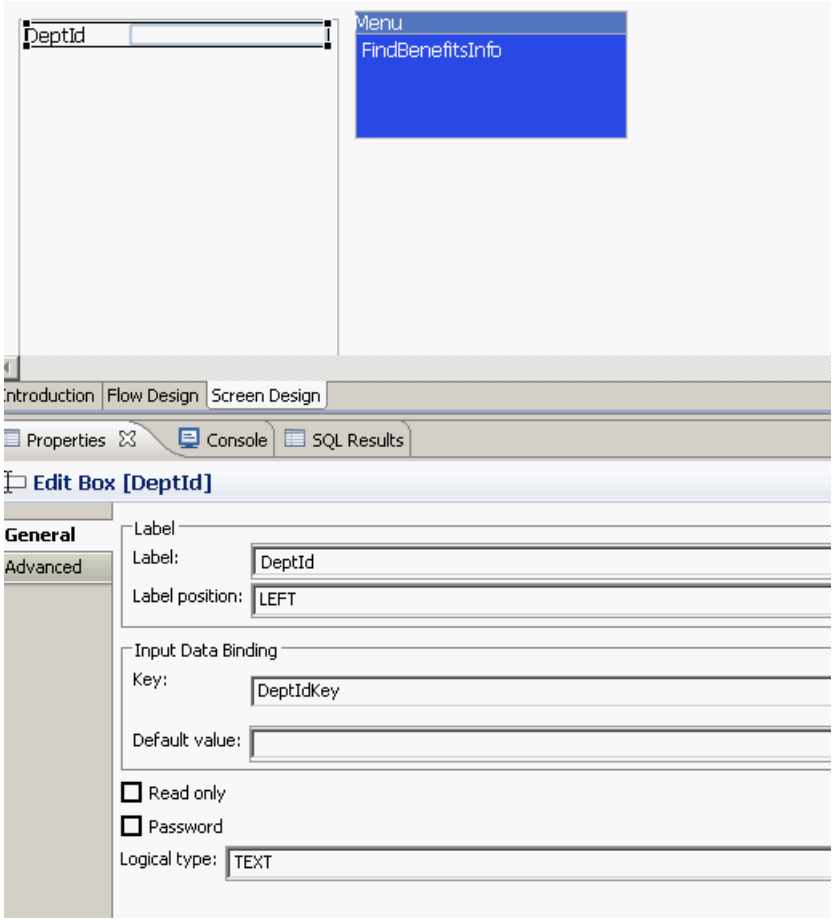
3. Highlight the menu item you want to map, or create a new menu item.
4. Define a Submit action named FindBenefitsInfo that invokes the findByDeptId object query:
   a) In the Properties view, in the General properties for the selected menu item, select **Online Request** as the Type.
   b) In the Details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

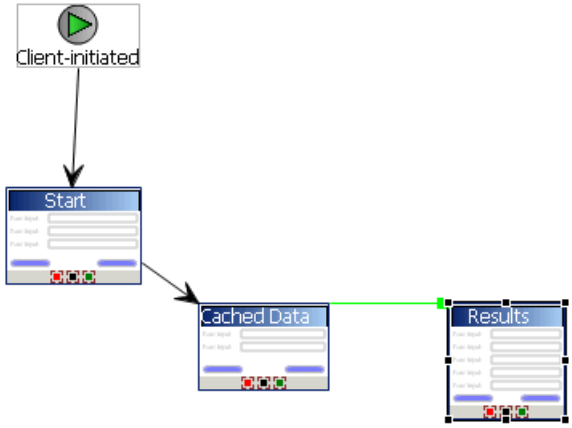
Defining the Control that Contains the findByDeptId Object Query Parameter

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

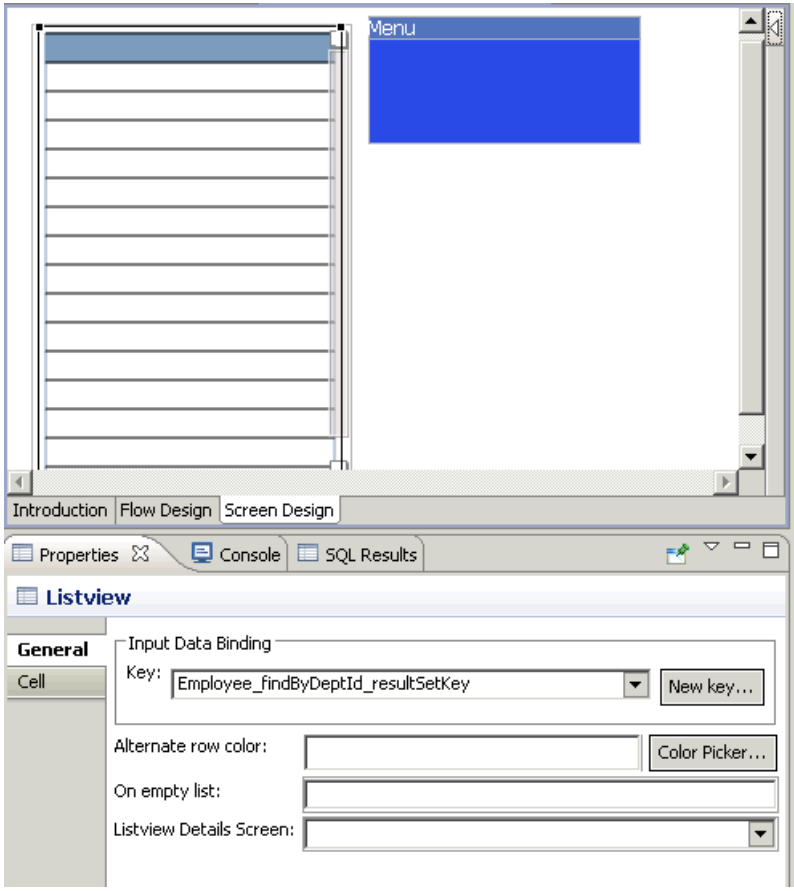
1. Define a control that passes the object query parameter to Unwired Server from the screen (named Cached Data) that contains the menu item (named FindBenefitsInfo) that invokes the findByDeptId object query:
   a) Select an **EditBox** control and click in the control area.
   b) Name the EditBox `DeptId`.
   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 Unwired 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).

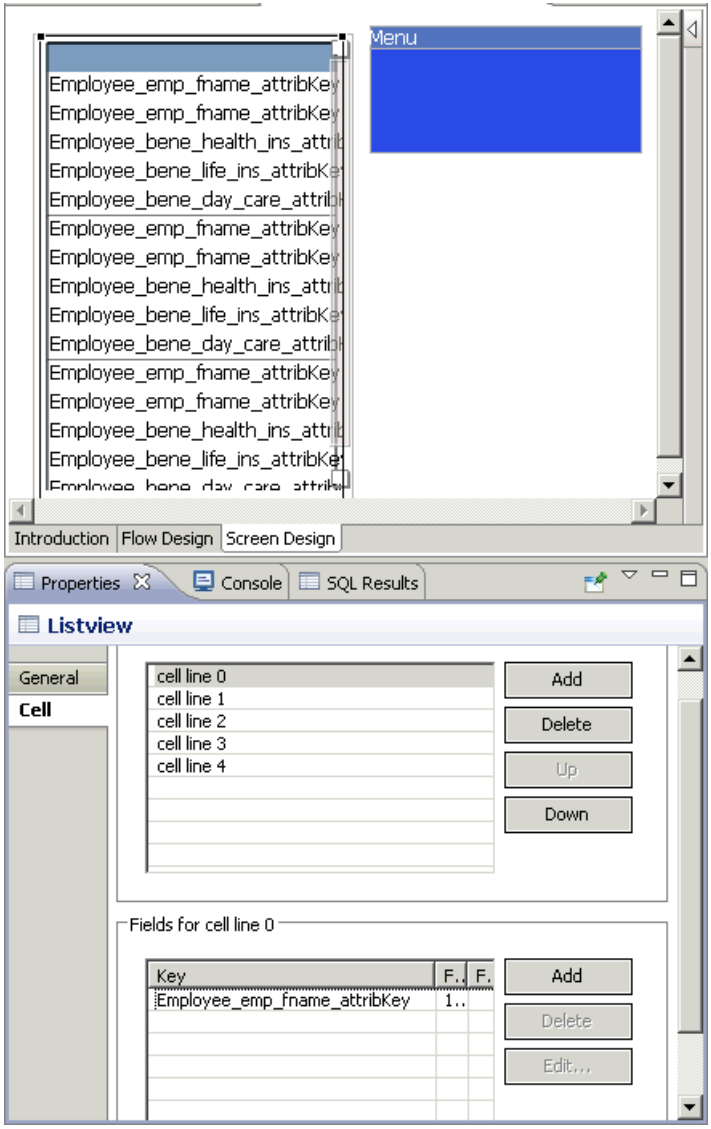


c) Select the **Cell** tab, then click **Add** to add cell line 0.

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 workflow 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 Mobile Workflow 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 Mobile Workflow Forms Editor from Unwired WorkSpace and create a new Mobile Workflow form:
   a) Select **File** > **New** > **Mobile Workflow Forms Editor**.
   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 Mobile Workflow Forms Editor, map the personalization keys to the Mobile Workflow 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 Unwired Server cache (CDB) and returned to the client.

Mobile Workflow Application Configuration for Data Change Notification

This section contains details about developing workflow applications that take advantage of DCN updates.

Mobile workflow applications require a server-initiated starting point and defined matching rules, which allows Unwired Server to push changes to workflow application clients. See the topics *Starting Points* and *Adding Matching Rules*.

Extending Data Change Notification to Mobile Workflow Clients

Mobile Workflow data change notification (WF-DCN) requests allow Unwired 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 workflow application receives notification, it can retrieve data directly from the EIS or from the Unwired Server cache, keeping the application synchronized. DCN messages targeted for MBOs used in workflow 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 workflow 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 workflow server-initiated starting point, 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: workflow engine convert MBO data and WF-DCN message into workflow email, and push it to device mobile inbox

1. Operation name(op) **:upsert** or **:delete**– same as regular DCN.
2. Message ID (id) of the Mobile Workflow – used for correlation (a **:delete** for a previously submitted request with **:upsert** is possible)
3. Username (to) – the Sybase Unwired 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 Sybase Control Center.

   Note: The "To" field must match the Unwired Platform user name—not the user name used to register the device.
4. Subject (subject) – subject of the workflow message.
5. Originator <from> – who the workflow message is from.
6. Body of the workflow message <body> – it can embed customized information.

7. <received> – received time of the Mobile Workflow message.
8. <read> – whether the Mobile Workflow message is read.
9. <priority> – whether the Mobile Workflow message has a high priority.
10. List of dcn request <data> – JSON format string.

Example DCN request in JSON format:

```
{
"op":":upsert",
"id":"WID123",
"to":"SUPAdmin",
"subject":"Trip request approval required",
"from":"user321",
 "body":"This is a message just used to do a test",
"received":"2009-03-29T10:07:45+05:00",
"read":false,
"priority":true,
"data":
   [
     {"id": "1",
     <general dcn request>
      }
      …
     {"id": "4",
     < general dcn request>
      }
   ]
}
```

Mobile Workflow 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, Unwired Server gets the **wf** cmd value from the request first, and:

1. Unwired Server invokes preProcessFilter if the DCN filter is specified.
2. Unwired 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 Mobile Workflow 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 workflow package in the “to” attribute of the Mobile Workflow message, then matches them with the receiver list.

   For every receiver, Unwired Server generates multiple workflow messages (all workflow 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. Unwired Server invokes the postProcessFilter, if specified.
8. If no errors occur, Unwired 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, Unwired Server returns an HTTP 500 error with the body of the JSON log records.

Non HTTP Authentication Workflow DCN Request

You can send Mobile Workflow DCN requests that are not authenticated.

The URL is:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=admin&domain=default&username=supAdmin&password=sup
Pwd&dcn_filter=aa.bb&dcn_request=<wfrequestdata>
```

where *supAdmin* represents the Unwired Server Administrator, and *supPwd* represents the Administrator's password defined during Unwired Platform installation.

Sending Workflow DCN to Users Regardless of Individual Security Configurations

You can send Mobile Workflow DCN requests to users in other security configurations if you belong to the default security configuration.

If the workflow DCN sender is authenticated against the default admin security configuration, they are automatically authorized to push data to all users regardless of their individual security configuration. If not, the sender can only push to users within the same security configuration.

For example, in the case of a non HTTP authentication request, this request is authorized to push data to users in other security configurations since the sender *supAdmin*, belongs to the admin security configuration:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=othersecurity&domain=default
&username=supAdmin@admin&password=supPwd&dcn_filter=aa.bb&dcn_reque
st=<request>
```

And this request is denied because *supAdmin*@mysecurityconfig can only push data to users in the same security configuration:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=othersecurity&domain=default
&username=supAdmin@mysecurityconfig&password=supPwd&dcn_filter=aa.b
b&dcn_request=<request>
```

Mobile Workflow DCN Request Response

After processing of the Mobile Workflow DCN request, Unwired 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 Mobile Workflow request.
2. The result of processing the general DCN requests.

The response is also in a JSON format string:

```
{
<wf dcn result>
"result":
  [
    {
     <general dcn result>
    },
    {
     <general dcn result>
    }
  ]
}
```

An example response is:

```
{
    "id":"1",
"success":false,
"statusMessage":"there is error in processing dcn",
"result":
  [
    {
    "id":"1",
    "success":true,
    "statusMessage":""
    },
    {
    "id":"2",
    "success":false,
    "statusMessage":"bad msg2"
    }
  ]
}
```

Workflow 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.

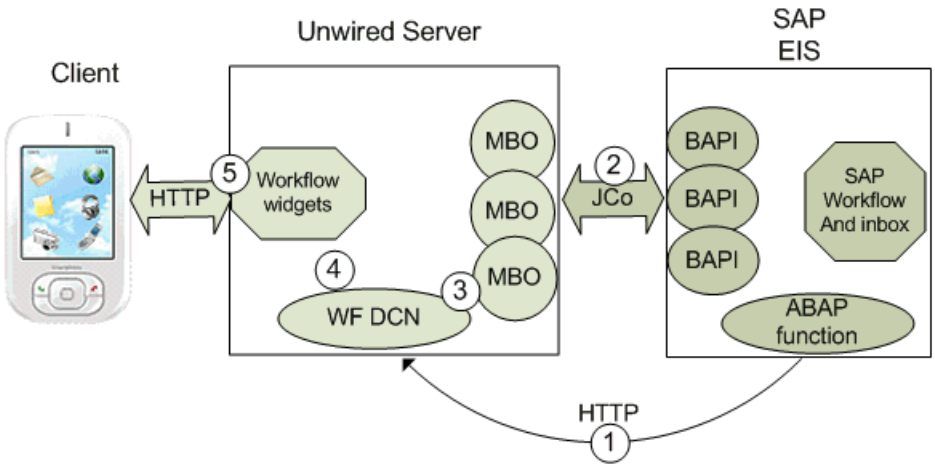
Comparing Workflow DCN With and Without Payload

This section compares the two types of WF-DCN and includes examples of each.

Mobile Workflow DCN Without Payload

Understand how to construct a workflow 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 Unwired Server, which are then delivered to the device, for example, a workflow notification.
2. After the EIS sends a workitem id to Unwired Server, Unwired Server uses workitem MBO and workitem id to retrieve details of the workitem from the EIS.
3. After Unwired Server receives the message, a matching workflow 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 workflow 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 workflow server-initiated starting point after extracting the parameter values from the WF-DCN message body.

Message format

The message format of the WF-DCN message without payload is:

```
{"id":"","op":"","subject":"","to":"","from":"","read":,"priority":
"","body":"",
"data":{}
```

For example:

```
{"id":"","op":":upsert","subject":"test","to":"test","from":"test",
"read":,
"priority":"","body":"MATCH:SUP_MWF,TaskID:TS97200149, WIID:
1470577,
USER:PERF0111*#END#*","data":{}
```

Unwired Server extracts information from the DCN message and retrieves details from the EIS.

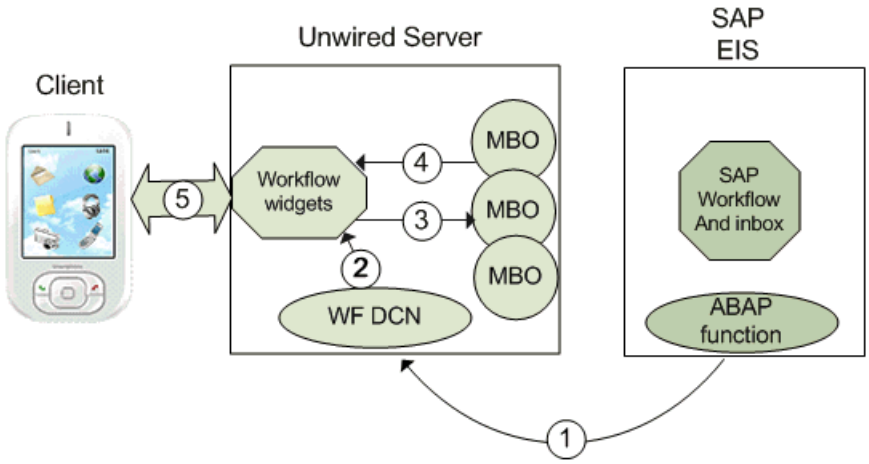
Processing the WF-DCN without payload message

After Unwired Server receives the message, a matching workflow 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.

Mobile Workflow DCN With Payload

Understand how to construct a workflow 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 Unwired 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 Unwired Server, the workflow engine evaluates the matching rule against all registered workflows. If a matching rule matches this message, the workflow server starting point for that workflow 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 workflow 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 workflow engine converts the MBO data and WF-DCN message into a workflow notification, then pushes it to the device mobile inbox using Sybase messaging (MOCA).

MBO cache group policy

The cache group policy of MBOs used in WF-DCN with payload must be DCN.

Message format

The message format of the WF-DCN message with payload is:

```
{”id”:””,”op”:””,”subject”:””,”to”:””,”from”:””,”read”:””,”priority
”:””,”body”:””,
“data”:[{”id”:””,”pkg”:”Package”,”messages”:
[{”id”:”2”,”mbo”:”MBO”,”op”:”:upsert”,
”cols”:
{”attribute1”:”value1”,”attribute2”:”value2”,”attribute3”:”value3”}
}
```

The message must contain e-mail information: subject, to, from, and so on, and include the MBO package name and version, MBO name, attribute name, and attribute value. The message can include multiple MBOs. For example:

```
{”id”:”1137”,”op”:”:upsert”,”subject”:”PERF0111’s Leave Request”,
”to”:”PERF0111”,”from”:”Leave Work
Flow”,”read”:”false”,”priority”:”true”,
”body”:”MATCH:SUP_MWF,TaskID:TS97200149, WIID:1470577,
USER:PERF0111*#END#*”,
”data”:[{”id”:”dcbtest”,”pkg”:”sup_mwf:1.2”,”messages”:
[{”id”:”2”,”mbo”:”Workitem”,
”op”:”:upsert”,”cols”:
{”WORKITEM”:”1470577”,”USERNAME”:”perf0111”,”DESCRIPTION”:”cc”,
”DECISION”:”test”}},{”id”:”6”,”mbo”:”Alternatives”,”op”:”:upsert”,
”cols”:
{”WORKITEM”:”1470577”,”USERNAME”:”perf0111”,”PKEY”:”01”,”PVALUE”:”A
p”}}]}]}
```

Sample Java Function for Generating Workflow DCN

This WF-DCN sample illustrates WF-DCN without payload.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;
import java.net.Authenticator;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.PasswordAuthentication;
import java.net.ProtocolException;
import java.net.URL;
import java.net.URLEncoder;

public class HttpAuth
{
    /**
     * @param args
     * @throws MalformedURLException
     */
    public static void main(String[] args) throws Exception
    {
        URL url = null;

        String 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#\"}";

        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);
        }

    }
}
```

Mobile Workflow Development

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

A Mobile Workflow application requires an integration module on the server side, which is implemented by a static set of logic that processes Mobile Workflow-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 generate Mobile Workflow forms that work on these platforms:

- Apple iOS
- BlackBerry
- Windows Mobile Professional
- Android

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

Develop a Mobile Workflow Application Using the Mobile Workflow Forms Editor

The Mobile Workflow Forms editor provides UI controls that make development of Hybrid Web Containers fast and easy.

For information about using the Mobile Workflow Forms editor to develop Mobile Workflow applications, see online help, *Sybase Unwired WorkSpace – Mobile Workflow Package Development*.

Deploy the Mobile Workflow Package to Unwired Server

Use the Mobile Workflow Package generation wizard to generate the Mobile Workflow package and deploy it to Unwired Server to make it available for device clients.

Generating the Files for a Mobile Workflow Package

Use the Mobile Workflow Package Generation wizard to generate the files for the mobile workflow package, optionally deploy the generated package files to the server, and assign the package to one or more users' devices.

1. Right-click in either the Flow Design or Screen Design page of the Mobile Workflow Forms Editor and select **Generate Mobile WorkflowPackage**, or click the code generation icon on the toolbar.
2. In the Mobile Workflow Package Generation wizard, enter or select:

Option	Description
Favorite configurations	(Optional) Select a configuration.
Package Generation and Deployment	
Update generated code	Generate the mobile workflow package and its files. When this option is unselected, the mobile workflow package files are not regenerated, so that modifications made to files that are normally regenerated are not overwritten. This also means, however, that changes made in the Mobile Workflow Forms Editor are not reflected in the generated files. **Note:** The `manifest.xml` and `workflow_package.zip` files are generated even if this is not selected.
Generate into the project	Place the generated mobile workflow package and its files in the current project.
Generate to an external folder	Place the generated ZIP file containing the mobile workflow package and its generated files into a location outside of the current project. Click **Browse** to select the alternate location.
jQuery Mobile theme	Choose a theme for devices that use the jQuery Mobile as the UI framework. The default is theme B. See the jQuery Mobile documentation at *http://jquerymobile.com/* for information about the jQuery Mobile themes.
Unwired Server profile	Select the Unwired Server profile with which to associate the mobile workflow and extract the user name and password credentials if you are using static authentication.
Deploy to an Unwired Server	Deploy the mobile workflow package to an Unwired Server.
Deploy mode	The deploy mode is automatically set; you cannot change it. • New • Replace • Update

Option	Description
Assign workflow to user(s)	The mobile workflow must be assigned to a device user before the mobile workflow is visible on the user's device. You can assign the same mobile workflow to multiple users. Separate multiple users with a comma. Device users must be registered in Sybase Control Center. Click **Get Users** to select device users from the list. You must have registered device users in Sybase Control Center to populate this.
Validate controls as soon as the user tries to change focus away from them	If this option is unselected, validation occurs only when the screen is saved. If selected, validation occurs as soon as the control loses focus. If validation fails, a help element appears and shows the error message. **Note:** Windows Mobile devices do not support this feature.
Optimize JavaScript in the generated workflow package	The public JavaScript files (API.js, Callbacks.js, Camera.js, and so on) contain the client API functions that you can access for use with your Mobile Workflow package customization. By default, the wizard generates a single JavaScript file (such as SUP0.js, SUP1.js, or SUP2.js), that concatenates these files. Unselect this option if you prefer to use the JavaScript files separately. **Note:** If you are deploying to a BlackBerry 7 or later device, selecting this option can make the workflow open more quickly.

3. Click **Finish**.

   A ZIP file containing the application and its generated files is created and placed in the specified location.

Deployment Modes

These are the deployment modes when you generate the mobile workflow package.

New

The New deployment mode initially generates and deploys the mobile workflow package.

Replace
The Replace deployment mode removes an installed mobile workflow package and installs a new mobile workflow package with the same name and version. The Replace deployment mode acquires a list of assigned devices for the original package, uninstalls the original package, installs the new package with the same name and version, then assigns the orginal device list to the new package, thus preserving any device assignments associated with the original package.

Use the replace deployment mode for minor changes and updates to the mobile workflow, or during initial development.

Update
The Update deployment mode installs a new mobile workflow package with the original package name and assigns a new, higher version number than the existing installed mobile workflow package. During the update operation, a list of assigned devices is acquired from the original package, the new package is installed and assigned a new version number and then the administrator specifies device assignments for the new package from the acquired list of assigned devices. Existing notifications are preserved.

Use the Update deployment mode for major new changes to the mobile workflow.

Hybrid Web Container Customization

Customize the appearance and default behavior of the Hybrid Web Container.

Android Hybrid Web Container Customization

The Android 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.

Before getting started, 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 as JavaDoc in the `docs` directory of the HybridWebContainer project.

Whenever a customization requires a source code modification, there is a reference to “touch points” in the code. These references are annotated with `ANDROID_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 `ANDROID_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.java` file.

Note: After performing any customizations, you must rebuild the Hybrid Web Container. Sybase recommends that you always test your changes before using the resulting application.

Android Customization Touch Points

All code areas associated with Hybrid Web Container customizations are annotated with `ANDROID_CUSTOMIZATION_POINT_<customization>` comment tags, or touch points.

Touch Point	Description
ANDROID_CUSTOMIZATION_POINT_COLORS	Use custom colors for the Hybrid Web Container.
ANDROID_CUSTOMIZATION_POINT_FONTS	Use custom fonts in the Hybrid Web Container.
ANDROID_CUSTOMIZATION_POINT_BRAND	Change application name, copyright, and developer information
ANDROID_CUSTOMIZATION_POINT_SPLASHSCREEN	Add a splash screen to the Hybrid Web Container.
ANDROID_CUSTOMIZATION_POINT_DEFAULTSETTINGS	Set the defaults for the Settings screen.
ANDROID_CUSTOMIZATION_POINT_PRESETSETTINGS	Hard code settings for the Settings screen so they do not show up on the device. This prevents the user from changing the settings.
ANDROID_CUSTOMIZATION_POINT_AUTOSTART	Make the Hybrid Web Container automatically launch a Workflow application.
ANDROID_CUSTOMIZATION_POINT_PIN	Use for PIN screen customizations, or to remove the PIN screen.
ANDROID_CUSTOMIZATION_POINT_SORTING	Sort Workflow application messages based on different criteria.
ANDROID_CUSTOMIZATION_POINT_FILTERING	Filter the list of Workflow application messages so only messages meeting certain criteria are shown.

Touch Point	Description
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSORT	Customize the criteria for how the Workflow application list is sorted.
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH	Make the list of Mobile Workflow packages searchable.
ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST	Customize the Mobile Workflow package list appearance.
ANDROID_CUSTOMIZATION_POINT_CATEGORIZEDVIEWS	Create categorized views of the Mobile Workflow packages.
ANDROID_CUSTOMIZATION_POINT_HTTPHEADERS	Set HTTP headers for the Android Hybrid Web Container to include authentication tokens.

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 Web application icons and name, changing the Mobile Workflow package icons, changing labels and text, adding support for new languages, and so on.

Changing the Hybrid Web Container Icon

Modify the application 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`

1. 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.

2. Rebuild the project.

Changing the Mobile Workflow Package Icon for Android

Modify the Workflow 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 Workflow application icons are named `ampicon<index>.png`, where *<index>* is a number between 30 and 116. The icon `ampicon48.png` is the default Workflow application icon. Any Workflow application that has not had its icon specified uses

this icon. This is also the icon that is shown on the menu item that shows all the Workflow applications.

Each Workflow icon has two associated image files that contain images for processed and unprocessed messages. The files have the names `ampicon<index>.png` and `ampicon<index>p.png`. The second file, with the additional "p" in the name, is the processed message icon, while the other is for unprocessed 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 Sybase Control Center and in Sybase Unwired Workspace. You must remember the original icon, so you can select it in Sybase Unwired WorkSpace and in Sybase Control Center.

1. Identify the image currently used by the Mobile Workflow Package that you want to replace. To find the image that is currently used by the Workflow:
   a) Log into Sybase Control Center.
   b) In **Workflows**, select the Workflow 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. Having extra files in the `drawable` folder may interfere with the indexing of the resources.
3. Rebuild the Hybrid Web Container.

Customizing the About Screen and Other Branding

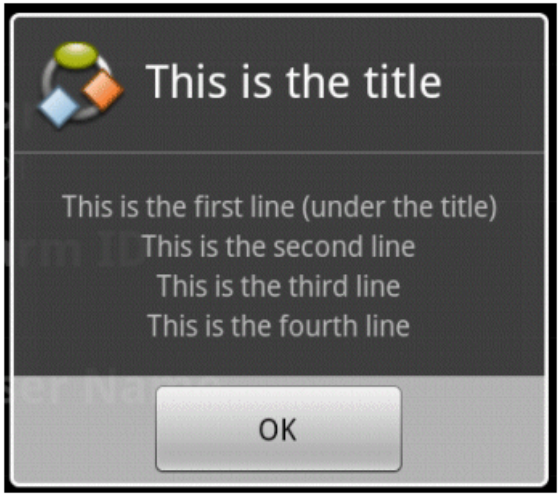
Customize the About screen.

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 files 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 Workflow messages and the Hybrid Web Container.

These examples modify the colors of the Workflow messages. You can also use custom colors for the Hybrid Web Container using similar steps. The related comment tag for customizing colors is `ANDROID_CUSTOMIZATION_POINT_COLORS`.

1. Open the `colors.xml` file, which is located in `...\HybridWebContainer\res\values`, for editing.
2. Find the `ANDROID_CUSTOMIZATION_POINT_COLORS` comment tag and add these tags inside the resources tag:

```
<color name="hybridapp_message_title_color">#F23431</color>
<color name="hybridapp_message_from_color">#FF1111</color>
<color name="hybridapp_message_date_color">#3234F1</color>
```

3. Open the `workflowmessages.xml` file, which is located in `...\HybridWebContainer\res\layout`, for editing.
4. In the `msg_datetime TextView` tag, modify the `android:textColor` attribute to:

   `android:textColor="@color/hybridapp_message_date_color"`
5. Make similar changes to the `msg_from` and the `msg_title` tags, using the color resource defined in step 2.

   If you build the Hybrid Web Container without making any more changes, notice that the custom colors are used for msg_datetime and msg_title, but not for msg_from. This is because the color for msg_from is overridden by the Java code. To stop a custom attribute from being overridden:

   a) Select **Search** > **File** from the menu.
   b) For Containing text, enter `msg_from` and click **Search**.

      The search result shows two files: `workflowmessages.xml` and `UiHybridAppMessagesScreen.java`.
   c) Open the `UiHybridAppMessagesScreen.java` file for editing.
   d) Search the file for "msg_from."

      You will find this line: `TextView tf = (TextView) v.findViewById(R.id.msg_from);`

      The `TextView object tf` represents msg_from.
   e) You are changing the color, so search for “tf.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 Workflow messages and the Hybrid Web Container.

This example customizes the fonts for Workflow messages.

1. Create a new XML file named `attrs.xml` in the `...\HybridWebContainer\res\values\` folder.
2. Open the `attrs.xml` and add this code:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <declare-styleable name="com.sybase.hwc.CustomFontTextView" >
        <attr name="customFont" format="string"/>
    </declare-styleable>
</resources>
```

3. You cannot set the font attribute using the standard TextView control, so you must extend the TextView object by creating a new file named `CustomFontTextView.java`.
4. Add this code to the `CustomFontTextView.java` file:

```
package com.sybase.hwc;

import android.content.Context;
import android.widget.TextView;
import android.text.TextUtils;
import android.util.AttributeSet;
import android.content.res.TypedArray;
import android.graphics.Typeface;

public class CustomFontTextView extends TextView {

   public CustomFontTextView( Context oContext )
   {
      super( oContext );
   }

   public CustomFontTextView( Context oContext, AttributeSet
oAttrs )
   {
      super( oContext, oAttrs );
      setCustomFont( oContext, oAttrs,
R.styleable.com_sybase_hwc_CustomFontTextView,
R.styleable.com_sybase_hwc_CustomFontTextView_customFont );
   }

   private void setCustomFont( Context oContext, AttributeSet
oAttrs, int[] aiAttributeSet, int iFontId)
   {
      TypedArray taStyledAttributes =
oContext.obtainStyledAttributes( oAttrs, aiAttributeSet );
      String sCustomFont =
taStyledAttributes.getString( iFontId );
      if( !TextUtils.isEmpty( sCustomFont ) )
      {
         Typeface oTypeFace = null;

         try
         {
            oTypeFace = getFont( oContext, sCustomFont );
            setTypeface( oTypeFace );
         }
         catch (Exception e)
         {
            System.out.println( "Count not set font!" );
            // can't set the font
         }
      }
      else
      {
         System.out.println( "Custom font string was empty!" );
      }
```

```
   }

   private Typeface getFont( Context oContext, String
sCustomFont )
   {
      String sFullCustomFont = "fonts/" + sCustomFont;
      Typeface oTypeFace =
Typeface.createFromAsset( oContext.getAssets(),
sFullCustomFont );
      return oTypeFace;
   }
}
```

5. Create a `fonts` folder in `...\HybridWebContainer\assets` and add the TTF font file to this new folder.

   For example, Windows fonts are usually in `C:\Windows\Fonts\` if you want to use one of those.

6. Open the `workflowmessages.xml` file for editing and add this attribute to the RelativeLayout tag:

   ```
   xmlns:custom="http://schemas.android.com/apk/res/com.sybase.hwc"
   ```

7. Find the TextView tag with the "ID msg_from" and change the tag from a TextView tag to a "com.sybase.hwc.CustomFontTextView" tag.

8. Add this attribute to the **com.sybase.hwc.CustomFontTextView** tag:

   ```
   custom:customFont="<NAME_OF_YOUR_FONT_FILE.TTF>"
   ```

9. Repeat the above steps for tags with the "id msg_title" and "msg_datetime."

   If you build the Hybrid Web Container without making any more changes, you see that "msg_title" and "msg_datetime" are shown with the custom font, but "msg_from" is not. This is because the font for "msg_from" is overridden in the Java code.

10. To prevent the font from being overridden:

    a) Select **Search** > **File** from the menu.

    b) For **Containing text**, enter `msg_from` and click **Search**.

       The search result shows two files: `workflowmessages.xml` and `UiHybridAppMessagesScreen.java`.

    c) Open the `UiHybridAppMessagesScreen.java` file for editing.

    d) Search the file for "msg_from."

       You will find this line: `TextView tf = (TextView) v.findViewById(R.id.msg_from);`

       The `TextView object tf` represents msg_from.

    e) You are changing the font, so search for “tf.setTypeface.”

       The search results return two occurrences because the text is either bolded or not depending on whether the message has been read. Set bold, italic, or normal style for the text in the same way you specify the font.

    f) To ensure your custom font is used, make these modifications to the two occurrences of the method calls to **setTypeface**:

```
tf.setTypeface( tf.getTypeface(), Typeface.BOLD );

tf.setTypeface( tf.getTypeface(), Typeface.NORMAL );
```

Default Behavior Customization for the Android Hybrid Web Container

Default behavior that you can change includes removing a PIN screen, configuring default values for the Settings screen, automatically launching the Workflow application, sorting Workflow 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.

Automatically Launching a Workflow Application

If you anticipate using the Hybrid Web Container only for a single Hybrid Web application, you can customize the Hybrid Web Container to launch the application directly at start-up.

The related comment tage is `ANDROID_CUSTOMIZATION_POINT_AUTOSTART`.

1. Open the `UiHybridAppMessagesScreen.java` file for editing and navigate to the **public void onCreate(Bundle)** function.
2. Insert the following lines of code just before the “m_sBaseTitle = this.getTitle().toString();” line.

```
HybridApp [] aoHybridApps = HybridAppDb.getInvocableHybridApps();
if(aoHybridApps.length >= 1)
{
   Intent oIntentHybridAppContainer =  new Intent( this,
UiHybridAppContainer.class );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
START_MODE, Consts.START_MODE_WORKFLOW );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
ID, ((HybridAppDb)aoHybridApps[0]).getHybridAppId() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
PROGRESS_TEXT, aoHybridApps[0].getDisplayName() );
   startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_WORKFLOW_CONTAINER );
}
```

3. To close the Hybrid Web Container when the application is finished executing, insert this code after the `if{…}` in the **public void onActivityResult(…)** function:

```
else
{
   finish();
}
```

Note: Implementing this step causes the Hybrid Web Container to exit when the Workflow application exits, so the user cannot navigate to the messages screen. Skip this step to go to the messages screen when exiting the Workflow application.

Using Multiple Hybrid Web Containers on the Same Android Device

It is possible to configure the Hybrid Web Container so that two or more Hybrid Web Containers can co-exist on the same 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 Sybase Control Center, and if you want to use the automatic registration option, you must also add an Application Connection Template.

   See *Sybase Control Center for Sybase Unwired Platform > Administer > Applications > Application Creation > Manually Creating Applications* and *Sybase Control Center for Sybase Unwired Platform > Administer > Applications > Application ID Overview*.

   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 Mobile Workflow Packages

You can sort and filter the list of Mobile Workflow packages.

By default, the Hybrid Web Container displays Mobile Workflow 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 Workflow Messages

Sort Workflow messages based on different criteria.

The comment tag associated with sorting Workflow 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 Workflow Messages

Filter the list of Workflow application messages so only messages that meet specified criteria are shown.

The comment tag associated with Workflow 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 all but the low importance messages to appear in the message list, you can change the code to the following:

```
// Eliminate low importance messages.
      ArrayList<Message> filteredMessages =
MessageDb.getMessages();
      for( int iMessageIndex = 0; iMessageIndex <
filteredMessages.size(); iMessageIndex++ )
      {
        if( filteredMessages.get(iMessageIndex).getMailPriority()
==  com.sybase.mo.AmpConsts.EMAIL_STATUS_IMPORTANCE_LOW )
         {
            filteredMessages.remove(iMessageIndex);
            //we need to decrement the index so we don't skip an
element now
            iMessageIndex--;
         }
      }
      return filteredMessages;
```

Modifying the Mobile Workflow Package List Appearance

Change how the Workflow packages are shown on the device.

The comment tag associated with customizing the Workflow package list appearance is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST`.

To show the list of applications, the HybridWebContainer calls the `getHybridAppScreenClass()` method in `CustomizationHelper.java`. That method returns the class that displays the list. The default class is `UiHybridAppScreen`.

This example changes the view from a list view to a gallery view.

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 CustomizationHelper to return the name of your new class.

2. Save the file.

Creating a Gallery View

Change the Mobile Workflow Package list view to a gallery view.

The comment tag associated with creating categorized views is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPLIST`.

1. Add an XML layout called `hybridappgallery.xml` to the HybridWebContainer project.
2. Match your `hybridappgallery.xml` layout to:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Gallery xmlns:android="http://schemas.android.com/apk/res/
android"
        android:id="@+id/gallery"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

3. Create a new activity for the HybridWebContainer.
   a) Open the `AndroidManifest.xml` file.
   b) Click the **Application** tab.
   c) In the Application Nodes section (at the bottom left), click **Add**.
   d) Choose **Activity** and click **OK**.
   e) Select the new activity and change its name to `com.sybase.hwc.HybridAppGalleryActivity`.
   f) Click **Name*** to generate the stub Java file.
   g) Click **Finish**.
4. Enter this code into the `HybridAppGalleryActivity.java` file:

```
package com.sybase.hwc;

import java.util.ArrayList;
import java.util.Vector;
import java.util.Arrays;
```

```
import com.sybase.hybridApp.*;
import com.sybase.hybridApp.amp.Consts;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;

public class HybridAppGalleryActivity extends Activity {

   ImageAdapter m_adapter;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       setContentView(R.layout.hybridappgallery);

       Gallery oGallery = (Gallery) findViewById(R.id.gallery);
       m_adapter = new ImageAdapter(this);
       oGallery.setAdapter(m_adapter);

       oGallery.setOnItemClickListener(new OnItemClickListener ()
       {
         public void onItemClick(AdapterView parent, View v, int
position, long id)
          {
             startHybridApp(parent, v, position, id);
          }
       });
   }


    public void startHybridApp(AdapterView  oParent, View  v, int
iPos, long id )
    {
        Intent oIntentHybridAppContainer =  new Intent( this,
UiHybridAppContainer.class );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
START_MODE, Consts.START_MODE_WORKFLOW );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
ID, m_adapter.getItem( iPos ).getHybridAppId() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
PROGRESS_TEXT, m_adapter.getItem( iPos ).getDisplayName() );
```

```
        startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_WORKFLOW_CONTAINER );
    }

   @Override
   public void onActivityResult( int iRequestCode, int
iResultCode, Intent relaunchData )
   {
      super.onActivityResult( iRequestCode, iResultCode,
relaunchData );
      if ( iRequestCode == Consts.INTENT_ID_WORKFLOW_CONTAINER &&
iResultCode == Consts.RESULT_RELAUNCH )
      {
            Intent oIntentHybridAppContainer =  new Intent( this,
UiHybridAppContainer.class );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
START_MODE, Consts.START_MODE_WORKFLOW );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
ID, relaunchData.getIntExtra( Consts.INTENT_PARAM_WORKFLOW_ID,
0 ));

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
PROGRESS_TEXT,
relaunchData.getStringExtra( Consts.INTENT_PARAM_WORKFLOW_PROGRES
S_TEXT ));
            startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_WORKFLOW_CONTAINER );
      }
   }

   public class ImageAdapter extends BaseAdapter
   {
      //int mGalleryItemBackground;
      private Context mContext;
      private Vector<HybridApp> mHybridApps;

      private ArrayList<Integer> mImageIds;

      public ImageAdapter(Context c)
      {
         mContext = c;
         mImageIds = new ArrayList<Integer>();

         //have to get a list of all installed HybridAppss
         mHybridApps = new
Vector<HybridApp>( Arrays.asList(HybridAppDb.getInvocableHybridAp
ps()) );
         for(int iHybridAppIndex = 0; iHybridAppIndex <
mHybridApps.size(); iHybridAppIndex++)
         {
            HybridAppDb oHybridApp = (HybridAppDb)
mHybridApps.get(iHybridAppIndex);
            int iconIndex = oHybridApp.getIconIndex();
            if(iconIndex >= 30 &&
```

```
                iconIndex <= 116)
            {
                //luckily the icon resources are consecutive
                int iResource = 0;
                if(iconIndex < 100)
                {
                    iResource = 0x7f020022;
                    iResource += (iconIndex - 30)*2;
                }
                else
                {
                    iResource = 0x7f020000;
                    iResource += (iconIndex - 100)*2;
                }
                mImageIds.add(new Integer(iResource));
            }
        }
    }

    public int getHybridAppId(int position)
    {
        return
((HybridAppDb)mHybridApps.get(position)).getHybridAppId();
    }

    public String getDisplayName(int position)
    {
        return
((HybridAppDb)mHybridApps.get(position)).getDisplayName();
    }

    public int getCount()
    {
        return mImageIds.size();
    }

    public HybridAppDb getItem(int position)
    {
        return (HybridAppDb)mHybridApps.get(position);
    }

    public long getItemId(int position)
    {
        return position;
    }

   public View getView(int position, View convertView, ViewGroup
parent)
    {
        ImageView imageView = new ImageView(mContext);


imageView.setImageResource(mImageIds.get(position).intValue());
        imageView.setLayoutParams(new
Gallery.LayoutParams(150,100));
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);
```

```
            return imageView;
        }
    }

}
```

5. Save the file.
6. Open the `CustomizationHelper.java` file, which is located in the `...\HybridWebContainer\src\com\sybase\hwc` folder and edit the `getHybridAppScreenClass()` method, to change the class returned to your new class.

   That class must extend **Activity**.
7. Update the `manifest.xml` file to include the new activity you create.

Creating Categorized Views

Create categories so that Workflow applications 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. Sybase recommends that you name the final category “Miscellaneous;” this adds all applications and messages that do not match a category to the Miscellaneous category. Also in this example, all applications that belong to a category must include the category name contained in their display name. For example, an application named “Financial Claim” belongs in the “Financial” category.

There are other ways to determine categories; if you know the names of the applications in advance, you can simply list all the application names that belong in each category.

1. Create a new XML layout called `category.xml` and paste the following code into the auto generated file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
android:layout_width="fill_parent"
android:layout_height="?android:attr/listPreferredItemHeight"
android:padding="6dip">

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
android:orientation="vertical"
android:layout_width="0dip"
android:layout_weight="1"
android:layout_height="fill_parent">
<TextView
android:id="@+id/category"
android:layout_width="fill_parent"
android:layout_height="0dip"
android:layout_weight="1"
```

```
android:singleLine="true"
android:ellipsize="marquee"
android:gravity="center_vertical"
/>
</LinearLayout>

</LinearLayout>
```

2. Copy the `UiHybridAppScreen.java` file and rename it to your own class, for example, `CategorizedAppScreen.java`, and open it for editing.
3. Add the list of categories to the `UiHybridAppScreen` class, as a public static final member variable:

```
public static final String[] m_asHybridAppCategories =
{ "Financial", "Utilities", "Miscellaneous" };
```

4. Replace the `HybridAppAdapter` class with:

```
private class HybridAppAdapter extends ArrayAdapter<Object>
      {
        private String[] m_asCategories;

         public HybridAppAdapter( Context context, int
textviewResourceId, List<Object> items, String[] categories ){
           super( context, textviewResourceId, items );

           m_asCategories = categories;

          for( int index = 0; index < m_asCategories.length; index
++ )
           {
             this.add( m_asCategories[index] );
           }
         }

         @Override
         public View getView(int position, View convertView,
ViewGroup parent)
         {
           Object oObject = this.getItem(position);
           View v = null;
           if( oObject instanceof HybridApp )
           {
             HybridApp oHybridApp = ( HybridApp ) oObject;
               LayoutInflater vi =
(LayoutInflater)getSystemService(Context.LAYOUT_INFLATER_SERVICE)
;
                  v = vi.inflate(R.layout.workflows, null);

               if ( oHybridApp != null )
               {
                  ImageView ic = (ImageView)
v.findViewById( R.id.workflow_icon );

ic.setImageResource( UiIconIndexLookup.getNormalIconIdForIndex( o
HybridApp.getIconIndex() ));
                  TextView tt = (TextView)
```

```
v.findViewById(R.id.workflow_title);
                  if (tt != null) {
                     tt.setText( oHybridApp.getDisplayName());
                  }
               }
            }
            else
            {  //This position is not a HybridApp, but a category
heading
              String sString = ( String ) oObject;
               LayoutInflater vi = ( LayoutInflater )
getSystemService( Context.LAYOUT_INFLATER_SERVICE );
               v = vi.inflate( R.layout.category, null );
               if( sString != null )
               {
                  TextView tt = (TextView)
v.findViewById( R.id.category );
                  if ( tt != null )
                  {
                     tt.setText( sString );
                  }
               }
            }
            return v;
         }

         public void remove( HybridApp oApp )
         {
            // The object to remove has a different pointer
            // so match it up with the one in the list
            for ( int i = 0; i < this.getCount(); i++ )
            {
              Object oObject = getItem( i );
              if( oObject instanceof HybridApp )
              {
                  HybridApp oTemp = ( HybridApp ) oObject;

                  if ( oApp.getModuleId() == oTemp.getModuleId()
                      && oApp.getVersion() == oTemp.getVersion() )
                  {
                        super.remove( oTemp );
                     return;
                  }
              }

            }
         }


         public void sort()
         {
            // Sorts applications by name
            this.sort( new Comparator<Object>()
            {
               @Override
               public int compare( Object oObject1, Object
```

```
oObject2 )
                {
                  if( oObject1 instanceof String && oObject2
instanceof String)
                  {
                    String sString1 = ( String ) oObject1;
                    String sString2 = ( String ) oObject2;
                 for( int index = 0; index < m_asCategories.length;
index++ )
                     {

if( sString1.equals( m_asCategories[index] ) )
                         {
                            return -1;
                         }
                      if( sString2.equals( m_asCategories[index]) )
                         {
                            return 1;
                         }
                     }

                  }
                else if( oObject1 instanceof HybridApp && oObject2
instanceof HybridApp )
                  {
                   HybridApp oHybridApp1 = ( HybridApp ) oObject1;
                   HybridApp oHybridApp2 = ( HybridApp ) oObject2;

                    int iCategoryIndex1 =
getCategoryIndex( oHybridApp1 );
                    int iCategoryIndex2 =
getCategoryIndex( oHybridApp2 );

                    if( iCategoryIndex1 == iCategoryIndex2 )
                    {
                      return
oHybridApp1.getDisplayName().toLowerCase().compareTo( oHybridApp2
.getDisplayName().toLowerCase() );
                    }
                    else
                    {
                      return iCategoryIndex1 - iCategoryIndex2;
                    }
                  }
                  else
                {  //we have one String (category heading) and one
HybridApp
                    HybridApp oHybridApp = null;
                    String sString = null;
                    int iSwitch = 1;
                    if( oObject1 instanceof HybridApp)
                    {
                      oHybridApp = ( HybridApp ) oObject1;
                      sString = ( String ) oObject2;
                    }
                    else
```

```
                    {
                      oHybridApp = ( HybridApp ) oObject2;
                      sString = ( String ) oObject1;
                      iSwitch = -1;
                    }

                    int iHybridAppCategoryIndex =
getCategoryIndex( oHybridApp );
                  int iCategoryIndex = getCategoryIndex( sString );
                   if( iCategoryIndex <= iHybridAppCategoryIndex )
                    {
                       return 1*iSwitch;
                    }
                    else
                    {
                      return -1*iSwitch;
                    }

                  }

                   return 0;
                }

                private int getCategoryIndex( String sString )
                {
                for( int index = 0; index < m_asCategories.length;
index++ )
                   {

if( m_asCategories[index].equalsIgnoreCase( sString ) )
                      {
                         return index;
                      }
                   }
                   return m_asCategories.length - 1;
                }

              private int getCategoryIndex( HybridApp oHybridApp )
                {
                for( int index = 0; index < m_asCategories.length;
index++ )
                  {

if( oHybridApp.getDisplayName().toLowerCase().indexOf( m_asCatego
ries[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.getInvocableHybridAp
ps() ) ), 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_WORKFLOW_
START_MODE, Consts.START_MODE_WORKFLOW );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
ID, ((HybridAppDb) oHybridApp).getHybridAppId() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
PROGRESS_TEXT, oHybridApp.getDisplayName() );
      startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_WORKFLOW_CONTAINER );
   }
}
```

7. Save the file.

8. Open the `UiHybridAppMessagesScreen.java` file for editing, and in the `onCreateContextMenu` method, make these modifications (changes are shown in **bold**):

```
public void onCreateContextMenu( ContextMenu oMenu, View v,
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**):

```
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:

```
private class MessageAdapter extends ArrayAdapter<Object>
      {
        String[] m_asCategories;

         public MessageAdapter( Context context, int
textviewResourceId, ArrayList<Object> items, String[]
categories ){
            super( context, textviewResourceId, items );

            m_asCategories = categories;

          for( int index = 0; index < m_asCategories.length; index
++ )
           {
              this.add( m_asCategories[index] );
           }
         }
```

```
          @Override
           public View getView(int position, View convertView,
ViewGroup parent) {
             Object oObject = getItem( position );
             View v = null;
             if( oObject instanceof Message )
             {
                   Message oMsg = (Message) oObject;
                   LayoutInflater vi =
(LayoutInflater)getSystemService(Context.LAYOUT_INFLATER_SERVICE)
;
                 v = vi.inflate(R.layout.workflowmessages, null);

                   if ( oMsg != null )
                   {
                      //set the workflow message priority icon
                      ImageView imageForPriority = (ImageView)
v.findViewById( R.id.priority_icon );

                      if ( oMsg.getMailPriority() ==
AmpConsts.EMAIL_STATUS_IMPORTANCE_HIGH )
                      {

imageForPriority.setImageResource( R.drawable.readhi );

imageForPriority.setVisibility( View.VISIBLE );
                      }
                      else if ( oMsg.getMailPriority() ==
AmpConsts.EMAIL_STATUS_IMPORTANCE_LOW )
                      {

imageForPriority.setImageResource( R.drawable.readlow );

imageForPriority.setVisibility( View.VISIBLE );
                      }
                      else
                     imageForPriority.setVisibility( View.GONE );

                       ImageView ic = (ImageView)
v.findViewById( R.id.msg_icon );
                       if ( oMsg.isMsgProcessed() )

ic.setImageResource( UiIconIndexLookup.getProcessedIconIdForIndex
( oMsg.getIconIndex()));
                       else

ic.setImageResource( UiIconIndexLookup.getNormalIconIdForIndex( o
Msg.getIconIndex()));
                         TextView tf = (TextView)
v.findViewById(R.id.msg_from);
                        TextView tt = (TextView)
v.findViewById(R.id.msg_title);
                        TextView bt = (TextView)
v.findViewById(R.id.msg_datetime);
                        if ( tf != null ) {
```

```
                            tf.setText( oMsg.getMsgFrom() );
                         }
                         if (tt != null) {
                               tt.setText( oMsg.getSubject());
                         }
                         if(bt != null){
                            Calendar dtReceived =
Calendar.getInstance();

dtReceived.setTime( oMsg.getReceivedDate() );

                          Calendar dtNow = Calendar.getInstance();

                            if ( dtNow.get( Calendar.YEAR ) ==
dtReceived.get(  Calendar.YEAR ) &&
                                  dtNow.get( Calendar.MONTH ) ==
dtReceived.get(  Calendar.MONTH ) &&
                             dtNow.get( Calendar.DAY_OF_MONTH ) ==
dtReceived.get(  Calendar.DAY_OF_MONTH ) )
                            {
                               bt.setText( ( new
SimpleDateFormat( "hh:mm
a" ) ).format( oMsg.getReceivedDate() ) );
                            }
                            else {
                          bt.setText( ( new SimpleDateFormat( "MM/
dd/yy" ) ).format( oMsg.getReceivedDate() ) );
                            }
                         }

                         // Update appearance unread messages
                      if ( tf != null && tt != null && bt != null )
                         {
                            if ( !oMsg.isMsgRead() )
                            {
                               // Setup view for unread message

v.setBackgroundResource(  R.drawable.unread_selector );

                               tf.setTextColor( Color.WHITE );
                             tf.setTypeface( null, Typeface.BOLD );
                            }
                            else
                            {
                               // Setup view for read message
                               v.setBackgroundResource( 0 );

                               TypedValue tv = new TypedValue();

getTheme().resolveAttribute( android.R.attr.textColorSecondary,
tv, true );


tf.setTextColor( getResources().getColor( tv.resourceId ) );
                          tf.setTypeface( null, Typeface.NORMAL );
                            }
```

```
                        }
                    }
                }
                else
                {
                   String sString = ( String ) oObject;
                  LayoutInflater vi = ( LayoutInflater )
getSystemService( Context.LAYOUT_INFLATER_SERVICE );
                  v = vi.inflate( R.layout.category, null );
                  if( sString != null )
                  {
                     TextView tt = (TextView)
v.findViewById( R.id.category );
                     if ( tt != null )
                     {
                        tt.setText( sString );
                     }
                  }
                }
                return v;
            }

           public void sort()
           {
              // Sorts applications by name
              this.sort( new Comparator<Object>()
              {
                 @Override
                 public int compare( Object oObject1, Object
oObject2 )
                 {
                   if( oObject1 instanceof String && oObject2
instanceof String)
                   {
                     String sString1 = ( String ) oObject1;
                     String sString2 = ( String ) oObject2;
                     for( int index = 0; index <
m_asCategories.length; index++ )
                       {

if( sString1.equals( m_asCategories[index] ) )
                           {
                              return -1;
                           }

if( sString2.equals( m_asCategories[index]) )
                           {
                              return 1;
                           }
                       }

                   }
                  else if( oObject1 instanceof Message && oObject2
instanceof Message )
                   {
                     Message oMessage1 = ( Message ) oObject1;
```

```
                    Message oMessage2 = ( Message ) oObject2;

                    int iCategoryIndex1 =
getCategoryIndex( oMessage1 );
                    int iCategoryIndex2 =
getCategoryIndex( oMessage2 );

                    if( iCategoryIndex1 == iCategoryIndex2 )
                    {
                      return
oMessage1.getReceivedDate().compareTo( oMessage2.getReceivedDate(
) );
                    }
                    else
                    {
                      return iCategoryIndex1 - iCategoryIndex2;
                    }
                  }
                  else
                {  //we have one String (category heading) and one
HybridApp
                    Message oMessage = null;
                    String sString = null;
                    int iSwitch = 1;
                    if( oObject1 instanceof Message)
                    {
                      oMessage = ( Message ) oObject1;
                      sString = ( String ) oObject2;
                    }
                    else
                    {
                      oMessage = ( Message ) oObject2;
                      sString = ( String ) oObject1;
                      iSwitch = -1;
                    }

                    int 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].toLo
werCase() ) >= 0 )
                            {
                              return index;
                            }
                         }
                      }
                    }
                  return m_asCategories.length - 1;
                 }
              });
           }
        }
```

11. In the **onResume** method, make these changes (changes are shown in **bold**):

```
try
         {
            // ANDROID_CUSTOMIZATION_POINT_FILTERING
         ArrayList<Message> alMessages = MessageDb.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**):

```
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 workflow is available
               HybridApp oHybridApp =
HybridAppDb.getHybridApp( oMsg.getModuleId(),
oMsg.getModuleVersion());

             // CR668069 -Check if we can handle transform data -
1mb limit by sqllite database
               try
               {
                  oMsg.getTransformData();
               }
               catch ( Exception ex )
               {
                  MocaLog.getAmpHostLog().logMessage( "Failed to
read transform data", MocaLog.eMocaLogLevel.Normal );

                  new AlertDialog.Builder( this )
                 .setTitle( android.R.string.dialog_alert_title )
                  .setMessage( R.string.IDS_MSG_ERR_MESSAGE_TOO_L
ARGE )
                  .setIcon( android.R.drawable.ic_dialog_alert )
                  .setPositiveButton( android.R.string.ok,
                        new DialogInterface.OnClickListener()
                  {
                 public void onClick( DialogInterface dialog, int
whichButton)
                     {
                        dialog.dismiss();
                     }
                  } )
                  .show();

                  return;
               }



               // Update read flag
               if ( !oMsg.isMsgRead() )
               {
                  m_adapter.notifyDataSetChanged();
               }
```

```
                // Open workflow
              Intent oIntentHybridAppContainer =  new Intent( this,
UiHybridAppContainer.class );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
START_MODE, Consts.START_MODE_MESSAGE );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
MSG_ID, oMsg.getMessageId() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
MODULE_ID, oMsg.getModuleId() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
MODULE_VERSION, oMsg.getModuleVersion() );

oIntentHybridAppContainer.putExtra( Consts.INTENT_PARAM_WORKFLOW_
PROGRESS_TEXT, oMsg.getSubject() );
               startActivityForResult( oIntentHybridAppContainer,
Consts.INTENT_ID_WORKFLOW_CONTAINER );
            }
          }
          catch( Exception ex )
          {
             MocaLog.getAmpHostLog().logMessage( "Failed to open
message. Caught exception - " + ex.getMessage(),
MocaLog.eMocaLogLevel.Normal );
          }
        }
```

13. Open the `CustomizationHelper.java` file, which is located in the `...\HybridWebContainer\src\com\sybase\hwc` folder and edit the `getHybridAppScreenClass()` method, to change the class returned to your new class, which you created in step 2.

    That class must extend **Activity**.

14. Update the `manifest.xml` file to include the new activity you create.

Making the List of Mobile Workflow Packages Searchable

Make the list of Mobile Workflow packages searchable.

The comment tag associated with making the list of Workflow packages searchable is `ANDROID_CUSTOMIZATION_POINT_HYBRIDAPPSEARCH`.

1. Add an XML layout called `emptyview.xml`, and do not add anything to the resulting autogenerated XML file.

2. Open the `workflows_list.xml` file for editing and add the following tag above the ListView tag:

```
<EditText
    android:hint="@string/SEARCH_HINT"
    android:id="@+id/EditTextSearchHybridAppList"
```

```
    android:layout_width="match_parent"
    android:layout_height="47dp" />
```

3. Open `...\Values\Strings.xml` and, between the <resource> and </resource> tags, add:

```
   <string name="SEARCH_HINT">search</string>
```

4. Copy the `UiHybridAppScreen.java` file to your own class name, for example, `SearchableAppScreen.java` and open it for editing.
   a) Add these import statements:

```
import android.widget.EditText;
import android.text.Editable;
import android.text.TextWatcher;
```

   b) Add the following code to the end of the `onCreate` method:

```
final EditText edittext = (EditText)
findViewById(R.id.EditTextSearchHybridAppList);
edittext.addTextChangedListener( new TextWatcher()
{
   public void afterTextChanged( Editable s)
   {
      String sSearchFor = s.toString();
      m_adapter.setSearch( sSearchFor );
      m_adapter.notifyDataSetChanged();
   }

   // stubs; have to implement the abstract methods
   public void beforeTextChanged( CharSequence s, int start, int
count, int after ) {}
   public void onTextChanged( CharSequence s, int start, int
before, int count) {}
});
```

   c) Add this member variable to the `HybridAppAdapter` class:

```
String m_sToSearchFor;
```

   d) Add this line of code to the end of the HybridAppAdapter contstructor 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_SERVI
CE);
   View v = vi.inflate(R.layout.workflows, 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.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
      {
        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_sToSear
chFor );
      if( iIndexOfResult >= 0 )
      {
         m_abDisplayThisApp[index] = true;
      }
   }
}
```

g) Add these methods to the `HybridAppAdapter` class:

```
public void notifyDataSetChanged()
{
  search();
  super.notifyDataSetChanged();
}
public void setSearch( String sSearchFor )
{
   m_sToSearchFor = sSearchFor;
 }
```

h) Add this member variable to the `UiHybridAppScreen` class:

```
private boolean[] m_abDisplayThisApp;
```

5. Open the `CustomizationHelper.java` file, which is located in the `...\HybridWebContainer\src\com\sybase\hwc` folder and edit the `getHybridAppScreenClass()` method, to change the class returned to your new class.

That class must extend **Activity**.

6. Open the file you created in step 4, 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.
7. Update the `manifest.xml` file to include the new activity you create.

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.
- `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 `ANDROID_CUSTOMIZATION_POINT_HTTPHEADERS`.

1. Open the `CustomizationHelper.java` file and make your changes.
2. Save the file.

Testing Android Hybrid Web Containers

After making any customizations to the provided Hybrid Web Container source code, you should test the changes before using the application.

Note: The steps or interface may be different depending on which Android SDK version you are using.

This procedure assumes that you are using Eclipse.

1. Create a new Android virtual device.
   a) a. Open the Android SDK Manager. If you are using Eclipse choose **Window > AVD Manager**.
   b) b. Select **Tools > Manage AVDs**.
   c) Click **New**.
   d) Enter a name for the device and select **Android 2.2** as the target.
   e) Click **Create AVD**.
2. Create a debug configuration for Android applications.

a) In Eclipse, in WorkSpace Navigator, right-click the Hybrid Web Container project and select **Debug as > Debug Configurations**.
b) Right-click **Android Application**.
c) Click **Target**.
d) In Deployment Target Selection Mode, select **Manual** and click **Debug**.

   In the future you will only need to right-click the project and choose **Debug As > Android Application**.
e) In the Android Device Chooser, select **Launch a New Android Virtual Device** (AVD) and select the AVD you created in step 1.
f) Click **Start**.
g) Click **Launch**.

   The Hybrid Web Container automatically launches when the emulator is fully started.

iOS Hybrid Web Container Customization

The Hybrid Web Container project that comes with Sybase Unwired Platform 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.

Before getting started, unzip the directory that contains the Hybrid Web Container project as outlined in *Building the Mobile Workflow Container Using the Provided Source Code*. The Hybrid Web Container project unzips to a directory called `WorkFlow`. Any references to a directory path in these procedures are relative to that top-level `WorkFlow` directory.

The `WorkFlow` directory contains directories such as `Classes`, `libs`, and `includes`, as well as images and other files. It also contains the `WorkFlow.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 inlcuded in the WorkFlow 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. Sybase recommends that you always test your changes before using the resulting application.

iOS Customization Touch Points

All code areas associated with Hybrid Web Container customizations are annotated with `IOS_CUSTOMIZATION_POINT_<customization>` comment tags, or touch points.

Touch Point	Description
IOS_CUSTOMIZATION_POINT_PRESET-SETTINGS	Provides alternative ways to get connection settings so they do not show up on the Settings screen. This prevents the user from changing them. There are variations on this customization.
IOS_CUSTOMIZATION_POINT_DEFAULT-SETTINGS	Set the defaults for the Settings screen.
IOS_CUSTOMIZATION_POINT_AUTOS-TART	Make the Hybrid Web Container automatically launch a Workflow application.
IOS_CUSTOMIZATION_POINT_PIN	Use for PIN screen customizations, or to remove the PIN screen.
IOS_CUSTOMIZATION_POINT_SORTING	Sort Workflow applications or messages based on different criteria.
IOS_CUSTOMIZATION_POINT_FILTERING	Filter the list of Workflow applications or messages so only items meeting certain criteria are shown.
IOS_CUSTOMIZA-TION_POINT_HTTPHEADERS	Set HTTP headers for the iOS Hybrid Web Container to include authentication tokens.
IOS_CUSTOMIZATION_POINT_FONTS	Customize fonts in the Hybrid Web Container.
IOS_CUSTOMIZATION_POINT_SPLASH-SCREEN	Change the splash screen, or the length of time for which it is shown.
IOS_CUSTOMIZATION_POINT_COEXIST-ING	Run two or more independent Hybrid Web Containers on the same device.

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 Web application icons and name, changing the Mobile Workflow package icons, changing labels and text, and adding support for new languages.

Replacing an Existing Mobile Workflow Package Icon

Mobile Workflow package icons appear in the WorkFlow list within the Hybrid Web Container and can be modified by replacing files in the `WorkflowImages` directory.

Each Workflow icon has two associated image files that contain images for processed and unprocessed messages. The files have the names `ampicon<index>.png` and `ampicon<index>p.png`. The second file, with the additional "p" in the name, is the processed message icon, while the other is for unprocessed messages. Processed means the message has been submitted to the server.

1. Identify the image currently used by the Mobile Workflow Package that you want to replace.
2. Go to the `Workflow/WorkflowImages` directory, and replace the `ampicon<index>.png` and `ampicon<index>p.png` image files you identified in step 1 with the new image files.

   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.
3. Rebuild the project.

   Note: The new icons do not show up in Sybase Unwired Platform or Sybase Control Center; those applications continue to display the original icons. You must remember the mapping between the icon you replaced and the icon you replaced it with if you want to use it when creating future Workflow packages.

Changing the Hybrid Web Container Application Icon

Modify the application icon shown on the home screen by replacing the image files in the `WorkFlow` directory.

1. Go to the `WorkFlow` directory, and replace the `Icon-72.png` (iPad) and `Icon.png` (iPhone) image files with the new images.

   Note: The new image files must use the same name as those you replaced, including the file extension, and they must have the same resolution as the original images.
2. Rebuild the project.

Changing the Hybrid Web Container Application Name

Edit a `plist` file to modify the application name.

1. In Xcode, use Project Navigator to find the file named `SUPWorkFlows-Info.plist`.
2. Open the file and change the **Bundle display name** to the new name.
3. Save the file.
4. Rebuild the project.

Splash Screen Customization

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.

Changing the Splash Screen Image

Change the image that is shown on the splash screen.

The splash screen is stored on a per-language basis in the `WorkFlow/<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`).

Note: 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 `WorkFlow/<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 project.

Changing the Length of the Time the Splash Screen Appears

Modify the length of time the splash screen is shown.

1. In Xcode Project Navigator, open the `CustomizationHelper.m` file, which is in the Workflow project, under the `Classes` group folder.
2. Locate the `splashScreenDelay` function, and change it so that it returns the new time interval in seconds.

   Note: Because iOS always tries to display a splash screen even if one does not exist, setting `splashScreenDelay` to return zero does not altogether remove the splash screen, but it will make the elapsed time as short as possible. You can couple this with removing the image files for the splash screen so that nothing is displayed.

3. Rebuild the Workflow 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.

Changes that you can make include:

- Buttons, labels, and error messages – these strings are in `Localizable.strings`, under the `Resources/<language>.lproj` group folders in the Xcode Project Navigator.
- Application branding – strings that identify the application, among other things. These strings are in `Branding.strings`, under the `Resources/<language>.lproj` group folder in the Xcode Project Navigator.
- About box – these strings are in `About.strings`, under the `Resources/Settings.bundle/en.lproj` folder. Expand the `Settings.bundle` under the `Resources` group folder in the Xcode Project Navigator. Here, you can change the company name or the version number that is shown in the About box in the Settings screen.

Keep in mind that for any change you make you must also make equivalent changes for each language if you want your changes to translate across other languages.

When modifying one of the `*.strings` files, you need only to change the second string value. For example, to change the AppId in `Branding.strings`, on this line: `AppId = HWC`, change only the "HWC."

Adding a New Language

Add support for new languages by dropping new `<language>.lproj` directories into the project.

By default, the Hybrid Web Container is localized to several different languages. Localized resources are in `<language>.lproj` directories and group folders throughout the project, where *<language>* may be the full language name, or a two-digit country code. The simplest way to add a new language is to copy existing `lproj` directories for another language, translate the strings into the new language, and add the new `lproj` directories to the project.

This procedure uses English as a starting point.

1. Copy `WorkFlow/English.lproj` directory to `WorkFlow/<new_language>.lproj`.

   This contains resources for the PIN screens and for the splash screen. You can localize or entirely redesign the PIN screen .
2. Add the newly created `WorkFlow/<new_language>.lproj` directory to the project, at the top level (not under any group folders).
3. In Finder, right-click `WorkFlow/Settings.bundle`, and select **Show Package Contents**.

   The `Settings.bundle` directory opens.
4. Copy `en.lproj` to `<new_language>.lproj`.
5. Translate the strings in `Root.strings` (these are the strings that identify names of settings in the Settings screen) and `About.strings` (associated with the About box).

6. In Xcode, in the Project Navigator, find the newly created `<new_language>.lproj` directory under the `Resources/Settings.bundle`.

   You do not need to explicitly add the new directory to the project, but you should verify it is there.
7. Copy `WorkFlow/strings/English.lproj` to `WorkFlow/strings/<new_language>.lproj`.
8. Translate the strings in `Branding.strings` and `Localizable.strings`.
9. In Project Navigator, add the newly created `WorkFlow/strings/<new_language>.lproj` directory to the project under the **Resources** group folder.

Using Custom Fonts

Change the fonts used in the applications or messages lists.

All code areas associated with font customization are annotated with `IOS_CUSTOMIZATION_POINT_FONTS`.

1. In the Xcode project, in the Project Navigator, find and open `CustomizationHelper.m` file in the `Classes` group folder.
2. Locate the customization tags that accompany several functions that each end in the word `Font`.

   You can override any of these functions to return the font you want to use in the applicable situation. See the comments in the file for how each is used.

   Note: If you replace the default table view as described in *Changing to a New UI Control*, the font settings in `CustomizationHelper.m` will not apply.
3. Save the file and rebuild the project.

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 Mobile Workflow 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 `WorkFlow/<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 `WorkFlow/<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 project.

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, `WorkFlow/<French>.lproj\CreatePasswordViewController.xib.strings`.

1. Open the `CreatePasswordViewController.xib.strings` and `EnterPasswordViewController.xib.strings` files, which are located in `WorkFlow/<language>.lproj`.
2. Modify and save the files.
3. Regenerate the `.xib` files:
   a) Open a Terminal window.
   b) Navigate to the `WorkFlow` directory, and execute:

   ```
   ibtool --strings-file <language>.lproj/<strings file>
   <language>.lproj/<xib file> --write <language>.lproj/
   <xib file>
   ```

   Note: *<language>* must be the same throughout, and the `.strings` file must correspond with the `.xib` file.
4. After rebuilding the `.xib` files, you can return to Xcode and view the new screens before rebuilding the Hybrid Web Container.
5. Rebuild the project.

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 `WorkFlow\Classes`.
2. Find the `usePIN` function and change it to return NO instead of YES.
3. Save the file.
4. Rebuild the project.

Settings Screen Customization

Using Default Connection Settings

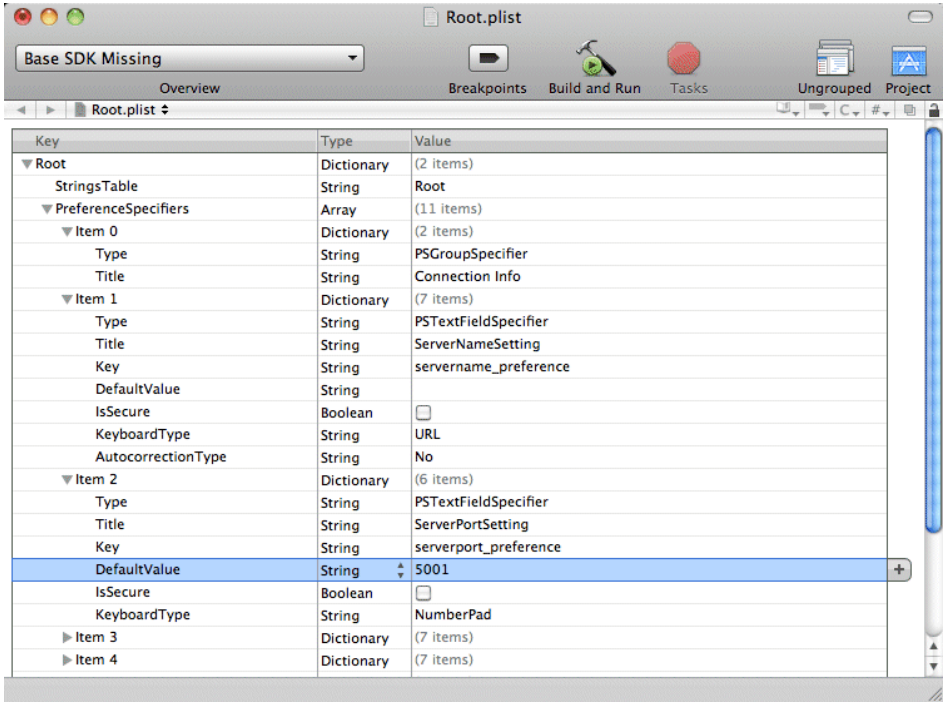
You can customize the Hybrid Web Container so that it is pre-populated with connection settings, or to use certain default values if nothing is provided by the user, or to always use default values on startup.

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.

   This example sets a default value of 5001 for the server port.



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 project.

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. For each property you remove from the Settings screen, you need to provide a way to configure that property.

   See *Providing Default Values for Missing Connection Settings*.

Providing Default Values for Missing Connection Settings

Provide default values for missing connection settings.

1. In the Xcode Project Navigator, open `CustomizationHelper.m`, which is located in the `WorkFlow\Classes` group folder.
2. Find the customization tag, `IOS_CUSTOMIZATION_POINT_DEFAULTSETTINGS`, inside the `registerSettingDefaults` function.

   This contains sample code that reads the current user-entered value, and supplies a hard-coded default value if the current value is invalid. You can follow this approach, or you can obtain the value in other ways, such as by prompting the user or reading from a custom database.
3. Save the file and rebuild the project.

Providing Default Connection Settings at Application Startup

You can provide default connection settings for the application to use when it starts.

1. In the Xcode Project Navigator, open `CustomizationHelper.m`, which is located in the `WorkFlow\Classes` group folder.
2. Locate the customization tags that accompany the functions that begin with `getDefaultConnection`.

   For example, the function `getDefaultConnectionServerName` returns the server name value that is used by the application when it starts up. You can override these functions so they always return a default value.
3. If you are providing a default activation code, you need change the implementation of the `hasCredentials` function.

   In the default implementation, this function checks the settings bundle to see if it contains a nonempty value. Since you know you are providing one, you can make

`hasCredentials` always return YES, or you can call `getDefaultConnectionActivationCode` and test that the returned value is nonempty. Be sure you call `getDefaultConnectionActivationCode` only if you override its implementation so that it does not call `getConfigProperty`.Your implementation would look like this:

```
- (BOOL) hasCredentials {
return [[self getDefaultActivationCode] length] != 0;
    }
```

Note: Providing default connection settings only populates the initial values each time the application starts. The user can still change the values in the Settings screen, but those changes are disregarded when the application starts. This approach still does not prevent a server change from overwriting the properties on the client, but those changes will get reverted once the application restarts.

Mobile Workflow Application Launching Behavior

If you anticipate using the Hybrid Web Container for only a single Workflow application, you can customize the Hybrid Web Container to launch the application directly at start-up.

Note: Make sure you implement a Cancel or Back button when you design your Workflow application. If you fail to do this, and you use this customization, your Workflow application opens automatically, but the user will have no way to navigate back to the list of Workflow messages.

This customization makes the Hybrid Web Container initially load an empty TableView until a Workflow package comes down from the server. When this happens, the first Workflow package that comes down opens. When you click Back or Cancel from within the Workflow, you return to the list of installed Workflow packages. From there, you can manually launch Workflows, or go to the Messages list. The Workflow does not launch automatically again until you return from the Messages list. This is the behavior on iPhone. iPad functions slightly differently, both in portrait and landscape mode.

If there is more than one Workflow application assigned to the user, this customization loads the first one that comes down from the server on the initial synchronization. After that, it loads the application that comes first alphabetically, which is the default sorting behavior. If you plan to assign more than one Workflow to a user and you want to use this customization, it is a good idea to combine this with a filtering or sorting customization.

Automatically Launching a Hybrid Web Application

This customization allows you to automatically launch a Workflow application if one exists.

If there are applications on the device, it loads the first one. It also toggles a flag, so it does not automatically open the application again until the Hybrid Web Container restarts.

Note: If you combine this customization with the Changing to a New UI Control customization, you will need to replace the logic in `SingleWorkflowTableView.m` to implement your own auto-launching behavior.

Note: If you combine this customization with the PIN screen removal customization, this interferes with the auto-launching on iPad devices because the auto-launch on iPad relies on events that are generated by the submission of PIN credentials.

1. In the Xcode Project Navigator, open `CustomizationHelper.m`, which is located in the `WorkFlow\Classes` group folder.
2. Locate the customization tag that accompanies the function `autoLaunchHybridApp`, and override this function to return YES.
3. Save the file.
4. Rebuild the project.

Using Multiple Hybrid Web Containers on the Same iOS Device

You can configure two or more Hybrid Web Containers to coexist on the same device.

All code areas associated with creating co-existing applications are annotated with `IOS_CUSTOMIZATION_POINT_COEXISTING`.

This customization allows two or more independent users to use the same device, but with their own private version of the application. In summary, you need to change the application ID, the bundle identifier, and possibly the URL scheme.

The application ID is used by the server to identify the application, and because of this, you cannot run two applications on the same device with the same application ID. By default, the Hybrid Web Container uses “HWC” for its application ID. Changing the application ID involves a minor change to `CustomizationHelper.m`. Additionally, you must signify to iOS that this is a distinct application. This requires a minor change to a `plist` file. Finally, if you are using Afaria to provision your application, you need to specify a unique URL scheme. This requires changes to the same `plist` file.

1. Change the application ID:
   a) In Xcode Project Navigator, find and open the `CustomizationHelper.m` file, which is located in the `Classes` group folder,
   b) Locate the customization point that accompanies the `getAppId` function, and change it so that it returns a unique name.
   c) Save and close the file.
2. To differentiate this version of the Hybrid Web Container from another:
   a) In Xcode Project Navigator, find and open the `SUPWorkFlows-Info.plist` file, which is located in the `Resources` group folder.
   b) Change the bundle identifier value to something unique.

   c) Save and close the file.

3. If you are using Afaria to provision your application, you must specify a unique URL scheme for your application.
   a) In Xcode Project Navigator, find and open the CustomizationHelper.m file, which is located in the Classes group folder.
   b) Locate the customization point that accompanies the `getAppUrlScheme` function, and change it so that it returns a unique name.
   c) In Xcode Project Navigator, find and open the `SUPWorkFlows-Info.plist` file, which is located in the `Resources` group folder.
   d) Expand the **URL types** item, and expand **Item 0**.
   e) Change the URL identifier value to the value you specified for the Bundle identifier in the previous section.
   f) Save and close the file.
4. Rebuild the project.

Sorting and Filtering the List of Mobile Workflow 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 Workflow packages from appearing according to whatever criteria you specify. You can filter out particular Workflow packages by name, or you can sort Workflow messages by subject. Workflow messages are server-initiated messages associated with a Workflow package, and appear in a separate TableView.

The `WorkflowViewController.h` file defines the interface for a Workflow object. You can sort and filter the properties of this object.

1. Locate the `WorkflowViewController.h` file.

   You do not need to modify this file, but you can view the properties of a WorkFlow object on which you might want to filter or sort.

   This file is included in the `WorkFlow/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 `Workflow.xcodeproj`.
   b) Open the `WidgetFolderController.h` file.
   c) Locate this line: `#import "WorkflowViewController.h"`, right-click inside the quotes, then select **Jump to Definition**.

      Xcode opens the file.
2. Customizations involving filtering and sorting for both Mobile Workflow 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 `Workflow\Clases`.

b) If you are customizing sorting behavior, locate the IOS_CUSTOMIZATION_POINT_SORTING customization tag that accompanies the functions `compareApplicationPackages` and `compareMessages`.

   Overwrite the implementation of one or both of these functions to customize the comparison criteria for application packages and messages, respectively.

c) If you are customizing the filtering behavior, locate the IOS_CUSTOMIZATION_POINT_FILTERING customization tag that accompanies the functions `filterApplicationPackages` and `filterMessages`.

   Overwrite the implementation of one or both of these functions to customize the filtering for application packages and messages, respectively.

3. Save the file.

4. Rebuild the project.

Changing to a New UI Control

You can change the way the list of Workflow packages and messages appear.

Hybrid Web Container uses `UITableView` objects to display the list of WorkFlow 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.

While this may seem daunting at first, it 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 `WorkFlow/Classes` directory, then add these files to the `Classes` group folder in the Project Navigator in Xcode.
3. If you are viewing this guide online from the Sybase 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 `WorkFlow/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.
- `onWorkflowTokenError` – 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 `WorkFlow \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 `onWorkflowTokenError` and `onHTTPError` that you can implement.

   The `onWorkflowTokenError` method is called when Workflow 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

`onWorkflowTokenError`, 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.

PhoneGap Support

PhoneGap is an open source framework that leverages Web technologies such as HTML and JavaScript to access native (system and third-party) functionality across platforms.

Sybase Unwired Platform comes with PhoneGap 1.4.1 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 Mobile Workflow application. See *www.phonegap.com* for information about the supported PhoneGap APIs.

You can use both Hybrid Web Container JavaScript APIs and PhoneGap APIs in a single Workflow application.

Table 1. PhoneGap Supported Features

API	Object and Function	Platform
Accelerometer		
	accelerometer • getCurrentAcceleration **Note:** On iOS, this function must be called after watchAcceleration. • watchAcceleration • clearWatch	• Android • iOS
	Acceleration • x • y • z • timeStamp	• Android • iOS
Camera		

API	Object and Function	Platform
	Camera • getPicture (Camera.PictureSourceType.CAMERA) • getPicture (Camera.PictureSourceType.PHOTOLIBRARY) • getPicture (Camera.PictureSourceType.SAVEDPHOTOALBUM)	• Android • iOS
	CameraOptions • quality • dedestinationType.DATA_URL • dedestinationType.FILE_URI FILE_URI is the default. • allowEdit • encodingType • targetWidth • targetHeight	• Android • iOS
Capture		
	Capture • captureAudio **Note:** On Android, whether this works depends on which application the device uses to record the audio. You can use `media.record` instead to work around this issue. • captureImage • captureVideo	• Android • iOS
	MediaFile • getFormatData	• Android • iOS
Compass		

API	Object and Function	Platform
	compass • getCurrentHeading • watchHeading • clearWatch • watchHeadingFilter	• Android • iOS
	Compass.Heading • magneticHeading • trueHeading • headingAccuracy • timestamp	• Android • iOS
Connection		
	network.connection.type	• Android • iOS
Contacts		
	Contacts.create	• Android • iOS
	Contacts.find	• Android • iOS
	Contact.clone	• Android • iOS
	Contacts.remove **Note:** On Android, there is an issue with contacts not being fully removed. See *https://issues.apache.org/jira/browse/CB-75*.	• Android • iOS

API	Object and Function	Plat-form
	Contacts.save	• Android • iOS
Device		
	Device.name	• Android • iOS
	Device.phonegap	• Android • iOS
	Device.platform	• Android • iOS
	Device.uuid	• Android • iOS
	Device.version	• Android • iOS
Events		
	Deviceready	• Android • iOS
	Pause	• Android

API	Object and Function	Platform
	Resume	• Android
	Online	• Android • iOS
	Offline	• Android • iOS
	Batterycritical	iOS
	Batterylow	iOS
	Batterystatus **Note:** On Android, PhoneGap 1.4.1, this does not work due to a known issue. See *https://issues.apache.org/jira/browse/CB-173*.	iOS
	Menubutton	• Android
	Searchbutton	• Android
File		

API	Object and Function	Plat-form
	DirectoryEntry • copyTo • moveTo • toURI • remove • removeRecursively • getParent • createReader • getDirectory • getFile	• An-droid • iOS
	FileEntry • copyTo • moveTo • toURI • remove • getParent • createWriter • file	• An-droid • iOS
	FileReader • abort • readAsDataURL • readAsText	• An-droid • iOS
	FileWriter • abort • seek • truncate • write	• An-droid • iOS
	DirectoryReader • readEntries	• An-droid • iOS

API	Object and Function	Platform
	LocalFileSystem • requestFileSystem • resolveLocalFileSystemURI	• Android • iOS
	FileTransfer • upload • download	• Android • iOS
Geolocation		
	geolocation • getCurrentPosition **Note:** This function does not work on the Android Galaxy Tab P1000 device. • watchPosition • clearWatch	• Android • iOS
	Position • coords • timestamp	• Android • iOS

API	Object and Function	Platform
	Coordinates • latitude • longitude • altitude • accuracy **Note:** On Android, the returned accuracy property is always null. • altitudeAccuracy **Note:** On Android, the returned altitudeAccuracy property is always null. • heading **Note:** Android only. The returned heading property is always null. • speed **Note:** On Android, the returned speed property is always null.	• Android • iOS
Media		
	Media.play	• Android • iOS
	Media.pause	• Android • iOS
	Media.stop	• Android • iOS
	Media.release	• Android • iOS

API	Object and Function	Platform
	Media.record	• Android • iOS
	Media.startRecord	• Android • iOS
	Media.stopRecord	• Android • iOS
	Media.getCurrentPosition	• Android • iOS
	Media.seekTo	• Android • iOS
	Media.getDuration **Note:** On Android, this function returns a value without an error but always returns -1, which indicates duration is not available.	• Android • iOS
Notification		
	Notification.beep	• Android • iOS

API	Object and Function	Platform
	Notification.confirm	• Android • iOS
	Notification.alert	• Android • iOS
	Notification.vibrate	• Android • iOS
Storage		
	window • OpenDatabase	• Android • iOS
	Database • transaction	• Android • iOS

	SQLTransaction
	• executeSQL
	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:
	`var db = window.openDatabase("aName1", "1.0", "aName1", 0);`
	`db = window.openDatabase("aName2", "1.0", "aName2", 200000);`
		• Android
		• iOS

API	Object and Function	Platform
	SQLResultSet • insertid • rowAffected **Note:** The returned SQLResultSet object does not contain a `rowAffected` property, as the PhoneGap API states. Instead, use `rowsAffected`. • rows	• Android • iOS
	SQLResultSetList • item • length	• Android • iOS
	SQLError • code • message	• Android • iOS
	localStorage • key • getitem • setitem • removeitem • clear	iOS

PhoneGap APIs

The Hybrid Web Container comes with the PhoneGap library linked in and ready to use.

The PhoneGap library included with Sybase Unwired Platform handles common tasks supported by Android and iOS 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 certain initialization has taken place. If you make calls to the PhoneGap API from the customAfterShowScreen function, they should occur only after the PhoneGap subsystem is initialized and ready to execute these calls. For more information, see *http://wiki.phonegap.com/w/page/36868306/ UI%20Development%20using %20jQueryMobile#HandlingPhoneGapsdevicereadyevent*.

You can make PhoneGap calls from the Hybrid Web Container JavaScript, such as `Custom.js`. For example, to save an entry to the contacts database, you can implement something similar to:

```
var contact = navigator.contacts.create();
contact.nickname = "Plumber";
var name = new ContactName();
name.givenName = "Jane";
name.familyName = "Doe";
contact.name = name;
// save
contact.save(onSaveSuccess,onSaveError);
```

Android

Supported PhoneGap APIs allow you to access the native iOS device functionality.

Upgrading the PhoneGap Library Used by the Android Hybrid Web Container

Sybase Unwired Platform comes with PhoneGap 1.4.1 libraries linked in; to upgrade to a later version of the PhoneGap library for use by the Hybrid Web Container, there are a few steps you must perform.

The PhoneGap library that is included with the Hybrid Web Container has been slightly modified to support title bars and loading URLs using binary.

1. Download the PhoneGap package you are upgrading to from *github.com*.
2. Open the `DroidGap.java` file for editing and under the **onCreate method**, comment out this line; doing so allows the screen to show the title bar:

   ```
   getWindow().requestFeature(Window.FEATURE_NO_TITLE)
   ```

3. Replace the existing `spinnerStop` method with the following:

   ```
   public void spinnerStop() {
           if (this.spinnerDialog != null) {
               try
               {
                   this.spinnerDialog.dismiss();
               }
               catch( Exception e )
               {
                // an exception occurs if the activity this dialog is
   associated with is closed
                   // before this dialog
                LOG.d( TAG, "Tried to dismiss a dialog of an activity
   that no longer exists." );
               }
               finally
               {
                   this.spinnerDialog = null;
               }
           }
       }
   ```

4. Add these methods to the `DroidGap` class:

```
public void loadUrlWithData(String url, byte[] abData)
{
  this.loadUrlIntoView(url, abData);
}
private void loadUrlIntoView(final String url) {
  loadUrlIntoView( url, null);
}
```

5. Modify the original `private void loadUrlIntoView(final String url)` method so that the signature accepts binary data.

```
private void loadUrlIntoView(final String url, final byte[]
abData)
```

6. At the bottom of the `private void loadUrlIntoView(final String url)` method, change the line `me.appView.loadUrl(url);` to:

```
if ( abData == null)
  me.appView.loadUrl(url);
else
  me.appView.loadDataWithBaseURL( url, new String( abData ), null,
"utf-8", null );
```

7. Open the `CordovaWebViewClient.java` file and, at line 137, add:

   if (url.indexOf(this.ctx.baseUrl) == 0) { return false; }

   This step is a workaround for a problem PhoneGap 1.4.1 has when loading a local file into an Iframe; it opens in the main window instead. See *https://issues.apache.org/jira/browse/CB-132*.

8. Use Apache Ant to build PhoneGap, run the following command from the PhoneGap framework directory:

   a) In the parent directory of the PhoneGap framework directory, create a file named "VERSION" (no extension).
   b) Edit this file with a text editor and enter the version number (this is the value that is used when naming the .jar file).

      For example, if you have `"1.5.0"` in the VERSION file, the jar is named `cordova-1.5.0.jar`.
   c) Create a file named "local.properties" in the PoneGap framework directory.
   d) Edit this file with a text editor and enter:

      `"sdk.dir=C:\\Program Files\\android-sdk-windows"`

      Ensure the filepath is the correct filepath to your Android installation directory.
   e) From the PhoneGap framework directory, execute:

      ant.bat –f build.xml

   If you do not have Apache Ant installed, you can download it from *http://ant.apache.org/bindownload.cgi*.

9. Put the resulting `cordova-<version>.jar` file in the `HybridWebContainer\libs` folder of the project and delete the old `phonegap-<oldversion>.jar` file from the folder.

10. In the Java perspective in Eclipse, right-click the **HybridWebContainer** project and choose **Properties**.
11. Go to the Java Build Path section, and click the **Libraries** tab.
12. Select the old `phonegap<version>.jar` file and click **Remove**.
13. Add the new `cordova-<version>.jar` file.
14. Open the `UiHybridAppContainer.java` file and update the Hybrid Web Container template code by changing `import com.phonegap.DroidGap` to `import org.apache.cordova.DroidGap`.
15. Open the plugins.xml file, which is located in `HybridWebContainer\res\xml\`, for editing and change all references of `com.phonegap` to `org.apache.cordova`

    For example, change `com.phonegap.App` to `org.apache.cordova.App`.

    These steps complete the upgrade to the new version of PhoneGap. Some additional steps are required to upgrade the Mobile Workflow Forms Editor to use the new version of PhoneGap so that new Mobile Workflow applications reference the correct version of PhoneGap.
16. Navigate to `<UnwiredPlatform_InstallDir>\UnwiredPlatform\Unwired_WorkSpace\Eclipse\sybase_workspace\mobile\eclipse\plugins`.
17. Use WinZip to open `com.sybase.uep.xbw.generatewizard_2.1.3.201202161213.jar`.
18. Replace the `generate\html\js\android\phonegap-1.4.1.javascript` file with the JavaScript file `framework\assets\www\cordova-<version>.javascript`.

    Note: The extension must be `.javascript`, not `.js`. If necessary, modify the extension to `.javascript`.

    To change the PhoneGap version for other platforms as well, replace the `phonegap-1.4.1.javascript` for each platform, for example, for iOS, replace `\html\js\ios\phonegap-1.4.1.javascript`.
19. For each Workflow application that is using the new version of PhoneGap, open the `Generated Workflow\<workflow_name>\html\js\API.js` file for editing.
20. Locate the **loadPhoneGap()** function at the bottom of the file and change the line `jsfile = pre + "js/android/phonegap-1.4.1.javascript";` to `jsfile = pre + "js/android/cordova-<version>.javascript";`, where *<version>* is the new version of PhoneGap.

Note: You must modify the same line for each platform if you want other platforms to use the new version of PhoneGap, for example, for iOS: `jsfile = pre + "js/ios/phonegap-1.4.1.javascript";`.

Performing Additional Steps for Android 2.2 OS

If you upgrade to the new version of PhoneGap and you are using the Android 2.2 operating system, there are some additional steps to perform if you want the Hybrid Web Container to be compatible with Android 2.2.

You must modify the PhoneGap Cordova framework to eliminate all references to features that are not included in Android 2.2.

1. Import the PhoneGap Cordova framework into Eclipse.
2. Right-click the Cordova project and choose **Properties**.
3. In the left pane of the Properties window, choose **Android**.
4. Select **Android 2.2** as the Project Build Target and click **OK**.

   The Cordova project now shows some errors because the code is trying to read and write attributes of a file, but these attributes do not exist in Android 2.2.
5. Open the `ExifHelper.java` file for editing (all errors occur in this file).
6. Comment out each line that has an error and save the file.

   Any functions that depend on these attributes subsequently do not work

Removing PhoneGap from the Android Hybrid Web Container

If PhoneGap functionality is not required, you can make a few modifications to remove all references to the PhoneGap library that is linked to the Hybrid Web Container.

Leaving PhoneGap in place does not cause any issues, but does increase overall application size by about 70KB.

1. Open the `UiHybridAppContainer.java` file for editing and comment out this line:

   ```
   //import com.phonegap.DroidGap;
   ```
2. Change the superclass of `UiWorkflowContainer` from `Droidgap` to `Activity`:

   ```
   public class UiWorkflowContainer extends Activity {
   ```
3. Around line 80, change the `USE_PHONEGAP` variable to false, so the line of code looks like this:

   ```
   private static final boolean USE_PHONEGAP = false;
   ```
4. At this point, there are 5 errors, which are caused by calling methods that were inherited from the `Droidgap` class (but do not exist in the `Activity` class); comment out the 5 lines that cause these errors :

   a) To find these lines, search for "USE_PHONEGAP."

      These lines are all contained in "`if ( USE_PHONEGAP )`" statements.

b) Around line 110, comment out:

```
// super.init();
          // m_oWebView = this.appView;
```

c) Around line 205, comment out:

```
// super.setStringProperty( "loadingDialog", m_sProgressText );
          // super.setIntegerProperty( "loadUrlTimeoutValue",
300000 );
          // super.loadUrlWithData( sBaseURL, abData );
```

5. Switch to the Java perspective, right-click on the **HybridWebContainer** project, and choose **Properties**.
6. Under Java Build Path, click the **Libraries** tab.
7. Remove the PhoneGap library (`phonegap<version>.jar-HybridWebContainer/libs`).
8. Delete the `phonegap<version>.jar` file from the `HybridWebContainer\libs` folder.

iOS

Supported PhoneGap APIs allow you to access the native iOS device functionality.

Upgrading the PhoneGap Library Used by the iOS Hybrid Web Container

Sybase Unwired Platform comes with PhoneGap 1.4.1 libraries linked in; to upgrade to a later version of the PhoneGap library used by the Hybrid Web Container, perfom these steps.

The PhoneGap library that is included with the Hybrid Web Container uses source code that has been modified slightly from the source available from PhoneGap, mainly because the original source does not support some Hybrid Web Container user interface.

Beginning with PhoneGap 1.5.0, PhoneGap rebranded the name PhoneGap to Cordova, which means that when you upgrade, changes to internal class names must be updated.

In addition to the name change, version 1.5.0 ushered in a reorganization of core classes in the PhoneGap implementation. Because of the coupling between the Hybrid Web Container UI code and PhoneGap classes, upgrading requires a careful replacement of the existing PhoneGap integration.

Note: This document describes the process of upgrading the iOS Hybrid Web Container from PhoneGap 1.4.1 to Cordova 1.5.0. Because of its rapid release cycle, Cordova remains a somewhat volatile platform. These instructions are up to date at the time of this writing, but no guarantee is made about how the Cordova implementation may change in the future.

1. Go to *http://www.phonegap.com* and download the Cordova package you are upgrading to.
2. Install the Cordova `.dmg` file.

   This typically places a set of Cordova files under `~/Documents/CordovaLib`.

3. In Xcode, open the **CordovaLib.xcodeproj**.
4. In the Xcode Project Navigator, find and open the `CDVViewController.h` file, which is in the `Classes/Cleaver` group folder.
   a) In the interface declaration, remove `UIWebViewDelegate` from the list of implemented protocols.
   b) Add a `UIViewController` property declaration:
   ```
   @property (nonatomic, retain) UIViewController*
   viewController;
   ```
   c) Find and remove the declaration of the **createGapView** function:
   ```
   #if 0
   -(void) createGapView;
   #endif
   ```
   d) Add these function declarations:
   ```
   -(void) reset;
   -(void)setTheWebView: (UIWebView*) theWebView;
   -(void)setTheViewController: (UIViewController*)
   theViewController;
   ```
5. In Xcode Project Navigator, find and open the `CDVViewController.m` file, which is next to the `CDVViewController.h` file.
   a) Synthesize the `viewController` property:
   ```
   @synthesize viewController;
   ```
   b) In order to avoid memory leak issues, you must move a portion of the **viewDidLoad** function, around line 94, to the **init** function. First, remove the following code from **viewDidLoad**:
   ```
   #if 0
   // read from Cordova.plist in the app bundle
   NSString* appPlistName = @"Cordova";
   NSDictionary* cordovaPlist = [[self class]
   getBundlePlist:appPlistName];
   if (cordovaPlist == nil) {
   NSLog(@"WARNING: %@.plist is missing.", appPlistName);
   return;
   }
   self.settings = [[[NSDictionary alloc]
   initWithDictionary:cordovaPlist] autorelease];

   // read from Plugins dict in Cordova.plist in the app bundle
   NSString* pluginsKey = @"Plugins";
   NSDictionary* pluginsDict = [self.settings
   objectForKey:@"Plugins"];
   if (pluginsDict == nil) {
   NSLog(@"WARNING: %@ key in %@.plist is missing! Cordova will
   not  work, you need to have this key.", pluginsKey,
   appPlistName);
   return;
   }

   // set the whitelist
   ```

```
self.whitelist = [[[CDVWhitelist alloc] initWithArray:
[self.settings objectForKey:@"ExternalHosts"]]
autorelease];

self.pluginsMap = [pluginsDict dictionaryWithLowercaseKeys];
#endif
```

If you do not move this portion of the **viewDidLoad** function, it is called automatically by the OS every time a workflow application is opened. This function contains code that initializes some Cordova components that need to be initialized only once.

c) Move the code you just removed into the **init** function inside the "`if (self != nil)`" block, at the very end of this block.

   Since the **init** function does not return void, change the two return statements in the code you just moved so they read "`return nil;`", to avoid compiler warnings.

d) There are portions of the viewDidLoad function that do some UI initialization that is unnecessary, and which clashes with UI behavior of the Hybrid Web Container. Around line 118, remove this code:

```
#if 0
    NSString* startFilePath = [self
pathForResource:self.startPage];
    NSURL* appURL  = nil;
    NSString* loadErr = nil;

    if (startFilePath == nil) {
       loadErr = [NSString stringWithFormat:@"ERROR: Start Page
at '%@/%@' was not found.", self.wwwFolderName,
self.startPage];
        NSLog(@"%@", loadErr);
        self.loadFromString = YES;
        appURL = nil;
    } else {
        appURL = [NSURL fileURLWithPath:startFilePath];
    }

    [ self createGapView];
#endif
```

In the same function, around line 181, remove this code:

```
#if 0
    if (!loadErr) {
        NSURLRequest *appReq = [NSURLRequest
requestWithURL:appURL
cachePolicy:NSURLRequestUseProtocolCachePolicy
timeoutInterval:20.0];
        [self.webView loadRequest:appReq];
    } else {
        NSString* html = [NSString
stringWithFormat:@"<html><body> %@ </body></html>", loadErr];
        [self.webView loadHTMLString:html baseURL:nil];
    }
#endif
```

e) Remove the implementation of the **createGapView** function:

```
#if 0
- (void) createGapView
{
    CGRect webViewBounds = self.view.bounds;
    webViewBounds.origin = self.view.bounds.origin;

    if (!self.webView)
    {
        self.webView = [[ [ CDVCordovaView alloc ]
initWithFrame:webViewBounds] autorelease];
        self.webView.autoresizingMask =
(UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight);

        [self.view addSubview:self.webView];
        [self.view sendSubviewToBack:self.webView];

        self.webView.delegate = self;
    }
}
#endif
```

f) Since you added a `UIViewController` property to the `CDVViewController` class, you must ensure that view controller gets used at the appropriate time. In the **getCommandInstance** function, there is code that creates an instance of a CDVPlugin object. Find the `if` block that attempts to set the view controller of this object, and change it to use `self.viewController` instead of `self`, for example:

```
if ([obj isKindOfClass:[CDVPlugin class]] && [obj
respondsToSelector:@selector(setViewController:)]) {
[obj setViewController:self.viewController];
}
```

g) Add these implementations for the **reset** function:

```
-(void) reset
{
[self onAppWillTerminate:nil];
self.pluginObjects = nil;
self.webView = nil;
self.commandDelegate = nil;
self.view = nil;
}
```

```
-(void) setTheWebView: (UIWebView*) theWebView {
self.webView = theWebView;
}
```

```
-(void) setTheViewController: (UIViewController
*)theViewController {
self.viewController = (CDVViewController*)theViewController;
}
```

This ensures that the plug-in objects that are saved by Cordova for later use are destroyed when the Workflow application is closed. The plug-in objects each contain a

reference to the WebView, and this causes problems if they are retained after closing a Workflow application and then opening a new one.

h) In the **dealloc** function, add the following line, just before `[super dealloc];`

```
self.whitelist = nil;
```

6. 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:

```
if (self.viewController != nil)
{
   self.viewController = nil;
}
```

7. In the Xcode Project Navigator, find and open the `CDVConnection.m` file, which is located in the `Classes/Commands` group folder, and in the **dealloc** function, locate the line that calls the **removeObserver** function:

```
[[NSNotificationCenter defaultCenter] removeObserver:self
        name:kReachabilityChangedNotification object:nil];
```

Change the line to:

```
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

This causes the connection plug-in object to remove itself as an observer for all of these events when a Workflow application is closed, rather than only for specified events. This is necessary because during initialization, Cordova creates a `CDVConnection` object, then adds this object as an observer to `NSNotificationCenter`. It adds itself as the callback delegate for online/offline connection events, as well as for background/foreground processing notifications. The Hybrid Web Container implementation of Cordova is somewhat nonstandard, in that it expects Cordova to initialize and de-initialize when a Workflow application is opened and closed. If the observers are left, they remain even after the Workflow application is closed, and may cause memory issues.

8. As of PhoneGap version 2.1.0, supporf for the `Contacts` API does not yet exist for iOS 6 devices. To add this support:

a) In Xcode, in the Project Navigator, find and open the `CDVContacts.h` file in the `Classes/Commands` group folder.

Add this protocol definition BEFORE the `CDVContacts` interface declaration:

```
@protocol MissingFeaturesProvider <NSObject>
- (void) requestContactsAccess;
@end
```

Add this function declaration inside the `CDVContacts` interface declaration:

```
+ (void) setContactsAccessDelegate:
(id<MissingFeaturesProvider>)accessProvider;
```

b) In Xcode, in the Project Navigator, find and open the `CDVContacts.m` file in the `Classes/Commands` group folder.

At the very TOP of the `CDVContacts` implementation block, just after `@implementation CDVContacts`, add:

```
static id<MissingFeaturesProvider> s_contactsAccessDelegate =
nil;
```

In the **initWithView:** function, before `return self`, add

```
if (s_contactAccessDelegate != nil)
{
[s_contactsAccessDelegate requestContactsAccess];
}
```

Somewhere within the `CDVContacts` implementation block, add this function definition:

```
+ (void) setContactsAccessDelegate:
(id<MissingFeaturesProvider>)accessProvider
{
s_contactsAccessDelegate = accessProvider;
}
```

There is one final modification necessary to prevent a crash due to freeing unallocated memory. In the **save:withDict:** function, you will see:

```
[aContact release];
CFRelease(addrBook);
```

Modify the code so that it checks whether `addrBook` is nil before trying to release it, like this:

```
[aContact release];
if (addrBook != nil)
{
CFRelease(addrBook);
}
```

9. Build all configurations of the **CordovaLib** target (Debug-iphoneos, Debug-iphonesimulator, Release-iphoneos, and Release-iphonesimulator), which produces files named `libCordova.a`.
10. Copy the `libCordova.a` file for each configuration to the corresponding `libs` folder in `WorkFlow/libs/<configuration>`.

    These folders already have `libMo.a`, existing PhoneGap libraries, and other Sybase Unwired Platform libraries in them. Delete the existing `libPhoneGap.a` for each configuration.
11. You must now include the PhoneGap javascript file, which is under the javascripts folder where PhoneGap was installed, in any workflow application that is built using the new Hybrid Web Container with the new PhoneGap library.

    Starting with PhoneGap 1.5.0, this file is called `cordova-<version>.js`.

    a) For each workflow package that is to be generated, copy this file to the `js` folder in the `Generated Workflow` folder of the Eclipse WorkSpace where the Sybase Mobile SDK is installed.

b) Remove any old instances of this file, and regenerate the workflow package.

Updating the iOS Hybrid Web Container Project

After upgrading the PhoneGap library to Cordova, you must update the Hybrid Web Container project.

1. In Xcode, in the Project Navigator, open **WorkFlow.xcodeproj**.
2. Select the WorkFlow project so that the project settings screen is displayed, then select the WorkFlow target, find the **Other Linker Flags** entry, and for each configuration, replace all instances of "libPhoneGap.a" with "libCordova.a."
3. Create two directories: `WorkFlow/CordovaLib` and `WorkFlow/CordovaLib/Classes`.
4. Copy all of the `.h` files from `~/Documents/CordovaLib/Classes` to `WorkFlow/CordovaLib/Classes`.

   Be sure to get all `.h` files, even in nested directories.
5. Again, open the project settings screen, find the **Header Search Paths** entry, and change all instances of "PhoneGapLib" to "CordovaLib."
6. In the Xcode project, perform two global search-and-replace operations:
   a) Replace all instances of `USE_PHONEGAP` with `USE_CORDOVA`.
   b) Replace all instances of `PHONEGAP_FRAMEWORK` with `CORDOVA_FRAMEWORK`.

      After this, it is assumed that the code no longer includes references to `USE_PHONEGAP`and instead contains references to `USE_CORDOVA`.
7. In the Xcode Project Navigator, find and open the `WorkFlowAppDelegate.h` file in the `Classes` group folder.
   a) Near the top of the file, replace the import of `PhoneGapDelegate.h`.

      The import should look like this:

      ```
      #ifdef USE_CORDOVA
      #ifdef CORDOVA_FRAMEWORK
      #import <Cordova/CDVViewController.h>
      #import <Cordova/CDVContacts.h>
      #else
      #import "CDVViewController.h"
      #import "CDVContacts.h"
      #endif
      #endif
      ```

      Note: `USE_PHONEGAP` has already been changed to `USE_CORDOVA`, and `PHONEGAP_FRAMEWORK` to `CORDOVA_FRAMEWORK`.

   b) Find the declaration of the SUPWorkFlowAppDelegate interface, and in the `#ifdef USE_CORDOVA` block, change the super class from `PhoneGapDelegate` to `CDVViewController`.

c) In the same `#ifdef` block, add the `UIApplicationDelegate` protocol to the list of implemented protocols.

8. In Xcode Project Navigator, find the `WorkFlowAppDelegate.m` file, which is located next to `WorkFlowAppDelegate.h`.

   a) Near the top of the file, remove the entire `#ifdef USE_CORDOVA` block that is currently importing `PhoneGapDelegate.h`, as this is already done in `WorkFlowAppDelegate.h`, and is unnecessary here.

   b) Find the **application:didFinishLaunchingWithOptions:** and remove the entire contents of the `#ifdef USE_CORDOVA` block at the very end:

```
#ifdef USE_CORDOVA
if ( [super
respondsToSelector:@selector(application:didFinishLaunchingWit
hOptions:)])
[super application:[UIApplication sharedApplication]
didFinishLaunchingWithOptions:launchOptions];
#endif
```

   c) Find the **applicationDidBecomeActive:** function and remove the entire contents of the `#ifdef USE_CORDOVA` block at the very end:

```
#ifdef USE_CORDOVA
if ( [super
respondsToSelector:@selector(applicationDidBecomeActive:)])

[super applicationDidBecomeActive:application];
#endif
```

   d) Find the **applicationWillResignActive:** function and remove the entire contents of the `#ifdef USE_CORDOVA` block at the very end:

```
#ifdef USE_CORDOVA
if ( [super
respondsToSelector:@selector(applicationWillResignActive:)])

[super applicationWillResignActive:application];
#endif
```

   e) Find the **applicationWillTerminate:** function and remove the entire contents of the `#ifdef USE_CORDOVA` block at the very end:

```
#ifdef USE_CORDOVA
if ( [super
respondsToSelector:@selector(applicationWillTerminate:)])
[super applicationWillTerminate:application];
#endif
```

   f) Find the **webViewDidFinishLoad:** function, and add this code at the top of that function:

```
#ifdef USE_CORDOVA
if ( [super
respondsToSelector:@selector(onAppDidBecomeActive:)])
[super onAppDidBecomeActive:nil];
#endif
```

g) Find the **applicationWillEnterForeground:** function and change it so it is no longer calling the same function on the super class, but is instead calling **onAppWillEnterForeground:**, like this:

```
if ( [super
respondsToSelector:@selector(onAppWillEnterForeground:)])
[super onAppWillEnterForeground:nil];
```

h) Find the **applicationDidEnterBackground:** function and change it so it is no longer calling the same function on the super class, but is instead calling **onAppDidEnterBackground:**, like this:

```
if ( [super
respondsToSelector:@selector(onAppDidEnterBackground:)])
[super onAppDidEnterBackground:nil];
```

i) In the **initializeAppAfterKeyVaultUnlocked:** find the line `[PGContacts setContactsAccessDelegate:self];` and replace the class name `PGContacts` with `CDVContacts`, like this:

```
[CDVContacts setContactsAccessDelegate:self];
```

9. In the Xcode Project Navigator, locate the file named `VERSION`, at the top level of the project hierarchy, and remove it.

10. In the `Resources` group folder, locate and remove these resources: `PhoneGap.plist`, `Capture.bundle`, and the `www` directory.

11. In the `WorkFlow/PhoneGapLib` directory, locate the `PhoneGap.plist` file and copy it to `WorkFlow/CordovaLib`.

a) Rename the newly copied `PhoneGap.plist` file to `Cordova.plist`.

b) With any text editor, open the `Cordova.plist` file, and perform these two global search-and-replace operations:

- Replace all instances of `com.phonegap` with `org.apache.cordova`.
- Replace all instances of `PG` with `CDV`.

c) Save the file.

12. Locate where `Cordova.framework` was installed on your machine, typically, is in `/Users/Shared/Cordova/Frameworks/Cordova.framework`.

a) Open this framework directory, and copy `VERSION`, `Capture.bundle`, and the `www` directory to `WorkFlow/CordovaLib`.

`Capture.bundle` is a resource bundle that contains around a dozen or so `png` files that enable the `capture.captureAudio` API to function.

b) Remove the `cordova-1.5.0.js` and `index.html` files from the `www` directory.

c) If you plan to use the `notification.beep` API in Cordova, you must also place a file named `beep.wav` in the `www` directory.

See the *PhoneGap documentation* for more details.

13. In the Xcode Project Navigator, under the top level of the project, add the new `VERSION` file.

a) In `WorkFlow/CordovaLib`, in the `Resources group` folder, add the new `Capture.bundle`, `Cordova.plist`, and `www` directory.

   Make sure you create folder references, *not* group references, for the added folders.

14. Clean and rebuild all configurations of the Hybrid Web Container.
15. The Cordova JavaScript file, usually named `cordova-<version>.js`, and which is typically located in `~/Documents/CordovaLib/javascripts`, must now be included in any Workflow targeting the new Hybrid Web Container with the new Cordova library.

    For each Workflow you want to generate, copy this file to the `Generated Workflow \js` folder of the Eclipse WorkSpace where the Sybase Mobile SDK is installed, and remove any old instances of the file.
16. Regenerate the Workflow.

Removing PhoneGap from the iOS Hybrid Web Container

If PhoneGap functionality is not required, you can make a few modifications to remove all references to the PhoneGap library that is linked to the Hybrid Web Container.

Leaving PhoneGap in place does not cause any issues, but does increase overall application size by about 400KB.

1. In Xcode, open the `WorkflowAppDelegate.h` file and comment out this line:

   ```
   #define USE_PHONEGAP 1
   ```
2. In the Build Settings tab, for the Workflow project under **Other Linker Flags**, remove `libPhoneGap.a` for all build configurations.
3. Under **Warning Linker Flags** remove `libPhoneGap.a` for all build configurations.
4. In the Workflow Project Navigator remove references to these files:
   - `VERSION`
   - `PhoneGap.plist`
5. In Xcode, in the Workflow Project Navigator, remove the reference to the `www` directory.
6. In Xcode, in the Workflow Project Navigator, remove the reference to the `Capture.bundle` directory.
7. Clean and rebuild the Workflow project for all configurations.

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 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 PhoneGap 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 plug-in to the Hybrid Web Container.

1. In Eclipse, open the HybridWebContainer project.
2. Open the `plugins.xml` file, which is located in `res/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 a location similar to: `. . . \<projectFolder>\assets\www\<nameOfPlugin>\`, where *<projectFolder>* is the root folder of the project, and *<nameOfPlugin>* is the name of the plug-in you are adding.
5. Add your Java source file that implements the custom plugin, for example, `DirectoryListPlugin.java`.

   This example PhoneGap plugin 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 com.phonegap.api.Plugin;
   import com.phonegap.api.PluginResult;
   import com.phonegap.api.PluginResult.Status;

   /**
   ```

```
 * PhoneGap plugin which can be involved in following manner from
javascript
 * <p>
 * result example - {"filename":"/
sdcard","isdir":true,"children":
[{"filename":"a.txt","isdir":false},{..}]}
 * </p>
 * <pre>
 * {@code
 * successCallback = function(result){
 *     //result is a json
 *
 * }
 * failureCallback = function(error){
 *     //error is error message
 * }
 *
 * window.plugins.DirectoryListing.list("/sdcard",
 *                                       successCallback
 *                                       failureCallback);
 *
 * }
 * </pre>
 * @author Rohit Ghatol
 *
 */
public class DirectoryListPlugin extends Plugin {

    /** List Action */
    public static final String ACTION="list";

    /*
     * (non-Javadoc)
     *
     * @see com.phonegap.api.Plugin#execute(java.lang.String,
     * org.json.JSONArray, java.lang.String)
     */
    @Override
    public PluginResult execute(String action, JSONArray data,
String callbackId) {
        Log.d("DirectoryListPlugin", "Plugin Called");
        PluginResult result = null;
        if (ACTION.equals(action)) {
            try {

                String fileName = data.getString(0);
                JSONObject fileInfo = getDirectoryListing(new
File(fileName));
                Log
                        .d("DirectoryListPlugin", "Returning "
                                + fileInfo.toString());
                result = new PluginResult(Status.OK, fileInfo);
            } catch (JSONException jsonEx) {
               Log.d("DirectoryListPlugin", "Got JSON Exception "
                        + jsonEx.getMessage());
                result = new PluginResult(Status.JSON_EXCEPTION);
```

```
            }
        } else {
            result = new PluginResult(Status.INVALID_ACTION);
          Log.d("DirectoryListPlugin", "Invalid action : "+action
+" passed");
        }
        return result;
    }

    /**
     * Gets the Directory listing for file, in JSON format
     * @param file The file for which we want to do directory
listing
     * @return JSONObject representation of directory list. e.g
{"filename":"/sdcard","isdir":true,"children":
[{"filename":"a.txt","isdir":false},{..}]}
     * @throws JSONException
     */
    private JSONObject getDirectoryListing(File file)
            throws JSONException {
        JSONObject fileInfo = new JSONObject();
        fileInfo.put("filename", file.getName());
        fileInfo.put("isdir", file.isDirectory());

        if (file.isDirectory()) {
            JSONArray children = new JSONArray();
            fileInfo.put("children", children);
            if (null != file.listFiles()) {
                for (File child : file.listFiles()) {
                    children.put(getDirectoryListing(child));
                }
            }
        }

        return fileInfo;
    }
}
```

6. Save the file.

   These are all the changes needed for the Hybrid Web Container; you can now build it and install it on the device. What the plug-in actually does is implemented in the Java file in the **execute** function. The rest of this example explains how to test and use the PhoneGap plug-in.

7. Test the plug-in:

   a) Create a new Mobile Workflow application:

      1. Select **File** > **New** > **Mobile Application Project**.
      2. In Project name, enter `PhonegapTest`.
      3. Click **Finish**.

   b) Right-click the **PhonegapTest** project folder and select **New** > **Mobile Workflow Forms Editor**.

   c) Click **Next**.

d) Select **Can be started, on demand, from the client** and click **Finish**.

e) Add an **HtmlView** control to the start screen of the Mobile Workflow application.

For this example, the HtmlView control key's name is "key2." This is the key name you will use in the `custom.js` file for the `customAfterWorkflowLoad()` function.

f) Run the Mobile Workflow Package Generation wizard to create the Generated Workflow directory structure `Generated Workflow\PhonegapTest\ html \js`.

g) Add your JavaScript implementation to the generated `js` folder, for example, `directorylisting.js,`, and paste in this code:

```
/**
 *
 * @return Instance of DirectoryListing
 */
var DirectoryListing = function() {

}

/**
 * @param directory The directory for which we want the listing
 * @param successCallback The callback which will be called when
directory listing is successful
 * @param failureCallback The callback which will be called when
directory listing encouters an error
 */
DirectoryListing.prototype.list =
function(directory,successCallback, failureCallback) {

    return PhoneGap.exec(successCallback,    //Callback which
will be called when directory listing is successful
                     failureCallback,    //Callback which will
be called when directory listing encounters an error
                    'DirectoryListPlugin',  //Telling PhoneGap
that we want to run "DirectoryListing" Plugin
                     'list',            //Telling the plugin,
which action we want to perform
                      [directory]);       //Passing a list of
arguments to the plugin, in this case this is the directory path
};

/**
 * <ul>
 * <li>Register the Directory Listing Javascript plugin.</li>
 * <li>Also register native call which will be called when this
plugin runs</li>
 * </ul>
 */
PhoneGap.addConstructor(function() {
    //Register the javascript plugin with PhoneGap
    PhoneGap.addPlugin('directorylisting', new
DirectoryListing());
});
```

This code has a wrapper for the PhoneGap plug-in execution call, and adds the custom plug-in to the list of known PhoneGap plug-ins.

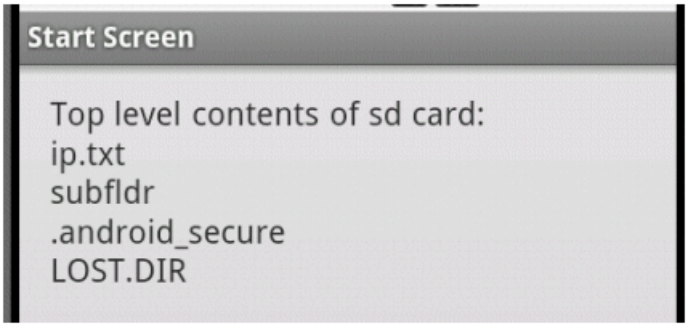
h) Open the `custom.js` file for editing and add this code, which makes use of the plug-in, to the **customAfterWorkflowLoad()** function:

```
var directoryListingFunction = function()
{

        var dl = new DirectoryListing();

        function onSuccess(r)
        {
            var replace = document.getElementById('key2');
            if(replace)
            {
                var theHtml = "<html><head><title>A Title</
title></head><body>Top level contents of sd card:<br/>";
                if(r.children)
                {
                    var index = 0;
                    for(index = 0; index <=
r.children.length;index++)
                    {
                        if(r.children[index]){
                       theHtml += r.children[index].filename +
"<br/>";
                        }
                    }
                }
                else
                {
                    alert("No r.children!!");
                }
                theHtml +="</body></html>";
                replace.innerHTML = theHtml;
            }
        }

        function onError(e)
        {
            alert( "Error: " + e );
        }

        var result = dl.list( "/sdcard", onSuccess, onError );

    }

    directoryListingFunction();
```

i) Generate the Mobile Workflow Package again.

j) Assign the Mobile Workflow to a device that has the modified Hybrid Web Container (that was built after steps 1 through 4).

k) On the device, run the Mobile Workflow application. You may want to add some files to the SD card so you get non-trivial results. The Mobile Workflow application should look something like the following (depending on what you put on the SD card):



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 `WorkFlow/Classes/Plugins`.

1. Copy the `networkActivityMonitor.h` and `networkActivityMonitor.m` files from `WorkFlow/Classes/Plugins` to the `Workflow.xcodeproj` project.
2. Add the `networkActivityMonitor.js` to the Generated Workflow/ <Workflow_Name>/html/js/ directory that corresponds with the Eclipse project that generated the Workflow.
3. Modify `Custom.js` for any event desired to call the new plug-in.

   Here is an example that reacts to a menu item and uses a global variable to toggle the activity indicator on and off.:

```
var gActIndicator = true; // global variable

function customAfterMenuItemClick(screen, menuItem) {
if (screen === "Start" && menuItem === "networkActivityIndicator")
{
window.plugins.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() {
alert("Failed to enable activity indicator");
}
```

4. Add a plug-in entry to `PhoneGap.plist`:

```
Key: networkActivityIndicator
Type: String
Value: networkActivityIndicator
```

5. Generate and deploy the Workflow application.
6. Test the event in the `Custom.js` 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*.

Mobile Workflow Package Customization

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

Adding Custom Code

Use JavaScript code to customize the Mobile Workflow application.

1. Use the Mobile Workflow Package Generation wizard to generate the Mobile Workflow package and its files.

   When the Mobile Workflow package is generated, the `Custom.js` file is generated if not already present in the project. The `Custom.js` file is located in `Generated Workflows\<workflow_project_name>\html\js`.
2. Right-click the `Custom.js` file and select the editor with which to open the file.

3. Add your JavaScript code.

   You can also add your own separate JavaScript files to `Generated Workflows \<workflow_project_name>\html\js`, then add custom code to the `Custom.js` file that calls the functions in the JavaScript files you added. This prevents the `Custom.js` file from becoming extremely long, which makes it difficult if multiple developers are working on the same Mobile Workflow application simultaneously.
4. Save and close the `Custom.js` file.

   Since the `Custom.js` file is generated only if it is not already present in the Mobile Workflow project, this file will not be re-generated if you subsequently re-generate the Mobile Workflow package, so any modifications you make are preserved.
5. Deploy the Mobile Workflow package to Unwired Server.

Adding Local Resources to a Mobile Workflow 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/workflow.html
- .../html/{locale}/workflow.html
- .../html/workflow.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/workflow.html. It is possible to copy the resources into each localization directory or reference the resources from one directory (paying attention to localization paths).

An example of a useful helper function to get the relative path to the HTML directory is:

```
/**
 * Returns relative URL to the html directory
 */
function getRelativeRoot()
{
   return ((resources != null) ? "../" : "")
}

// Helper function usage
var imageElement = document.getElementById("ImageElement");
imageElement.src = getRelativeRoot() + "images/myImage.gif";
```

Generated Mobile Workflow Files

When you generate mobile workflow package files, some files are generated every time and others are generated only under certain conditions.

These files are generated every time you generate the mobile workflow package:

- `manifest.xml` – describes how the contents of the Mobile Workflow package .zip file are organized.
- `workflow_name.zip` – contains all of the Mobile Workflow files, including the Web application files, look and feel files, the JavaScript files, and so on.

These files are regenerated only if you select the **Generate** option in the Mobile Workflow Package Generation wizard:

- *workflow_name*`.html` – an HTML file that describes simple workflow screens and forms. Default name: `workflow.html`.
- *workflow_name*`_CustomLookAndFeel.html` – a workflow html file that adds Sybase JavaScript functions and CSS styles.
- *workflow_name*`_jQueryMobileLookAndFeel.html` – a workflow html file that adds jQuery Mobile functions and CSS styles.
- `WorkflowClient.xml` – contains metadata that specifies how to map the data in the workflow message to and from calls to Mobile Business Object (MBO) operations and object queries.
- `workflow_name.xml` – look and feel file that uses the basic *workflow_name*`.html` file.
- `Resources.js` – allows you to access localized string resources.
- `Workflow.js` – contains functions for common menu, screen, and database operations.

These files are generated only if you select the **Generate** option and the files do not exist:

- `API.js` and `Utils.js` – provide Mobile Workflow functions used to communicate with the Hybrid Web container.
- `Custom.js` – enables you to add JavaScript code to customize the Mobile Workflow application. Your file is preserved each time you regenerate the package.
- `WorkflowMessage.js` – provides functions to access Workflow Message resources.
- All `*.css` files – defines formatting rules to render the screens in HTML.

When you generate the mobile workflow package into the current project, the ZIP file containing the mobile workflow application and its files is placed in the `Generated Workflow` folder in the project, for example, `C:\Documents and Settings\`*`username`*`\workspace\`*`Project_Name`*`\Generated Workflow`. The files are shown in the WorkSpace Navigator. This shows the generated file structure for a project named PurchaseOrder.

The `Generated Workflow\`*`project_name`* folder contains:

- `html` – this folder includes:
  - `workflow.html` – contains all the screens in the Hybrid App, each in its own section. This is used on BlackBerry, Android, and iOS platforms with the **Optimize for performance** look and feel. On Windows Mobile, it is used for all looks-and-feels.

  - workflow_customlookandfeel.html – contains all the screens in the Hybrid App. This is used with the **Optimize for appearance** look and feel on BlackBerry 5.0
  - workflow_jquerymobilelookandfeel.html – contains all the screens in the Hybrid App. This is used with the **Optimize for appearance** look and feel on iOS, BlackBerry 6.0, and Android.
  - js\Custom.js – edit this file to customize the Mobile Workflow application. Since you can modify this file, it is generated only once, or when not already present in the generated files. This ensures that it is not overwritten if you subsequently regenerate the Mobile Workflow package. Examples of ways you can customize the Mobile Workflow application include:
    - Manipulating HTML elements.
    - Writing code that is called before or after generated behavior is invoked for menu items.
    - Implementing custom validation logic.
- <project_name>.zip – contains the mobile workflow application and its files, including the images, user interface, and controls

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-1.0.css – located in Generated Workflow \<mobile_workflow_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 Workflow\<mobile_workflow_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 Workflow \<mobile_workflow_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.

BlackBerry 6.0, Android, and iOS Look and Feel

The default look and feel for BlackBerry 6.0, Android, and iOS is provided by the jQuery Mobile framework.

For this look and feel, the layout of the HTML at a high level is:

- Each screen has a block, contained in a element, with a data-role of "page" and a data-theme of 'a.' Each has a with a data-role of "header," and a child element for the menu. Use the contents of the header to manipulate the menu.

```
<div data-role="page" data-theme='a'
id="Department_createScreenDiv">
      <div data-role="header" data-position="inline">
       <a data-icon="arrow-l"
id="Department_createScreenDivCancel" name="Cancel"
onclick="menuItemCallbackDepartment_createCancel();"> Cancel</a>
       <h1>Department_create</h1>
       <a id="Department_createScreenDivCreate" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();">
Create</a>
     </div>
```

- The menu has one anchor, <a>, for each menu item:

```
<a id="Department_createScreenDivCreate" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();">
Create</a>
```

- In addition to a menu, each screen has a with a data-role of "content," a child element where the controls are hosted. The content has a child with a data-role of "scroller." This in turn has a form with a number of s. The "content" is where you can do customizations, for example, branding.

```
<div data-role="content" class="wrapper" >
       <div data-role="scroller">
        <form name="Department_createForm"
id="Department_createForm">
          <div class="customTopOfFormStyle"  ><span
id="Department_createForm_help" class="help"></span></div>
          <div class="customTopOfFormStyle"
id="topOfDepartment_createForm"></div>
          <div class="editbox">
            <label class="left"
for="Department_create_dept_name_paramKey">Dept name:</label>
            <input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_name_paramKey_help"
class="help"></span>
          </div>
```

The first is a block for use to display help, a element.
The next is a built-in element that can be used to find the top of the form. The last is another built-in element that can be used to find the bottom of the form.
If you look into Custom.js file, it is recommended that you add customizations such as branding to the "TopOf" ScreenKey "Form" and "bottomOf" screenKey "Form." For example:

```
/*
var screenKey = getCurrentScreen();
var form = document.forms[screenKey "Form"];
if (form) {
var topOfFormElem = document.getElementById("topOf" screenKey
"Form");
```

```
! topOfFormElem.innerHTML = "Use this screen to ...";
var bottomOfFormElem = document.getElementById("bottomOf"
screenKey "Form");
bottomOfFormElem.innerHTML = "<a href=\"help.html\">Click here to
open help</a>";
}
*/
```

All the other s in the form correspond to the controls put on that screen during design time in the Mobile Workflow Forms editor. You might see, for example, a that holds a label, <label>, and a textbox, <input>. 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 mobile workflow with this look and feel, without extraneous attributes, looks similar to this:

```
<html>
  <body onload="onWorkflowLoad();">
    <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>
```

BlackBerry 5.0 Look and Feel

A custom look and feel is used, by default, for BlackBerry 5.0.

Each screen has a block, a . Each has a form, <form>, where the controls are hosted. Each form has a number of divs. The first div has a block put aside for use to display help, a 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. All the divs in the form correspond to the controls put on that screen in the Mobile Workflow Forms Editor. You might get, for example, a that holds a label, <label>, and a textbox, <input>.

A typical mobile workflow with this look and feel, without extraneous attributes, looks similar to this:

```
<html>
  <body onload="onWorkflowLoad();">
    <div id="Department_createScreenDiv">
        <form name="Department_createForm"
id="Department_createForm">
          <div class="customTopOfFormStyle"  ><span
id="Department_createForm_help" class="help"></span></div>
          <div class="customTopOfFormStyle"
id="topOfDepartment_createForm"></div>
          <div class="editbox">
            <label class="left"
for="Department_create_dept_name_paramKey">Dept id:</label>
            <input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_id_paramKey_help"
class="help"></span>
          </div>
        </form>
    </div>
  </body>
</html>
```

Optimized for Performance Look and Feel

This is a simple look and feel you can use on all platforms.

Note: Windows Mobile 6.*x* Professional platforms always use the Optimized for performance look and feel, as this platform is not supported by jQuery Mobile.

Choose the **Optimized for performance** option when you configure Mobile Workflow Forms Editor preferences. For this look and feel, the layout of the HTML at a high level is:

- Each screen has a block, a element. Each of those 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 has a form element, <form>, where the controls are hosted.

- Each form has a single table, <table>, with a number of table rows, <tr>. The first table row has a block to display help, a element. The next table row is a built-in element, a table data or <td>, that can be used to find the top of the form.
- The last table row is another built-in element, a <td>, that can be used to find the bottom of the form.
- All the other rows in the form correspond to the controls put on that screen in the Mobile Workflow Forms editor. 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 Mobile Workflow Forms Editor with this look-and-feel.

A typical mobile workflow with this look and feel, without extraneous attributes, looks similar to this:

```
<html>
  <body onload="onWorkflowLoad();">
    <div id="Department_createScreenDiv">
        <ul id="Department_createScreenDivMenu" class="menu">
            <li><a class="nav" name="Create"
onclick="menuItemCallbackDepartment_createSubmit_Workflow();">Creat
e</a></li>
            <li><a class="nav" name="Cancel"
onclick="menuItemCallbackDepartment_createCancel();">Cancel</a></
li>
        </ul>
        <form name="Department_createForm"
id="Department_createForm">
        <table class="screen">
          <tr>
              <td colspan="2"><span id="Department_createForm_help"
class="help"></span></td>
          </tr>
          <tr>
               <td colspan="2" id="topOfDepartment_createForm"></td>
          </tr>
          <tr>
                <td class="left"><label
for="Department_create_dept_name_paramKey">Dept name:</label></td>
                <td class="right"><input class="right" type="text"
id="Department_create_dept_name_paramKey"/><span
id="Department_create_Department_create_dept_name_paramKey_help"
class="help"></span></td>
          </tr>
          <tr><td colspan="2" id="bottomOfDepartment_createForm">
td></tr></table>
        </form>
    </div>
  </body>
</html>
```

Reference

This section describes the generated files and the Workflow client API.

Workflow Client API

Sybase Unwired Platform Mobile Workflow applications include a JavaScript API that open Mobile Workflow applications to customization, from including client-side business logic to changing the presentation layer.

Use the client API to build custom applications to support Sybase Unwired Platform Mobile Workflow features and functionality.

Public JavaScript Functions

The JavaScript files contain the functions that you can access for use with your Mobile Workflow package customization.

These JavaScript files are also included:

- Utils.js – does not contain public functions to call
- Workflow.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*
- phonegap-1.4.1.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 Utility Functions

This file gives you access to the Mobile Workflow general utility functions.

All of general utility functions are synchronous.

Method	Description
guid()	Generates a unique string.
S4()	This function is for use in generating a GUID.
trimSpaces(str)	Removes spaces from the specified string. str – the specified string.

Method	Description
convertToValidJavaScriptName(str)	Converts the specified string to one that is a valid JavaScript function name. str – the specified string.
escapeValue(str)	Replaces all instances in the specified string: • Of the & character with '&'. • Of the < character with '<'. • Of the > character with '>'. • Of the " (quotation mark) character with '"'. • Of the ' (apostrophe) character with '''. str – the specified string.
unescapeValue(str)	Replaces all instances in the specified string: • Of the '&' substring with ‘&’. • Of the '<’ substring with '<’. • Of the ‘>’ substring with ‘>’. • Of the ‘"’ substring with "". • Of the ‘'’ substring with ". str – the specified string.
isIOS()	Returns `true` if the Mobile Workflow application is running on an iOS platform.
isBlackBerry()	Returns `true` if the Mobile Workflow application is running on a BlackBerry platform.
isBlackBerry5()	Returns `true` if the Mobile Workflow application is running on the BlackBerry 5 platform.
isBlackBerry5WithTouchScreen()	Returns true if the Mobile Workflow application is running on the BlackBerry 5 platform with touchscreen capabilities.
isBlackBerry6NonTouchScreen()	Returns true if the Mobile Workflow application is running on a BlackBerry 6 platform without touchscreen capabilities.
isWindowsMobile()	Returns `true` if the Mobile Workflow application is running on a Windows Mobile Professional platform.
isWindows()	Returns true if the Mobile Workflow application is running on a Windows platform.

Method	Description
isAndroid()	Returns true if the Mobile Workflow application is running on the Android platform.
isAndroid3()	Returns true if the Mobile Workflow application is running on the Android 3.0 platform.
isLocaleDatetimeFormat(htmlElement)	Returns `true` if the specified HTML element has an attribute indicating that it should use a locale-specific display. • htmlElement – the specified HTML element. • attributeName – the attribute name.
getAttribute(htmlElement, attributeName)	Reliably returns the specified attribute value for the specified HTML element. • htmlElement – the specified HTML element. • attributeName – the attribute name.
getElementsByTagName(htmlElement, tagName)	Reliably returns the list of elements with the specified tag name, searching only the subtree underneath the specified element. • htmlElement – the specified HTML element. • tagName – the specified tag name.
isSomeFormOfParent(htmlElement, htmlElementToTest)	Determines whether the second specified HTML element is an ancestor of the first specified HTML element. • htmlElement – the specified HTML element. • htmlElementToTest – the HTML element to test.
getFormElementById(formElement, elementID)	Returns the form element with the specified ID. • formElement – the form element. • elementId – the specified ID.
getXMLHTTPRequest()	Reliably returns an XMLHttpRequest object. **Note:** This method is supported only on BlackBerry and Windows Mobile platforms.
getURLParam(paramName)	Returns the specified parameter value from the current URL (window.location.href). paramName – the specified parameter name.

Mobile Workflow Utility Functions

Methods that allow you to access the Mobile Workflow utility functions.

All of the Mobile Workflow utility functions are synchronous.

Method	Description
getISODateString(date)	Returns a string representation of the specified date (currently in yyyy-mm-dd format only). date – the specified date.
getLocaleDateString(date)	Returns a string representation of the specified date using a locale-specific display. date – the specified date.
getISODateTimeStringToDisplay(datetime, precision)	Returns a string representation of the specified date with the specified precision (currently in yyyy-mm-ddThh, yyyy-mm-ddThh:mm or yyyy-mm-ddThh:mm:ss format only, depending on the precision string (HOURS, MINUTES, SECONDS)). • datetime – the specified datetime. • precision – (optional) the specified precision, determines the precision used when rounding.
getLocaleDateTimeString(datetime)	Returns a string representation of the specified datetime using a locale-specific display. datetime – the specified datetime.
getISOTimeString(time, precision)	Returns a string representation of the specified time with the specified precision (currently in hh, hh:mm, or hh:mm:ss format only, depending on the precision string—HOURS, MINUTES, or SECONDS). • time – the specified time. • precision – (optional) the specified precision, determines the precision used when rounding.

Method	Description
getLocaleTimeString(time)	Returns a string representation of the specified time using a locale-specific display. time – the specified time.
getTimeStringToDisplayFromStr(datetime, precision)	Returns a string representation of the specified datetime string as a time with the specified precision—HOURS, MINUTES, or SECONDS. • datetime – the specified datetime. • precision – (optional) the specified precision, determines the precision used when rounding.
getDateFromExpression(toolingStr)	Returns a date for the specified string, which must be either a string representation of a date or of the form “today” or “today+d” or “today-d”, where *d* is a number of days. toolingStr – the date as specified in the Mobile Workflow Forms editor.
parseBoolean(value)	Returns true if the specified string is equal, in a case-insensitive way, to true.
parseDateTime(value)	Returns a date that corresponds to the specified string.
parseTime(value)	Returns a date that corresponds to the specified string.
convertToSUPType(typeAttribute)	Returns the XmlWorkflowMessage type for the type attribute value. • typeAttribute – the type of the specified HTML element.
getHTMLValue(htmlElement, typeAttribute)	Returns a string representation of the specified HTML element’s value. • htmlElement – the specified HTML element. • typeAttribute – the type of the specified HTML element.

Method	Description
setHTMLValue(htmlElement, value, screen-Name, adjustForUTC)	Sets the value of the specified HTML element from the specified string representation of the value. • htmlElement – the specified HTML element. • value – the new value. • screenName – the screen on which the HTML element appears. • adjustForUTC – hardware clock will adjust for coordinated universal time (UTC)
resetHTMLValue(htmlElement, screenName)	Resets the value of the specified HTML element. • htmlElement – the specified HTML element. • screenName – the screen on which the HTML element appears.

Workflow UI Functions

Functions that allow you to access the Mobile Workflow user interface (UI).

The Mobile Workflow UI functions are synchronous.

Method	Description
getCurrentScreen()	Returns the key of the current (open) screen.
setCurrentScreen(screenKey)	Sets the value of the current (open) screen. **Note:** This does not open the specified screen—this function is called only after the screen has already been opened.
getPreviousScreen()	Returns the key of the screen that was open previous to the current screen being opened, if applicable.
getListViewKey(screenName)	Returns the key of the first listview on the specified screen. screenName – the specified screen.

Method	Description
navigateForward(screenKey, listviewKey)	Navigates from the current (open) screen to a new screen with the specified key. • screenKey – the screen to open. • (optional) listViewKey – the listview row for which the details screen is being opened.
navigateBack(isCancelled)	Closes the current screen and returns to the previous screen, if applicable. If the specified parameter value is false, the values on the open screen are persisted to the Mobile Workflow message if they pass validation. isCancelled – true for a Cancel action, false for a Save action.
updateUIFromMessageValueCollection(screenName, values)	Updates the values of the controls on the given screen based on the contents of the specified `MessageValueCollection`. This function will rarely, if ever, need to be called. • screenName – the screen. • values – the message value collection.
updateMessageValueCollectionFromUI(values, screenName, keys, keyTypes, updateModifiedValue)	Updates the contents of the specified `MessageValueCollection` based on the values of the controls on the given screen. In most cases, `saveScreen` is called instead of this function. • screenName – the screen. • values – the message value collection. • (optional) keys – an array of keys, which is a list of only the keys to be updated. • (optional) keyTypes – an array of types for the list of keys, if supplied. • updateModifiedValue –
removeModifiedMessageValuesBasedOnCurrentScreen(values, screenName)	Removes the modified contents of the specified `MessageValueCollection`. This function is called when a screen is cancelled. • screenName – the screen. • values – the message value collection.

Method	Description
saveScreen(values, screenKey, needsValidation)	Saves the contents of the specified screen to the specified MessageValueCollection. This function differs from `updateMessageValue-Collection` in these ways: • If directed, it first performs validation on the screen. • It supports customization. • It is capable of handling the credential request screen. Parameters include: • values – the current message value collection. • screenKey – the current screen. • (optional) needsValidation – false if validation should not be done before saving, true (or unspecified) if validation should be done before saving. Returns true if saving (and validation, if requested) was successful, otherwise, returns false.
saveScreens(skipValidation)	Saves the contents of all open screens if they are successfully validated. skipValidation –

`updateUIFromMessageValueCollection`

To completely override the behavior provided by updateUIFromMessageValueCollection for a given screen, provide a UIUpdateHandler object for that screen. That UIUpdateHandler object has a screenName property, which indicates which screen's behavior it is overriding, and a callback function that indicates the function to call for that screen. That function is passed in the relevant MessageValueCollection object and it is its responsibility to update the controls' values based on its contents. An example of this is:

```
function MyListViewUpdateHandler() {
        this.screenName = "Prev_Expenses";
        this.values;
    }

   MyListViewUpdateHandler.prototype.callback = function(valuesIn)
{
        // Rows returned from RMI Call
        this.values = valuesIn;

        // construct our table
        try  {
```

```
            var mvc =
this.values.getData("PurchaseTrackingJC_findOtherRequests_resultSet
Key");
            var txt = "";
            var htmlOut = "<p>";

            // Do we have any rows to display?
            if (mvc.value.length > 0) {
                // Start the table and header
                htmlOut += "<table id='MyPrevExpensesTable'
class='altrowstable'>";
              htmlOut += "<tr><th>Item Name</th><th>Cost</th></tr>";

                // Draw the rows+H15
               for (var rows = 0; rows < mvc.value.length; rows++) {
                    var mvName =
mvc.value[rows].getData("PurchaseTrackingJC_itemName_attribKey");
                    var mvCost =
mvc.value[rows].getData("PurchaseTrackingJC_itemCost_attribKey");

                    if (mvName && mvCost) {
                        // Alternate the row colors
                        htmlOut += "<tr
onclick='navigateForward(\"Prev_Expenses_Detail\", " +
mvc.value[rows].getKey() + ");'";
                        if (rows % 2 == 0) {
                            htmlOut += " class='evenrowcolor'>";
                        }
                        else {
                            htmlOut += " class='oddrowcolor'>";
                        }

                        htmlOut += "<td>" + mvName.getValue() + "</
td><td>" + mvCost.getValue(); +"</td></tr>";
                    }
                }

                // Finish the table
               htmlOut += "</table>";
            }
            else {
                htmlOut += "No rows returned.";
            }
            htmlOut += "</p>";

            //Now add the table to the document
            var form = document.forms[curScreenKey + "Form"];
            if (form) {
              //var topOfFormElem = document.getElementById("topOf" +
curScreenKey + "Form");

                var topOfFormElem =
document.getElementById("PurchaseTrackingJC_findOtherRequests_resul
tSetKey");
                topOfFormElem.innerHTML = htmlOut;
            }
```

```
        }
        catch (e) {
            alert(e.message);
        }
    } // function callback

    function customAfterWorkflowLoad() {
        //Setup UIHandler to draw our Listview Screen
        UIUpdateHandlers[0] = new MyListViewUpdateHandler();
    }
```

Mobile Workflow Native Device Functions

Access the native features of the device using the native device functions.

Method	Description
setScreenTitle(screenKey, screenTitle)	Sets the specified screen's title based on its sup_screen_title attribute value. • screenKey – the screen. • (optional) screenTitle – an explicit screen title to use rather than the sup_screen_title attribute value.
closeWorkflow()	Closes the Mobile Workflow application.
addMenuItem(menuItemName, functionName, subMenuName, screenToShow)	Allows the user to add a native menu item with the specified name and with the specified callback, which is invoked when the menu item is clicked. • menuItemName – the specified menu item name. • functionName – the specified callback name. • (optional) subMenuName – the specific submenu name for Windows Mobile platforms. • screenToShow – the screen that is about to be shown. This is an optional parameter; if it is not supplied, it is assumed that you are adding the menu item to current screen.
removeAllMenuItems()	Removes all native menu items. **Note:** Removes all menu items from the current screen. On iOS devices, all the menu buttons on the header and footer bars are removed. On Blackberry and Android devices, all native menus are removed. It is recommended that this method be called from **customAfterShowScreen**.

Method	Description
clearCache()	Allows the user to clear the contents of the on-device request result cache for the current workflow.
clearCacheItem(cacheKey)	Allows the user to clear an item from the contents of the on-device request result cache for the current Mobile Workflow form. cacheKey – the key for the item to be removed. See the `Workflow.js` file for details on how to construct this. **Note:** This function is for the "Online Request" cache only and does not apply to cached credentials.
logToWorkflow(message, level, notifyUser)	Allows the user to log a message to the device trace log, which can be remotely retrieved from the server. • message – message to log. • level – level at which to log (ERROR, WARN, INFO, or DEBUG). • notifyUser – if true, an alert dialog is shown before logging commences.
showCertificatePicker()	Opens a form on the device that allows the user to specify the credentials through the use of certificate-based authentication.

showUrlInBrowser(url)	Open the supplied URL in the browser.
	
	Use this method only for opening external HTML files. To open a local HTML file, you must add an HtmlView control to the screen and use something similar to the following for the control's value:
	
	`<iframe src="./local_file_name.html" width="100%" height="80%">`
	`<p>Your browser does not support iframes.</p>`
	`</iframe>`

Method	Description
showAttachmentContents(contents, mimeType, fileName)	Shows the given file contents in a content-appropriate way. The content type is supplied by either the MIME type or the file name, at least one of which must be supplied. The content itself should be presented as a Base64-encoded string. • contents – the Base64-encoded version of the binary content of the attachment to show. • mimeType – (optional) the MIME type of the file. • fileName – (optional) the name of the file.
showAttachmentFromCache(uniqueKey, mimeType, fileName)	Shows the given file contents in a content-appropriate way. The content type is supplied by either the MIME type or the file name, at least one of which must be supplied. The content itself is a unique key supplied earlier to a call to `doOnlineRequest`. • uniqueKey – the unique key for the attachment. • (optional) mimeType – the MIME type of the file. • (optional) fileName – the name of the file.
showLocalAttachment(key)	Shows a local attachment. key – the key.

Method	Description
doOnlineRequest(screenKey, requestAction, timeout, cacheTimeout, errorMessage, errorCallback, workflowMessageToSend, cacheKey)	Allows the user to cause an operation/object query to be invoked. • screenKey – the specified screen on which the submit is occurring. • requestAction – the specified action for the submit. • timeout – specifies the time, in seconds, to wait for a response. • 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. • errorMessage – specifies the string to show if an online request fails. • errorCallback – name of the function to be called if an online request fails. • workflowMessageToSend – Mobile Workflow message that is sent as the input in an online request. • cacheKey – string used as the key for this request in the on-device request result cache.
doAttachmentDownload(screenKey, requestAction, workflowMessageToSend, attachmentKey, requestGUID, downloadCompleteCallback)	• screenKey – the specified screen on which the submit is occurring. • requestAction – the specified action for the submit. • workflowMessageToSend – the Mobile Workflow message that is sent as the input in an online request. • attachmentKey – the specified key of the result is not returned in the workflow message but is, instead, stored on the device for later access. • requestGUID – if specified, represents a unique key that can be used to store/access the cached key value from the request results. Must be specified if keyToCache is specified. Used to support attachments. • downloadCompleteCallback – if specified, is a function that is invoked when the cached value has been downloaded to the device and is ready to be accessed. Must be specified if keyToCache is specified. Used to support attachments.

Method	Description
doSubmitWorkflow(screenKey, requestAction, submitMessage, resubmitMessage)	Allows the user to cause an operation/object query to be invoked. Will close the workflow application when finished. • screenKey – the specified screen on which the submit is occurring. • requestAction – the specified action for the submit. • submitMessage – specifies the string to show if an asynchronous request is successfully submitted. • resubmitMessage – specifies the string to show if an asynchronous request is not submitted because the workflow has already been processed.
showAlertDialog(message, title)	Brings up a message dialog with the specified message, or, optionally, on iOS only, the specified title.
showConfirmDialog(message, title)	Shows a confirmation dialog.

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 to Sybase Mobile Workflow</b><br>
<br>Your activation was successful, the newly created Workflow
requests will automatically be pushed to you.<br>
<br>For more information contact your administrator or visit us
at:<br>
<br>
<a href="javascript:showUrlInBrowser('http://www.sybase.com/
unwiredenterprise')">The Unwired Enterprise</a>
</body>
</html>
```

View an attachment such as an image, a Word document, a .pdf file, and so on as part of the Mobile Workflow package. This example uses an image file.

1. Generate the Mobile Workflow package and its files.
2. In WorkSpace Navigator, go to the location where the generated Mobile Workflow files are located and add an `images` folder under the `html` folder, for example, `Generated Workflow\<Workflow_name>\html\images`.
3. Copy an image to the `images` folder.
4. In the Mobile Workflow Forms editor, add a menu item to the Mobile Workflow.

5. Open the `Custom.js` file with a text editor and edit the method `customBeforeMenuItemClick`:

```
if (screen === "ScreenKeyName" && menuItem === "ShowAttachment") {
    showLocalAttachment("html/images/ipod.jpg");
        return false;
    }
```

6. Save and close the `Custom.js` file.
7. Deploy the Mobile Workflow package to Unwired Server.

Workflow Message Data Functions

Access the mobile workflow application message data functions.

A mobile workflow application 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 getWorkflowMessage(). This returns a WorkflowMessage object. This object has a function, getValues(), that is used to return the top-level MessageValueCollection object. This object has a list of key-value pairs, represented by MessageValue objects and is retrieved by calling getData(key). getData() returns either a single MessageValue object, or an array of MessageValueCollection objects.

Method	Description
getCurrentMessageValueCollection(listKey)	Recommended for listviews so that the user gets the appropriate part, or values, of the message, for the current screen. Each row in the listview corresponds to a Values section. So, for example, if the user is on the details screen and `getCurrentMessageValueCollection` is called, the portion that matches the listview row the user clicked appears. If `getWorkflowMessage` is called, the entire message appears.
getWorkflowMessage()	Allows the user access to the Mobile Workflow message.
getMessageValueCollectionForOnlineRequest(screenKey, requestAction, keys, keyTypes)	Gets the message value collection to be sent in an online request.

A typical mobile workflow message might look similar to this.

```
WorkflowMessage
        .getHeader()        <undefined>
```

```
        .getWorkflowScreen()    "salesorderList_newSOCreate"
        .getRequestAction()    "Submit_Workflow"
        .getValues()         MessageValueCollection
            .getData("salesorderList_newSOCreate_WITHOUT_COMMIT_para
mKey")
                .getKey()
"salesorderList_newSOCreate_WITHOUT_COMMIT_paramKey"
                .getType()    "TEXT"
                .getValue()    "1"
            .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_DOC_TYPE_attribKey")
                .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_DOC_TYPE_attribKey"
                .getType()    "TEXT"
                .getValue()    "1"
            .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_SALES_ORG_attribKey")
                .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_SALES_ORG_attribKey
"
                .getType()    "TEXT"
                .getValue()    "1"
            .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_DISTR_CHAN_attribKey")
                .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_DISTR_CHAN_attribKe
y"
                .getType()    "TEXT"
                .getValue()    "1"
            .getData("BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN
_DIVISION_attribKey")
                .getKey()
"BAPI_SALESORDER_CREATEFROMDAT1_ORDER_HEADER_IN_DIVISION_attribKey"
                .getType()    "TEXT"
                .getValue()    "1"
            .getData("salesorderList_newSOCreate_ORDER_PARTNERS_para
mKey")
                       MessageValue
                .getKey()
"salesorderList_newSOCreate_ORDER_PARTNERS_paramKey"
                .getType()        "LIST"
                .getValue()        MessageValueCollection[]
                    [0].getKey()        "6476c1a4-94e9-e5a4-b903-
caf2ca613c4a"
                    [0].getState()        "add"
                    [0].getData("PARTN_ROLE")
                                MessageValue
                        .getKey()        "PARTN_ROLE"
                        .getType()        "TEXT"
                        .getValue()        "1"
                    [0].getData("PARTN_NUMB")
                                MessageValue
                        .getKey()        "PARTN_NUMB"
                        .getType()        "TEXT"
                        .getValue()        "1"
```

`getCurrentMessageValueCollection`

Handling individual items

```
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;
}
```

Workflow Validation Functions

Workflow validation methods allow you to access the Mobile Workflow application validation functions.

The Mobile Workflow validation methods are synchronous.

Method	Description
setValidationText(helpElement, helpMessage)	Sets the text for the specified help element.

Method	Description
validateRegularExpression(value, regularExp, user-SuppliedMsg, helpElement)	Returns true if the specified value matches the specified regular expression. • value – the specified value. • regularExp – the specified regular expression. • (Optional) userSuppliedMsg –the message to use if the specified value is invalid. • (Optional) helpElement – the help element.
validateDate(value, required, minValue, maxValue, userSuppliedMsg, helpElement)	Returns 0 if the specified value represents a valid date, given the constraints specified; otherwise, returns an error code. • value – the specified value. • (Optional) required – true if the value must be specified, otherwise, false. • (Optional)minValue – the minimum allowable value. • (Optional) maxValue – the maximum allowable value. • (Optional) userSuppliedMsg – the message to use if the specified value is invalid. • (Optional) helpElement – the help element.

Method	Description
validateDateTime(value, required, minValue, maxValue, userSuppliedMsg, helpElement)	Returns 0 if the specified value represents a valid datetime, given the constraints specified; otherwise, returns an error code. • value – the specified value. • (Optional) required – true if the value must be specified, otherwise, false. • (Optional) minValue – the minimum allowable value. • (Optional) maxValue – the maximum allowable value. • (Optional) userSuppliedMsg – the message to use if the specified value is invalid. • (Optional) helpElement – the help element.
validateTime(value, required, minValue, maxValue, userSuppliedMsg, helpElement)	Returns 0 if the specified value represents a valid time, given the constraints specified; otherwise, returns an error code. • value – the specified value. • (Optional) required – true if the value must be specified, otherwise, false. • (Optional) minValue – the minimum allowable value. • (Optional) maxValue – the maximum allowable value. • (Optional) userSuppliedMsg – the message to use if the specified value is invalid. • (Optional) helpElement – the help element.

Method	Description
validateNumber(value, required, minValue, maxValue, numOfDecimals, maxLength, userSuppliedMsg, helpElement)	Returns 0 if the specified value represents a valid number, given the constraints specified; otherwise, returns an error code. • value – the specified value. • (Optional) required – true if the value must be specified, otherwise, false. • (Optional) minValue – the minimum allowable value. • (Optional) maxValue – the maximum allowable value. • (Optional) numOfDecimals – the maximum allowable number of digits after the decimal place. • (Optional) maxLength – the maximum number of characters. • (Optional) userSuppliedMsg – the message to use if the specified value is invalid. • (Optional) helpElement – the help element.
validateText(value, required, maxLength, userSuppliedMsg, helpElement)	Returns 0 if the specified value represents a valid text, given the constraints specified; otherwise, returns an error code. • value – the specified value. • (Optional) required – true if the value must be specified, otherwise, false. • (Optional) maxLength – the maximum number of characters. • (Optional) userSuppliedMsg – the message to use if the specified value is invalid. • (Optional) helpElement – the help element.

Method	Description
validateEmail(value, helpElement)	Returns 0 if the specified value represents a valid email; otherwise, returns an error code. • value– the specified value. • (Optional) helpElement – the help element.
validateControl(screenKey, controlKey, control)	Validates the specified control. • screenKey – the screen the control is on. • controlKey – the control's key. • control – the HTML element for the control.
validateScreen(screenName, values, keysToValidate)	Validates the specified screen. • screenName– the specified screen. • values – the current message value collection. • keysToValidate –
validateAllScreens()	Validates all open screens.

Credential Functions

Access the Mobile Workflow credential functions.

Method	Description
saveLoginCredentials(userName, password)	Saves login credentials to the credential cache. • userName – the user name to save • password – the password to save
saveLoginCertificate(certificate)	Saves login credentials from a certificate. The common name is used for the user name, and the signed certificate is used for the password.

Callbacks.js File

This file contains callback functions.

Callback functions are typically used for event handlers that are asynchronous.

Function	Description
CallbackSet()	Invoked to instantiate a new CallbackSet object.
CallbackSet.prototype.registerCallback()	Invoked asynchronously to handle callbacks from the container.
CallbackSet.prototype.callbackHandler()	Invoked asynchronously to handle callbacks from the container.

Camera.js

Using the Camera.js file, you can take a picture from the camera, or pick one from the photolibrary and use the picture in the mobile workflow.

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 mobile workflow application is restored. If the device does not have a camera application, the function reports that it is not supported.

Function	Description

getPicture(onGetPictureError, onGetPictureSuccess, options)	• onGetPictureError – If an error occurs with the picture chooser or the device camera, an appropriate error code is returned in the onGetPictureError function.
	• onGetPictureSuccess (fileName, response) – the return value is sent to this function
	Note: The function defined must use the format onGetPictureSuccess(fileName, response). fileName is required.
	• options – picture options:
	• PictureOption.SourceType
	  • CAMERA – specifies the built-in camera as the image source where image content is not persisted by the device
	  • PHOTOLIBRARY –specifies the photo library as the image source where image content is already persisted on the device
	  • BOTH –Specifies the built-in camera as the image source where image content is persisted by the device
	  Note: The BOTH source type is not supported on Android or Windows Mobile.
	• PictureOption.DestinationType
	  • imageUri – returns uniform reference identifier for the image
	  • imageData – Deprecated. It is recommended that you use imageUri.

Function	Description
onGetPictureError(errorcode)	This is a user-defined function. Error codes include: • PictureError.NO_ERROR = 0; • PictureError.NOT_SUPPORTED = -1; – getPicture() not implemented, camera not present, • PictureError.IN_PROGRESS = -2; – getPicture() has already been requested but has not yet completed. • PictureError.USER_REJECT = -3; – the user has canceled the request. • PictureError.BAD_OPTIONS = -4; – supplied options were not recognized. • PictureError.TOO_LARGE = -5; – the returned image size was too large to be handled by JavaScript • PictureError.UNKNOWN = -6; – an unknown error occurred.

Function	Description
onGetPictureSuccess(fileName, response)	This is a user-defined function. The return value is sent to the `onGetPictureSuccess` function, in one of the following formats, depending on the `GetPicture` options you specify. You can take this value and set it as a value into the `MessageValueCollection` function. • File name – file name of the image • response – the response will be either a Base64-encoded JPG string or a URI. In the code, onGetPictureSuccess(fileName, response) will be used as onGetPictureSuccess(fileName, imageURI) or onGetPictureSuccess(fileName, imageData). The parameter should coincide with the PictureOption.DestinationType. There are two conditions: • destinationType: PictureOption.DestinationType.IMAGE_URI – returns a uniform reference identifier for the image onGetPictureSuccess(fileName, imageURI) • destinationType: PictureOption.DestinationType.IMAGE_DATA – returns Base64-encoded string onGetPictureSuccess(fileName, imageData)

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 Workflow message as a separate MessageValue to use it on the server. For the HTML image tags, the browser should be able to determine the type through the HTTP connection opened on the URI.

You must create a new option object similar to this:

```
var options =  { destinationType:
PictureOption.DestinationType.IMAGE_URI,
                     sourceType: PictureOption.SourceType.CAMERA
                   };

getPicture(onPictureError, onPictureSuccess, options);
```

The destinationType can be PictureOption.DestinationType.IMAGE_DATA (Base64 string behavior), or the new PictureOption.DestinationType.IMAGE_URI type. Depending on the destination type specified, the picture success callback's second parameter may be a Base64 string or a URI. The source type can be PictureOption.SourceType.CAMERA, PictureOption.SourceType.PHOTOLIBRARY., or PictureOption.SourceType.BOTH.

The image URI passed back is expected to be valid and resolvable to the image by the browser. You can create an HTML image tag with a URI to display the image, for example, `<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 Workflow 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 workflow, the image contents come back from the online request, which can result in too large of a workflow message for the container to handle. It is recommended that you use the submit workflow 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 `getWorkflowMessage().setHasFileMessageValue(true);` for the parse query to be sent to the container.

When uploading the image to the server for a URI, the JavaScript looks similar to this example:

```
var options = { destinationType:
PictureOption.DestinationType.IMAGE_URI, sourceType:
PictureOption.SourceType.PHOTOLIBRARY };

getPicture( onGetPictureError, onGetPictureSuccess, options );

function onGetPictureSuccess(fileName, imageUri){
    // Set file for upload
    var fileDataKey = "Picture_create_fileData_paramKey";

    var messageValue =
getWorkflowMessage().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);
        getWorkflowMessage().getValues().add(fileDataKey,
messageValue);
    }

    getWorkflowMessage().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).
    getWorkflowMessage().getValues().add(fileKey, messageValue);

     getWorkflowMessage().setHasFileMessageValue(true); //
Explicitly tell Workflow about image
}
 var options = { destinationType:
PictureOption.DestinationType.IMAGE_URI, sourceType:
PictureOption.SourceType.CAMERA};
 getPicture( onGetPictureError, onGetPictureSuccess, options );
```

Limitations

The server has a limit of 75MB per parameter, which is what the Hybrid Web Container uses as the XmlWorkflowMessage. Therefore, the server imposes a maximum size limit of 50 MB

(assuming one picture per XmlWorkflowMessage, and no other keys are present). Keep in mind that clients may impose a lower limit than 50MB.

Note:

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

Certificate.js

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.

Function	Description
CertificateStore.parseCertDate(value)	Parses a certificate date.
CertificateStore.createCert(value)	Creates a certificate from the specified JSON string value.
CertificateStore.prototype.certificateLabels(filterSubject, filterIssuer)	Returns a list of all the certificate labels in this store (can be empty). Each certificate in this store has a unique label.
CertificateStore.getDefault()	Returns a certificate without the `signedCertificate` part set.
CertificateStore.prototype.getPublicCertificate(label)	Returns a certificate without the `signedCertificate` part set. Supported platforms: • Windows Mobile Professional • BlackBerry

Function	Description
CertificateStore.prototype.getSignedCertificate(label, password)	Returns the certificate with the specified label, decrypting it if necessary using the specified password; or returns null if the certificate is encrypted and the password is incorrect. Supported platforms: • Windows Mobile Professional • BlackBerry
CertificateStore.prototype.listAvailableCertificatesFromFileSystem(folder, fileExtension)	Returns a list of full path names for the certificate files found in the file system for import. Android platforms only.
CertificateStore.prototype.getSignedCertificateFromFile(filePath, password)	Gets a certificate from a file. Android platforms only.
CertificateStore.prototype.getSignedCertificateFromServer(username, serverPassword, certPassword)	Gets a certificate from the server. iOS platforms only.

getSignedCertificateFromAfaria(commonName, challengeCode)	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. The parameters are:
	• commonName – this parameter is required.
	• challengeCode – this parameter is optional, depending on how the Afaria server is configured.
	For example:
	`var certStore = CertificateStore.getDefault();`
	`afariaCert = certStore.getSignedCertificateFromAfaria(commonName, challengeCode);`

You can choose to set the results of a getSignedCertificate function as the password.

```
certificateLabels(filterSubject, filterIssuer)

// The following script gets all the labels for certificates
// with the provided subject and issuer
var certStore = CertificateStore.getDefault();
var labels = certStore.certificateLabels("MyUser", "mydomain.com");

- getPublicCertificate(label)

// The following script gets the certificate data for the first
// certificate to match the provided subject and issuer
var certStore = CertificateStore.getDefault();
var labels = certStore.certificateLabels("MyUser", "mydomain.com");
var cert = certStore.getPublicCertificate(labels[0]);

- getSignedCertificate(label)

// The following script gets the signed certificate data for the
first
// certificate to match the provided subject and issuer
var certStore = CertificateStore.getDefault();
var labels = certStore.certificateLabels("MyUser", mydomain.com");
var cert = certStore.getSignedCertificate(labels[0]);

var username = cert.subjectCN;
var password = cert.signedCertificate;

- listAvailableCertificatesFromFileSystem(sFolder, sFileExtension)

// The following script gets an array of file paths for files on
// the sdcard with the extension p12
var certStore = CertificateStore.getDefault();
var certPaths = certStore.listAvailableCertificatesFromFileSystem("/
sdcard/", "p12");

- getSignedCertificateFromFile(filePath, password)

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

- getSignedCertificateFromServer(username, serverPassword,
certPassword)

// The following script gets the signed certificate data for the
// user MYDOMAIN\MYUSERNAME from the server
var certStore = CertificateStore.getDefault();
cert = certStore.getSignedCertificateFromServer("MYDOMAIN\
\MYUSERNAME", "myserverpassword", "mycertpassword");
```

Custom.js File

The first time you generate the Mobile Workflow package files, the `Custom.js` file is generated.

In subsequent file generations for the same Mobile Workflow 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 **custom Before** method is invoked and a **customAfter** method is invoked. The Before method returns a boolean. If it returns true, it continues to execute the default behaviour, 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:

Method	Description
customBeforeWorkflowLoad()	Invoked when the application is first launched, before any data is loaded, or screens are opened. Because workflow settings are not yet initialized at this point, you cannot call any SharedStorage functions here.
customAfterWorkflowLoad()	Invoked when the application is first launched, after data is loaded and screens are opened.
customBeforeSubmit(screenKey, requestAction, workflowMessageToSend)	Invoked before an operation or object query is about to be called as the result of the user clicking a Submit menuitem. You can set this to return false to prevent the default behaviour from occurring.
customAfterSubmit(screenKey, requestAction)	Invoked after an operation or object query is called as the result of the user clicking a Submit menuitem.
customBeforeNavigateForward(screenKey, destScreenKey)	Invoked when another screen is about to be opened. Set to false to prevent the screen from being opened.
customBeforeNavigateBackward(screenKey, isCancelled)	Invoked when another screen is about to be opened. You can set to false to prevent the screen from being opened.

Method	Description
customAfterNavigateForward(screenKey, destScreenKey)	Invoked after another screen has been opened.
customAfterNavigateBackward(screenKey, isCancelled)	Invoked after a screen has been closed.
customBeforeShowScreen(screenToShow, screenToHide)	Invoked when a screen is about to be shown. User can return false to prevent the screen from being shown.
customAfterShowScreen(screenToShow, screenToHide)	Invoked after a screen is shown.
customValidateScreen(screenKey, values)	Invoked when the contents of a screen need to be validated. User can return false to indicate that the contents of the screen are not valid.
customBeforeMenuItemClick(screen, menuItem)	Invoked after a menuitem has been clicked. User can return false to prevent the default behavior (which might open a new screen, or perform a submit, and so on) from occurring.
customAfterMenuItemClick(screen, menuItem)	Invoked after a menuitem has been clicked and the default behavior has occurred.
customBeforeSave(screen)	Invoked before a screen's contents are persisted to the Mobile Workflow message. User can return false to prevent the default behaviour from occurring.
customAfterSave(screen)	Invoked after a screen's contents are persisted to the Mobile Workflow message through the default logic.

Method	Description
customConditionalNavigation(currentScreenKey, actionName, defaultNextScreen, conditionName, workflowMessage)	For online request menu items and custom actions, this method is invoked to evaluate the given condition after a given action is executed. If the screen associated with the condition should be navigated to, the condition is true. For server-initiated starting points, the judgement condition is `if((currentScreenKey === SERVERINITIATEDFLAG).` This method is different from the others in two of its attributes: • It returns true or false – the custom code used to implement this method can peer into the workflow message and execute logic. This routine generally does not modify the HTML or anything else. • There is no before or after behavior – this function is executed after the workflow message is received from the server, but before the screen is opened. Therefore, this is executed before the "customBeforeShowScreen()" because this function is used to help decide what screen to show next. Conditions set by the user in the designer are executed serially, and the first one that returns true determines what the start screen is. As soon as a true condition is found, evaluation stops and the screen is executed.
customBeforeReportErrorFromNative(errorString)	Invoked when a native error is reported on. Return false to prevent the default behavior from executing (bringing up an alert dialog)
customAfterReportErrorFromNative(errorString)	Invoked after a native error is reported.
customAfterDataReceived(incomingWorkflowMessage)	Invoked after data is received from the server. This allows you to view and manipulate the data.
customGetPictureURIFromCamera()	Use this method to get a picture URI from the camera for submission to the workflow message.

Method	Description
customGetPictureURIFromLibrary()	Use this method to get a picture URI from the photo library for submission to the workflow message.
customGetPictureURISuccess(fileName, image-URI)	Handles success from one of the URI calls.
customGetPictureDataFromCamera()	Use this method to get picture data from the camera for submission to the workflow message. **Note:** The picture data length should not be too long.
customGetPictureDataFromLibrary()	Use this method to get picture data from the photo library for submission to the workflow message. **Note:** The picture data length should not be too long.
customGetPictureDataSuccess(fileName, imageData)	Handles success from one of the "Data" calls.
customGetPictureError(result)	Invoked after an error is reported.
getMimeType(fileName)	A helper method that allows you to include the MIME data type in the workflow message.

Note: You can delegate the implementation of these functions to different functions supplied in other custom JavaScript files. It is not necessary to include all of your customization logic in the single `Custom.js` file.

```
//Use this method to add custom html to the top or bottom of a form
function customBeforeWorkflowLoad() {

    var form = document.forms[curScreenKey + "Form"];
       if (form) {
          // header
          var topOfFormElem = document.getElementById("topOf" +
curScreenKey + "Form");

          if (topOfFormElem) {
             topOfFormElem.innerHTML = "<img id='ImgSylogo' src='./
images/syLogo.gif'/><br/>";

             // footer
           var bottomOfFormElem = document.getElementById("bottomOf"
+ curScreenKey + "Form");
            bottomOfFormElem.innerHTML = "<p>Copyright 2010, Sybase
Inc.</p>";
          }
```

```
        }
    return true;
}
```

When using the **customBeforeNavigateForward(screenKey, destScreenKey) { }** function, if you want to create your own JQuery Mobile style listview, remember that JQueryMobile does not allow duplicate ID attributes. So if there is an existing listview with the same ID attribute, you must:

1. Delete the existing listview with the same ID attribute.
2. Re-create the listview.
3. Call **refresh** for your listview.

For example:

```
//Use this method to add custom code to a forward screen transition.
If you return false, the screen
//transition will not occur.
function customBeforeNavigateForward(screenKey, destScreenKey) {



..
try {
        if (destScreenKey == 'Personal_Work_Queue') {

            //grab the results from our object query
            var message = getCurrentMessageValueCollection();
            var itemList = message.getData("PersonalWorkQueue");
            var items = itemList.getValue();
            var numOfItems = items.length;
            var i = 0;

            //iterate through the results and build our list
            var htmlOutput = '<div id="CAMSCustomViewList"><ul data-
role="listview" data-filter="true">';
            var firstOrder = '';

            while ( i < numOfItems ){
                var currItem= items[i];
                var opFlags =
currItem.getData("PersonalWorkQueue_operationFlags_attribKey").getV
alue();
                var orderId =
currItem.getData("PersonalWorkQueue_orderId_attribKey").getValue();
                var operationNumber =
currItem.getData("PersonalWorkQueue_operationNumber_attribKey").get
Value();
                var description =
currItem.getData("PersonalWorkQueue_description_attribKey").getValu
e();
                try {
                    var promDate =
currItem.getData("PersonalWorkQueue_datePromised_attribKey").getVal
ue();
                } catch (err) {
                    var promDate = "";
```

```
                }

                try {
                    var planDate =
currItem.getData("PersonalWorkQueue_dateStartPlan_attribKey").getVa
lue();
                } catch (err) {
                    var planDate = "";
                }

                var onHold =
currItem.getData("PersonalWorkQueue_onHold_attribKey").getValue();

               htmlOutput += '<li><a id ="' + currItem.getKey() + '"
class="listClick">';
               htmlOutput += '<p><b>Flags: </b>' + opFlags + '</p>';
                htmlOutput += '<p><b>Order Id: </b>' + orderId + '</
p>';
                htmlOutput += '<p><b>Operation No: </b>' +
operationNumber + '</p>';
               htmlOutput += '<p><b>Title: </b>' + description + '</
p>';
                htmlOutput += '</a></li>';

                i++;

            }

            htmlOutput += '</ul></div>';

          //append the html to the appropriate form depending on the
key
            if (destScreenKey == 'Personal_Work_Queue') {

                var listview = $('div[id="CAMSCustomViewList"]');
                //Try to remove it first if already added
                if (listview.length > 0) {
                    var ul = $(listview[0]).find('ul[data-
role="listview"]');
                    if (ul.length > 0) {
                        htmlOutput = htmlOutput.replace('<div
id="CAMSCustomViewList"><ul data-role="listview" data-
filter="true">','');
                        ul.html(htmlOutput);
                        ul.listview('refresh');
                    }
                } else {
                    $
('#Personal_Work_QueueForm').children().eq(2).hide();
                    $
('#Personal_Work_QueueForm').children().eq(1).after(htmlOutput);
                }
            }
            //add the listener based on the class added in the code
above
            $(".listClick").click(function(){
```

```
                currListDivID = $(this).parent().parent();
                $(this).parent().parent().addClass("ui-btn-active");

                //special case for bb
                navigateForward("Shop_Display",  this.id );

                if (isBlackBerry()) {
                    return;
                }
            });
        }
```

Overriding the showErrorFromNative Function

The generated JavaScript allows you to override the behavior of the showErrorFromNative function using the customBeforeReportErrorFromNative(errorString)and customAfterReportErrorFromNative(errorString) methods.

This shows an example of how to override or customize the error message based on the returned numeric error codes through `customBeforeReportErrorFromNative`.

```
function customBeforeReportErrorFromNative(errorString) {
    var errorCode = getURLParamFromNativeError("errCode",
errorString);
    // 500 and above are network errors
   if ( errorCode >= 500 )
    {
        // Could check lang global variable if so desired
        //if ( lang == ... )
        {
            // Show your own custom error message based on errorCode
           showAlertDialog("Do you have a network connection?", "My
custom error");
            // return false to by pass default behavior
            return false;
        }
    }
    return true;
}
```

Identified error scenarios include:

- Any network related errors during an online (synchronous) request contain an error code of 500 or greater (check for >= 500)
- `public static final int UNKNOWN_ERROR = 1; // "unknown error"`
- `public static final int ATTACHMENT_NOT_DOWNLOADED = 100; //"Attachment has not been downloaded"`
- `public static final int UNKNOWN_MIME_TYPE = 101; //"Unknown MIME type"`
- `public static final int FILENAME_NO_EXTENSION = 102; //"File name without extension"`

- `public static final int REQUIRED_PARAMETER_NOT_AVAILABLE = 103; //"Required parameter is not available"`
- `public static final int UNSUPPORTED_ATTACHMENT_TYPE = 105; //attachment type is not supported`
- `public static final int SSOCERT_EXCEPTION = 106; //SSO Certificate manager exception`
- `public static final int FAIL_TO_SAVE_CREDENTIAL = 107; // Fail to save credential`
- `public static final int FAIL_TO_SAVE_CERTIFICATE = 108; // Fail to save certificate`
- `public static final int DEVICE_NOT_CONNECTED = 109; // Device is not connected`

Resources.js

The resource functions allow you to access localized string resources.

Function	Description
Resources(currentLocaleName)	Creates a new resources object with the specified locale as the default locale.
resources.prototype.hasLocale(localeName)	Returns true if the locale supplied is included in the Mobile Workflow application; otherwise, returns false.
resources.prototype.getStringFromLocale(key, localeName)	Returns the localized string for the supplied key for the current locale.
resources.prototype.getString(key)	Returns the localized string for the supplied key for the specified locale.

ExternalResource.js

These functions allow you to access resources on external HTTP servers.

Function	Description

getExternalResource(url, options)	Makes a request to access resources on an external HTTP server.
	• options – a set of key/value pairs that configure the underlying request. Supported options are:
	  • **method** – one of GET, PUT, DELETE, HEAD, OPTIONS, or POST. The default is GET.
	  • HTTP and HTTPS 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.
	  • **complete** – callback for when this method completes.
	Note: An X.509 certificate blob (used in SSO) cannot be used as a client certificate for HTTPS communication. In other words, you can use only client certificates from trusted certificate authorities, which means that self-signed certificates do not work.

Function	Description
complete(resultXHR)	The complete(**resultXHR**) callback is sent when getExternalResource finishes. Its input is a resultXHR structure which closely mirrors the JavaScript `XmlHttpRequest` object. The fields/methods available on resultXHR are: • status • statusText • responesText • getReponseHeader(key) • getAllResponesHeaders() These fields and methods are not supported for **resultXHR**: • open() • setRequestHeader() • responseXML • send() • abort()

This shows an example of the UPDATE function:

```
function update() {
        // Using json to update a value
        var url = // URL of your external resource;
        var webResponse;
        var options = {
            method: "PUT",
            data: "{\"Value\":\"Value A Updated\"}",
            headers: {
                "Content-type": "application/json"
            },
            async: false,
            complete: function(response) { webResponse = response; }
        };

        getExternalResource(url, options);

        if (webResponse.status === 200)
            alert("Update successful");
        else
            alert("Update Failed");
}
```

This shows an example of the DELETE function:

```
function delete() {
      // Delete a value
```

```
        var url = // URL of your external resource;
        var webResponse;
        var options = {
            method: "DELETE",
            async: false,
            complete: function(response) { webResponse = response; }
        };

        getExternalResource(url, options);

        if (webResponse.status === 200)
            alert("Delete successful");
        else
            alert("Delete Failed");
}
```

SUPStorage.js

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).

Method	Description
SUPStorage(store)	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.
SharedStorage()	Used to construct a new shared SUP 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.
getSharedStorageKey()	Used to return the shared storage key defined for the mobile workflow application during design time. An empty string is returned if the shared storage function is disabled.
isSharedStorageEnabled()	Indicates whether the shared storage is enabled for the current workflow application.

Method	Description
SUPStorage.prototype.length()	Gets the number of available keys in this object. Retrieve the keys themselves by using **key()**. length – returns the number of key/value pairs currently present in the list
SUPStorage.prototype.key(index)	Returns the key at the supplied index. Keys are guaranteed to remain at the same index until a modification is made.
SUPStorage.prototype.getItem(key)	Retrieves the value associated with the specified key. Returns a copy of the current value associated with the given key. If the given key does not exist in the list, null is returned.
SUPStorage.prototype.setItem(key, value)	Sets the value associated with a specified key. This replaces the key's previous value, if any. If the given key does not exist in the list, a new key value is added to the list with the given key and with its value set to a copy of **value**.
SUPStorage.prototype.clear()	Removes all key/value pairs from this object.
SUPStorageException(code, message)	Constructs a new SUPStorageException object.

Calls to these methods do not trigger events.

- constructor

```
// The following script creates a 2 local storage instances with
their own domain
var store1 = new SUPStorage("mydomain");
var store2 = new SUPStorage("myotherdomain");
```

- length

```
// The following script displays the current number of elements in
the storage
var store = new SUPStorage();
alert(store.length());
```

- key(index)

```
// The following script displays the value at the provided index in
the storage
var store = new SUPStorage();
alert(store.key(2));
```

- getItem(key)

```
// The following script displays the value for the provided key
```

```
var store = new SUPStorage();
alert(store.getItem("mykey"));
```

- setItem(key, value)

```
// The following script sets a key/value pair
var store = new SUPStorage();
store.setItem("mykey", "myvalue");
```

- removeItem(key)

```
// The following script removes a key/value pair
var store = new SUPStorage();
store.removeItem("mykey");
```

- clear

```
// The following clears the storage
var store = new SUPStorage();
store.clear();
```

SUP Storage

The SUP 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 Mobile Workflow package. Values are also strings and values can be duplicated in the same Mobile Workflow package. Keys and values can contain multi-byte characters.

SUP Storage can span multiple screens in the Mobile Workflow application, 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 SUP Storage API are encrypted according to the particular platform policies:

Platform	Encryption policy
BlackBerry	PersistentStore, which adheres to the Content Protection BES IT policy
Android	Encrypted before storing into the SQLite database
iOS	Stored in SQLite Encryption Extensions database
Windows Mobile	Unencrypted SQLite—security is deferred to Afaria Security Manager

The amount of data that can be stored on the client is limited only to the available storage space on the particular platform:

Platform	Data storage
BlackBerry	Amount of free PersistentStore
iOS and Android	Amount of free file system for the SQLite database, and/or the SQLite database size limit
Windows Mobile	Amount of free file system, and/or the SQLite database size limit.

Limitations

- The amount of data that can be retrieved using the SUPStorage API, and returned to the JavaScript space, is limited to the JavaScript size limitation as established for each platform. See the topic *Attachment Viewer and Image Limitations*.
- 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 SUP storage is tied to a Mobile Workflow package. When the Mobile Workflow package is uninstalled, the corresponding SUP storage for the Mobile Workflow package is removed immediately.
- Items stored using the SUPStorage API are persisted, and therefore, survive soft device resets.
- SUP Storage persists through invocations of the Mobile Workflow application.
- The SUPStorage API does not restrict reading or writing of the storage data from different domains. For example, if a Mobile Workflow application 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 Mobile Workflow applications with a shared storage key assigned share the storage with other Mobile Workflows that have the same storage key assigned.

- When the last Mobile Workflow application 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 Workflow 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 Workflow application.

Function	Description
getCurrentLocale()	Returns the current locale for the device.
getLocalizedDateTime(date)	Returns the specified Date as a locale-specific display as a datetime using native functionality.
getLocalizedDate(date)	Returns the specified Date as a locale-specific display as a date using native functionality.
getLocalizedTime(date)	Returns the specified Date as a locale-specific display as a time using native functionality.
convertUtcToLocalTime(date)	Converts the specified Date to the device’s local time.
convertLocalTimeToUtc(date)	Converts the specified Date to UTC time.
getOffsetFromUTC(date)	Given the specified Date, queries the device's OS to determine the total offset (difference) between the given “local” time and UTC, including any daylight savings time (DST) offsets if applicable. Returns a signed integer representing this GMT offset in minutes. For example, if the device was in the London timezone (GMT+1) and it is DST is currently in effect, the function would return “120”—60 minutes normal offset plus 60 minutes for its DST offset.
isDstActiveAtGivenTime(date)	Given the specified Date, queries the device's OS to determine if DST rules are in effect for the current timezone at the time specified in the Date object.
getDstOffsetAtGivenTimeInMinutes(date)	Given the specified Date, queries the device's OS to determine the DST offset for the current timezone at the time specified in the Date object.

Function	Description
getTimezoneId()	Returns the current timezone's standard name.
getUsesDST()	Determines whether or not the device's current timezone resides in a timezone that practices daylight savings.

WorkflowMessage.js

Use these functions to access Workflow Message resources.

Function	Description
MessageValue()	Constructs a new MessageValue object.
MessageValue.prototype.getKey()	Returns the key of the given MessageValue object
MessageValue.prototype.setKey(key)	Sets the key of the given MessageValue object
MessageValue.prototype.getValue()	Returns the value of the given MessageValue object
MessageValue.prototype.setValue(value)	Sets the value of the given MessageValue object
MessageValue.prototype.getType()	Returns the type of the given MessageValue object
MessageValue.prototype.setType(type)	Sets the type of the given MessageValue object
MessageValueCollection()	Constructs a new MessageValueCollection object
MessageValueCollection.prototype.getKey()	Returns the key of the given MessageValueCollection object
MessageValueCollection.prototype.setKey(key)	Sets the key of the given MessageValueCollection object
MessageValueCollection.prototype.getState()	Returns the state of the given MessageValueCollection object.
MessageValueCollection.prototype.setState(state)	Sets the state of the given MessageValueCollection object

Function	Description
MessageValueCollection.prototype.getParent()	Returns the parent key of the given MessageValueCollection object
MessageValueCollection.prototype.setParent(parent)	Sets the parent key of the given MessageValueCollection object.
MessageValueCollection.prototype.getParentValue()	Returns the parent object of the given MessageValueCollection object
MessageValueCollection.prototype.setParentValue(parentValue)	Sets the parent object of the given MessageValueCollection object. Does not change the actual parenting.
MessageValueCollection.prototype.add(key, value)	Adds a new value to the given MessageValueCollection
MessageValueCollection.prototype.clear()	Removes all value from the given MessageValueCollection
MessageValueCollection.prototype.getData(key)	Returns the value corresponding to the specified key for the given MessageValueCollection
MessageValueCollection.prototype.remove(key)	Removes the value corresponding to the specified key for the given MessageValueCollection
MessageValueCollection.prototype.getCount()	Returns the number of values in the given MessageValueCollection
MessageValueCollection.prototype.getKeys()	Returns an array of the keys in the given MessageValueCollection.
MessageValueCollection.prototype.getValues()	Returns an array of the values in the given MessageValueCollection.
WorkflowMessage(message)	Constructs a new WorkflowMessage object with the specified contents (represented as a string).
WorkflowMessage.prototype.getHeader()	Returns the header of the given WorkflowMessage.
WorkflowMessage.prototype.setHeader(header)	Sets the header of the given WorkflowMessage.
WorkflowMessage.prototype.getRequestAction()	Returns the request action of the given WorkflowMessage.

Function	Description
WorkflowMessage.prototype.setRequestAction(requestAction)	Sets the request action of the given WorkflowMessage.
WorkflowMessage.prototype.getValues()	Gets the values in the given WorkflowMessage.
WorkflowMessage.prototype.getWorkflowScreen()	Returns the workflow screen of the given WorkflowMessage.
WorkflowMessage.prototype.setWorkflowScreen(workflowScreen)	Sets the workflow screen of the given WorkflowMessage.
WorkflowMessage.prototype.createFromString(messageAsString)	Updates the contents of the given WorkflowMessage object from the given string.
WorkflowMessage.prototype.serializeToString()	Returns a string representation of the given WorkflowMessage.
WorkflowMessage.prototype.updateValues(sourceValues, listViewValuesKey)	Updates the values of the given WorkflowMessage.

Using Third-Party JavaScript Files

To include your own files in the container application, copy them into the appropriate place in the Generated Workflows folder.

To load external JavaScript and CSS files dynamically, copy the relevant third-party JavaScript and CSS files to the `Generated Workflow \<Workflow_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 Mobile Workflow manifest.xml and .zip files automatically when the Mobile Workflow package is re-generated.

Repackaging Mobile Workflow Package Files

After modifying the `Custom.js` file, you must redeploy the Mobile Workflow package to Unwired Server.

1. Save and close the modified files after adding your custom code.
2. In WorkSpace Navigator, right-click the `<mobile_workflow_name>.xbw` file and select **Generate Mobile Workflow Package**.

3. In the Mobile Workflow Package Generation wizard, select **Deploy to an Unwired Server**, and select the Unwired Server connection profile.
4. In Deploy Mode, select either:
   - New – generates and deploys the mobile workflow package and its files for the first time.
   - Update – updates any pre-existing Mobile Workflow package in-place, preserving associated assignments.
   - Replace – removes any pre-existing Mobile Workflow package and notifications before deploying.
5. Click **Finish**.

Common Customizations

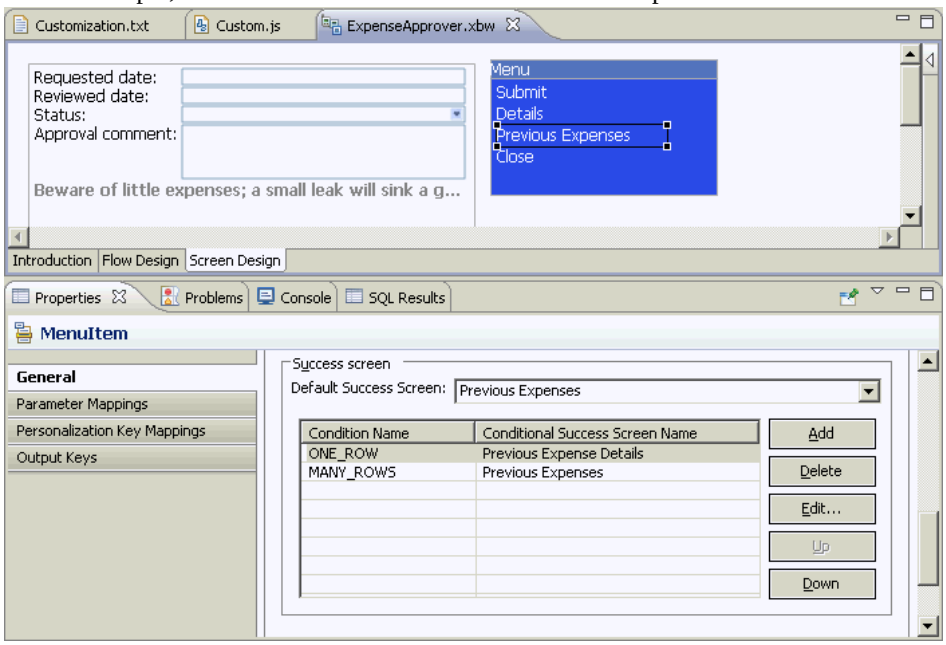
Implementing Conditional Navigation

Conditional navigation allows you to implement a custom function that allows you to override navigation behavior between screens.

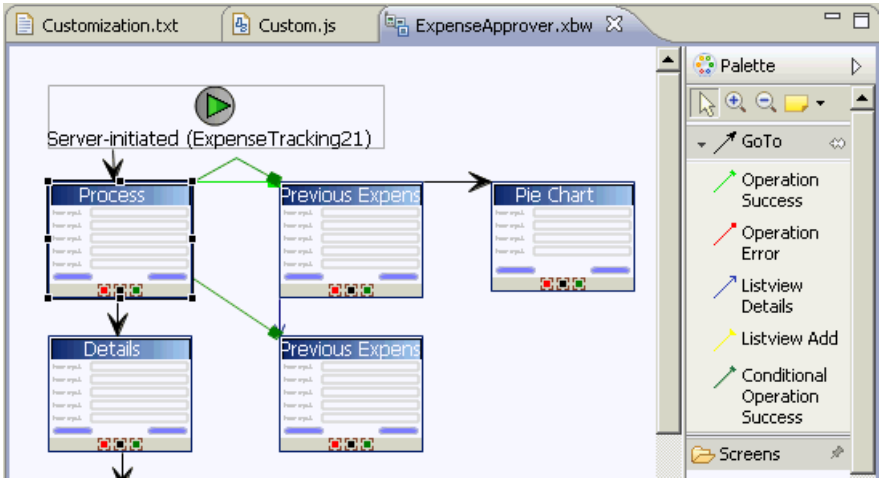
This procedure gives an example of how you can use conditional navigation to skip a screen.

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.



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.

```
//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 Mobile Workflow Package Generation wizard to re-generate the Mobile Workflow Package with a new `workflow_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.

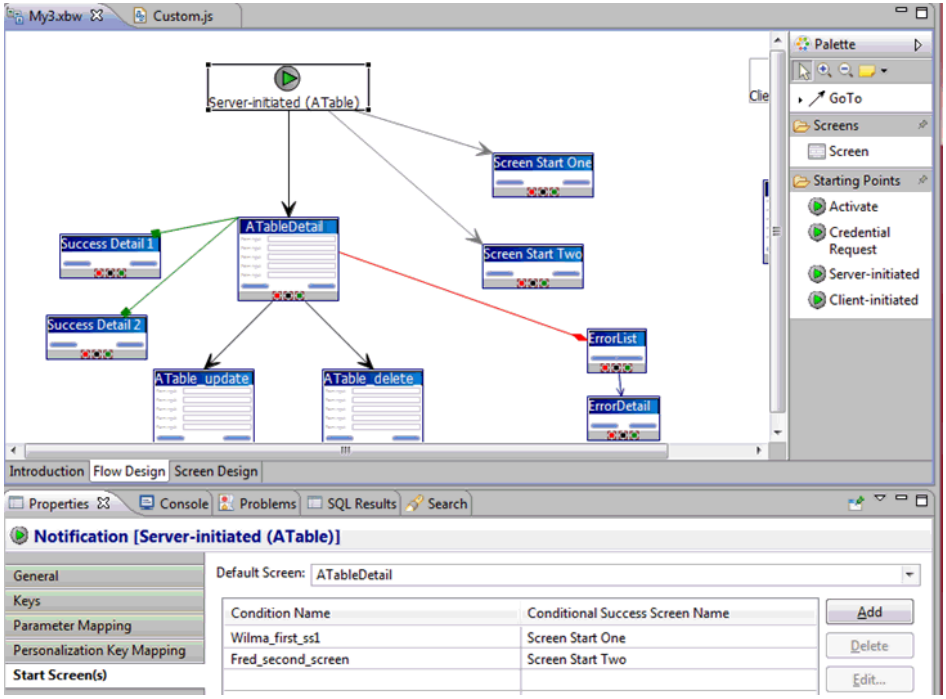
Like the conditional success navigation feature, there is a table of condition names with the matching Start screen. If all of the conditions are evaluated as false (or if they are absent), the default navigation is executed.

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:

```
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 Mobile Workflow package.

   When you regenerate the Mobile Workflow package, the `Workflow.js` file is regenerated. The conditional start screen method is shown in the `Workflow.js` file similar to this:

```
function customNavigationEntry() {
    this.condition;
    this.screen;
}
function customNavigationEntry( a_condition, a_screen ) {
    this.condition = a_condition;
    this.screen = a_screen;
}

/**
 * For the specific pair - screen named 'currentScreenKey' and the
action 'actionName', return
 * the list of custom navigation condition-names and their
destination screens.
 */
function getCustomNavigations( currentScreenKey, actionName )  {
    var customNavigations = new Array();
    if((currentScreenKey === SERVERINITIATEDFLAG) && (actionName
=== '')) {
    customNavigations[0] = new
customNavigationEntry( 'Wilma_first_ss1',
'Screen_Start_One' );
    customNavigations[1] = new
customNavigationEntry( 'Fred_second_screen',
'Screen_Start_Two' );
return customNavigations;
    }
```

```
    return customNavigations;
}
```

Clearing the Contents of the Signature Control

Add JavaScript to clear the contents of a signature control.

1. Use the Mobile Workflow Package Generation wizard to generate the Mobile Workflow package and its files.

   When the Mobile Workflow package is generated, the Custom.js file is generated if not already present in the project. The Custom.js file is located in Generated Workflows \<workflow_project_name>\html\js.

2. Open the Custom.js file and add your JavaScript code to the click event of a menu or button.

   For example:

```
function customAfterMenuItemClick(screen, menuItem) {
    if (menuItem === "Clear_Signature") {
        $.data(document.getElementById('sigKey'),
'signature').clearSignature();
    }
}
```

3. Save and close the Custom.js file.
4. Re-generate the Mobile Workflow package and deploy it to Unwired Server.

Install and Configure the Hybrid Web Container On the Device

To enable deploying mobile workflow packages to a device, you must download, install, and configure the Mobile Workflow container on the device.

Deploy the Mobile Workflow container to devices and register the devices with Unwired Server. You can use Afaria® to install the container on devices for enterprise deployment. For information on setting up an Afaria environment, see *System Administration > Device and Application Provisioning Overview > Provisioning With Afaria*.

See the configuration procedure for your device type.

Preparing Android Devices for the Mobile Workflow Package

Install the Mobile Workflow container on the Android device using the Android SDK.
In the Settings for your Android device, disable all keyboards except the Android keyboard.

Installing Sybase Mobile Workflow on Android Devices

Use the Android SDK Manager to install Sybase Mobile Workflow application files.

To install Sybase Mobile Workflow on your Android device:

1. Connect the device.
2. Install the Android SDK.
3. Run `platform-tools\adb` and install *`UnwiredPlatform_InstallDir`*`\UnwiredPlatform\MobileSDK<`*`version`*`>\HybridWeb\Android\HybridWebContainer.apk`.

   For example:

   ```
   C:\Android\android-sdk\platform-tools\adb install ^
   C:\Sybase\UnwiredPlatform\MobileSDK\HybridWeb\Android
   \HybridWebContainer.apk
   ```

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 Sybase Unwired Platform Mobile SDK installation.

Prerequisites

- Install the Android SDK version 2.2, API Level 8. You can get the Android SDK at *http://developer.android.com/sdk/index.html*.
- If you are developing in Eclipse, install the ADT Plug-in for Eclipse.

Task

This example uses Eclipse as the development environment, but you can use any development environment.

1. Open Eclipse and select **File > Import**.
2. Expand the **General** folder, choose **Existing Projects into Workspace**, and click **Next**.
3. Choose **Select archive file**, browse to `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK<version>\HybridWeb\Android`, and select `template.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 Sybase Unwired Platform.

1. Open a command prompt and navigate to the base directory of the Hybrid Web Container project.
2. Run either the **ant debug** or **ant release** command, depending on whether you want to debug or release the Hybrid Web Container.

   You can download Apache Ant from *http://ant.apache.org/bindownload.cgi*, if necessary.

   A file named either `HybridWebContainer-debug.apk` or `HybridWebContainer-release-unsigned.apk` (depending on the command you used) is added to the `bin` folder. If a file already exists with that name, it is overwritten.
3. Use Android Debug Bridge (ADB), which is included in the Android SDK installation, to install the `.apk` to the emulator.

   a) Launch an Android Virtual Device (AVD) that does not have the Hybrid Web Container installed (or uninstall it if it is installed).
   b) In the Command Prompt window, navigate to the folder that contains the `adb.exe` file, which should be in the `…/android-sdk/platform-tools/ folder`.
   c) Execute: **adb install <path>**, where *<path>* is the full filepath to the `HybridWebContainer.apk` file.

Configuring the Android Emulator

Configure an Android emulator for testing a Sybase Mobile Workflow package.

Note: The steps or interface may be different depending on which Android SDK version you are using.

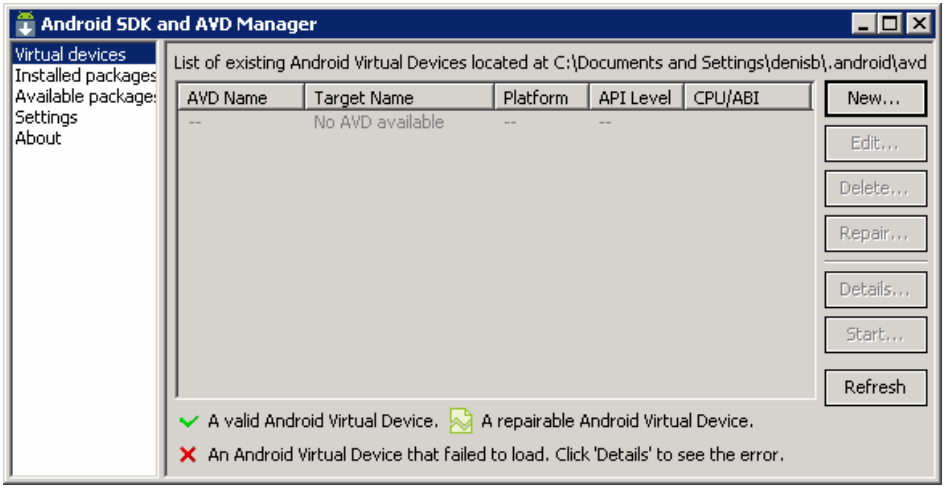
1. Install the Android SDK.

   Go to *http://developer.android.com/sdk/* to download and install the Android SDK. Click **Install** and accept the default values. Follow the instructions on the Android page, with these exceptions:
   - See *Supported Hardware and Software* for the most current version information for mobile device platforms and third-party development environments.
   - When specifying the install location, consider choosing a path that does not contain spaces, such as `C:\Android\android-sdk`. Some versions of the Android SDK

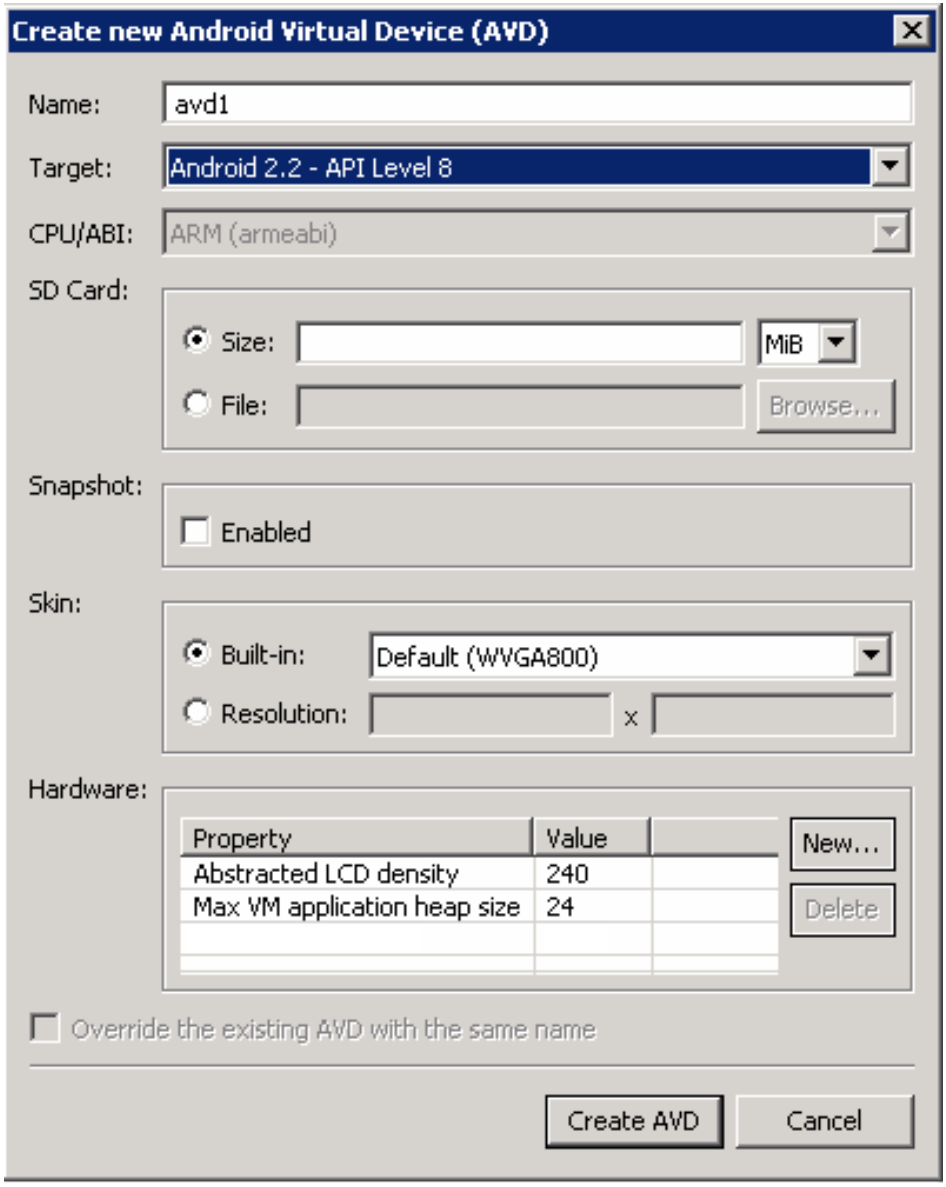
do not work correctly when installed in the default `drive:\ Program Files` location.

- If the Android installer stops with a message that the required Java JDK is not found on your system (even when the JDK is installed), try clicking **Back** and then **Next**, one or more times, until the installer detects the JDK.

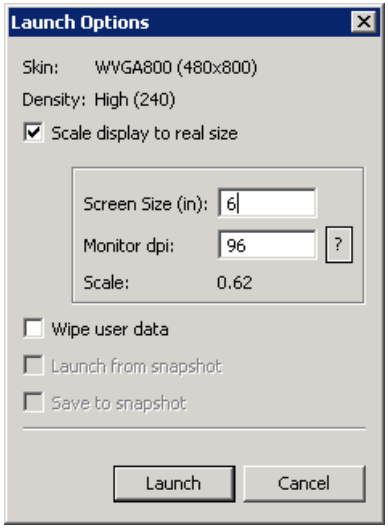
2. Install the SDK Platform-tools:
   a) Run the Android SDK Manager
   b) Select these options:
      - In Tools, **Android SDK Platform-tools**.
      - The version of Android whose emulators you want to use, and that Unwired Platform supports.
   c) Click the **Install** button.
3. Click **Start Programs** > **Android SDK Tooks** > **AVD Manager**.



4. Add a device:
   a) In the Android AVD Manager, click **New**.
   b) In the Create new Android Virtual Device window, enter a name.
   c) For the target, select a supported Android version.
   d) Set any other available options you want, then click **Create AVD**.



5. Select the new virtual device and click **Start**.
6. In Launch Options, optionally modify the default display scaling, then click **Launch**.



7. When the Android screen finishes loading, open a command prompt and run the *Android_InstallDir*`\android-sdk\platform-tools\adb.exe` command to install `HybridWebContainer.apk` to the emulator:
   For example:

```
C:\Android\android-sdk\platform-tools\adb install ^
C:\Sybase\UnwiredPlatform\MobileSDK213\HybridWeb\Android
\HybridWebContainer.apk
```

Preparing iOS Devices for the Mobile Workflow Package

Install the Mobile Workflow client on the device using the App Store, or use the source code provided for the Mobile Workflow container to deploy to the iOS simulator from the Xcode project.

Complete these prerequisites before provisioning the Mobile Workflow application:

- Determine your security policy – Unwired Platform provides a single administration console, Sybase Control Center, which allows you to centrally manage, secure, and deploy applications and devices. Device user involvement is not required and you can maintain the authorization methods you already have in place. See *Security > Device Security*.
- Register each application connection using Sybase Control Center – application connections pair an application with a device. See *Sybase Control Center for Sybase Unwired Platform* documentation.

Apple Push Notification Service

Sybase Unwired Platform provides support for Apple Push Notification Service by pushing notifications to Mobile Workflow applications when the Mobile Workflow application is offline.

With APNS, each device establishes encrypted IP connections to the service and receives notifications about availability of new items awaiting retrieval on Unwired 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 *System Administration > Device and Application Provisioning Overview*.

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 Mobile Workflow is assigned to the device.
  - A Mobile Workflow DCN message is sent to Unwired Server, targeting a particular user and the Mobile Workflow is not running.

If you want to use APNs for the Mobile Workflow application, you can:

- Use the `.p12` located in `<Unwired_Platform_InstallDir>\UnwiredPlatform\Servers\MessagingServer\bin\` with the pre-built Workflow application that is available from the App Store.

  These .p12 certificates are provided:
  - MobileWorkflowPushDistCert.p12 – for Sybase Mobile Workflow – Free, 2.0, or 2.0.1
  - MobileWorkflow21PushDistCert.p12 – for Sybase Mobile Workflow 2.1
  - MobileWorkflow212PushDistCert.p12 – for Sybase Mobile Workflow 2.1.2 and later
- Use the Apple Provisioning Portal to create your own .p12 certificate if you build your own Mobile Workflow application using the source code included in `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK<version>\HybridWeb\iOS`.

After creating the .p12 certificate, you must configure the APNs settings in Sybase Control Center.

Provisioning iOS Devices

Use this procedure to provision your iOS device for APNs if you build your own Mobile Workflow application using the source code provided in *`UnwiredPlatform_InstallDir`*`\UnwiredPlatform`

```
\MobileSDK<version>\HybridWeb\iOS\MobileWorkflow-
<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 Mobile Workflow applications to devices, you must create a certificate in your developer account and provision your device. See the *Apple Local and Push Notification Programming Guide* at *http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ProvisioningDevelopment/ProvisioningDevelopment.html* for details.
2. Use the iPhone Provisioning Portal at *http://developer.apple.com/devcenter/ios/index.action* to create the SSL certificate and Keys. Configure the certificate to enable for Apple Push Notification service.
3. On your Mac, launch the Keychain Access program. This is located in the Utilities folder.
   a) In Keychain Access, select **Keychain Access > Certificate Assistant > Request a Certificate from Certificate Authority**.
   b) In the Certificate Information window, enter the information. Use a unique Common Name.

      Note: Make sure you use a different Common name than a development certificate you already have. This creates a private key with the name you enter here.

      A certificate request is created and saved in the Desktop folder by default.
4. In the Apple Provisioning Portal, continue with the App ID provisioning and browse to the certificate request file created in Keychain Access in the previous step, then click **Generate**.
5. Click **Continue**.
6. Click **Download Now**.
   The certificate is downloaded onto your Mac and 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 Mobile Workflow application.
9. In XCode, open the MobileWorkflow project.

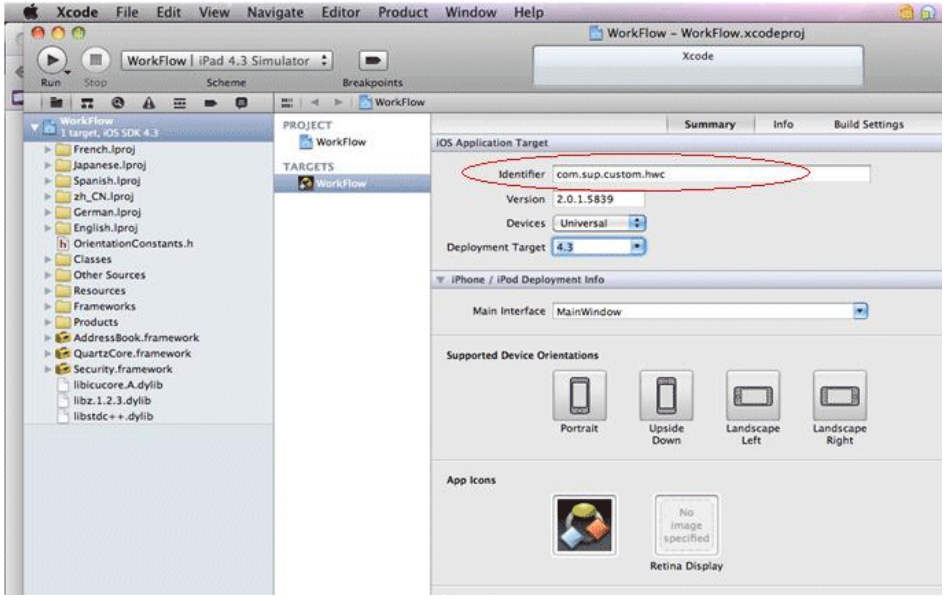
   Note: Note the product name. This is used to configure the mobile workflow in Sybase Control Center and corresponds to the Application Name property in SCC. By default, the

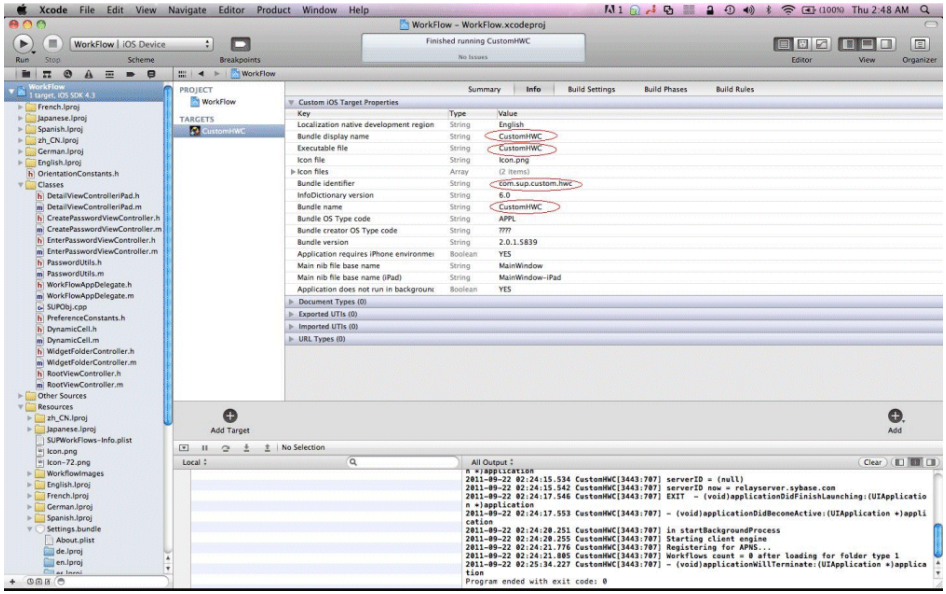
application name is HWC. This needs to be configured in the properties for the target. There is a 15-character limit for the product name.

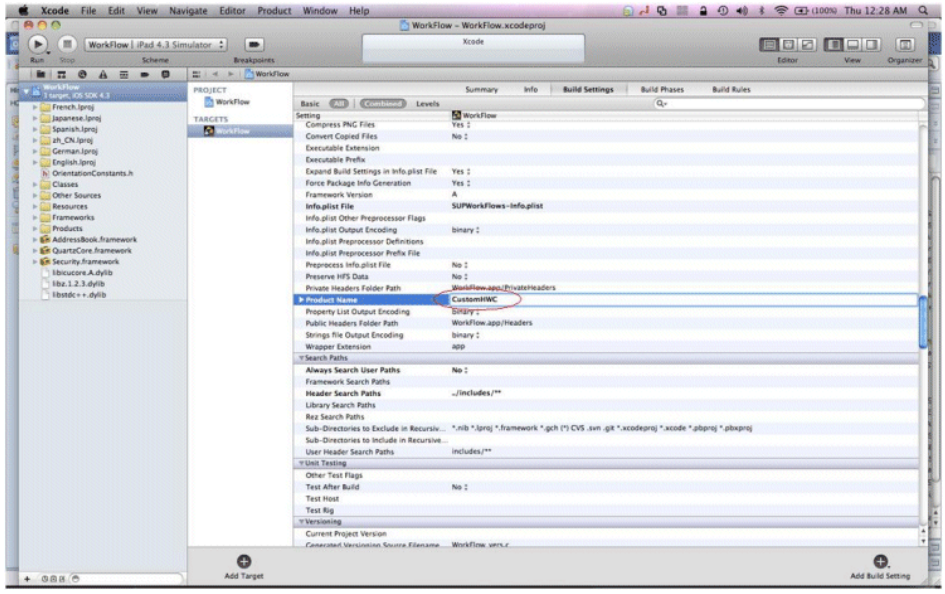
10. Change AppName and AppId in the `Branding.strings` file for the necessary language resources.

This file is available under the **Resources** folder of the Workflow XCode project.

Note: The Bundle Identifier must correspond to the Bundle identifier specified in the App ID. By default, the project comes with a bundle ID of com.sybase.mobileworkflow<xxx>. Change it to something unique.







11. Copy the exported `certificate_name.p12` certificate to the machine where Sybase Control Center is installed and follow the instructions in Configuring Apple Push Settings for the Mobile Workflow Application 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 Mobile Workflow Application

The certificate that was exported from the keychain corresponding to Apple Push settings must be configured with the correct application name in SCC.

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. After making these changes, you must restart your machine.

---

1. In the left navigation pane, expand the **Servers** folder and select a server.
2. Select **Server Configuration**.
3. In the Messaging tab, select **Apple Push Configuration**.
4. Click **New**.
5. Enter the **Application name**. Make sure this name matches the AppId entered in the `Branding.strings` file.

   Note: The application name is **HWC**.

   ---

6. Select **Use existing certificate** to use a security certificate file that already exists on the server.

   When you select this option, the list of available certificates appears in the **Certificate name** menu.

   a) Select the desired certificate from the list, for example, `MobileWorkflow212PushDistCert.p12`, which is located in `<Unwired_Platform_InstallDir>\UnwiredPlatform\Servers\MessagingServer\bin`.

   b) Enter and confirm the certificate password.

   Note: If you are using the **Sybase Mobile Workflow <version>** version of the Workflow from the App Store, enter the password: `M0bileW0rkfl0w;SUP`.

   ---

7. Select **Use new certificate** to create a new certificate on the server.

   a) Enter a name for the new certificate.

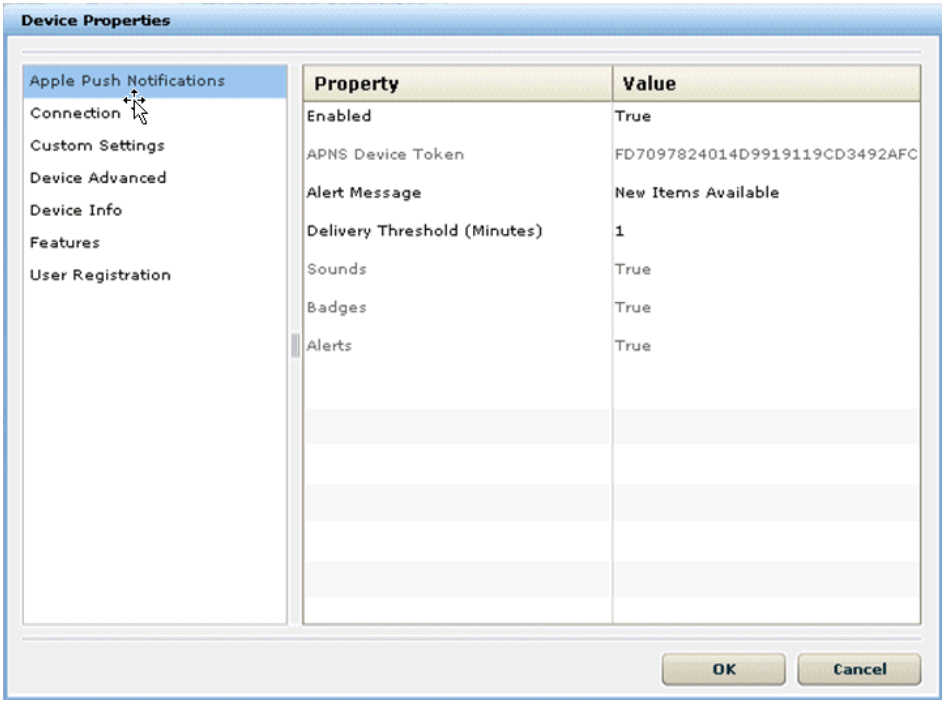
   b) Specify a Base64-encoded string by choosing one of these:

      - **Browse from file** – select a security certificate file on the server that contains the Base64-encoded string.
      - **Base64-encoded string** – manually enter the Base64-encoded string.

   c) If you selected a file from the server for the Base64-encoded string, you can overwrite the existing certificate file with the details you specify during new certificate creation. To do so, select the box adjacent to **Overwrite existing certificate**.

d) Enter and confirm the certificate password.

8. Click **OK**.
9. You can verify that the device is configured for APNS correctly by verifying that the device token has been passed from the workflow application after the workflow application runs once on the device.



Use the **Send a Notification** tool inside the Mobile Workflow Forms editor to send a test notification.

APNS Trace Files

`<UnwiredPlatform_InstallDir>\Unwired Server\log\trace\APNSProvider` has tracing information.

Increase the trace level to **Debug** in Sybase Control Center.

Apple Push Notification Properties

Apple push notification properties allow iOS users to install client software on their devices. This process requires you to create a different e-mail activation message using the appropriate push notification properties.

- **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 a 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

Installing the Mobile Workflow Application on Your iOS Device

How you install the Mobile Workflow application 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 Mobile Workflow application. The possible methods include:

- Downloading and installing the free version of the Mobile Workflow application 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 Sybase Unwired Platform installation.

Building the Mobile Workflow Container Using the Provided iOS Source Code

The mobile workflow container referenced is a sample container. You can use the provided source code in Xcode to build your own customized user interface and configure other resources.

Prerequisites

- Register the device in Sybase 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 Sybase Unwired 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 `MobileWorkflow-version.tar.gz` archive from *`UnwiredPlatform_InstallDir`*`\UnwiredPlatform\MobileSDK213\HybridWeb\iOS\` to a location on your Mac.

   In the archive file name, *version* is the current Unwired Platform version number. For example, `MobileWorkflow-2.1.3.tar.gz`.
3. Unpack `MobileWorkflowversion.tar.gz`.
   The extraction creates a `Workflow` directory.
4. In the `Workflow` directory, double-click `WorkFlow.xcodeproj` to open it in the Xcode IDE.
5. If necessary, click **Project** > **Edit Active Target** > **ProjectName** > **General** and add these files from the SDK to the project:
   - `Security.framework`
   - `AddressBook.framework`
   - `QuartzCore.framework`
   - `CoreFoundation.framework`
   - `libicucore.A.dylib`
   - `libz.1.2.5.dylib`
   - `libstdc++.dylib`
6. In Xcode, click **Build** > **Build** to build the project.

Installing the Mobile Workflow Container from the Apple App Store

Install the Mobile Workflow container from the Apple App Store.

This is a free version of the Sybase Mobile Workflow and should not be used for enterprise deployment.

1. On your device, on the iOS home page, tap **App Store**.
2. Search for **Sybase Mobile Workflow**.
3. In the search results find the version of the Sybase Mobile Workflow container to install and click **Free**.
4. Tap **Install** to download the application.
5. In **Settings** > **Workflow**<***version***>, for Connection Info, enter:

   a) In Connection Info, enter:
      - Server Name – the machine that hosts the server where the mobile application project is deployed.
      - Server Port – Unwired Server port number. The default is 5001.
      - Farm ID – the farm ID you entered when you registered the application connection in Sybase Control Center.
      - User Name – the user you registered in Sybase Control Center.
      - Activation Code – the activation code for the user, for example, `123`.
      - Protocol – HTTP or HTTPS. The protocol with which to connect to the Relay Server or the reverse proxy server. The default is HTTP.
      - Password – enter your password so that the container registers using the automatic registration option.

        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.
      - (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 System Administration > System Reference > Application Connection Properties > Device Advanced Properties*.
6. Scroll to the page that contains the **Workflow** icon, then tap to launch.
7. Enter your personal identification number (PIN). Choose the number that you need to enter to start the Mobile Workflow application. This PIN is a security measure to safeguard your company's data.
   - 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 **Workflows** page appears.

8. Click the **Messages** tab bar, then tap **Messages** to view the Workflows.
9. (Optional) If instructed by your system administrator, enable notifications on your device.

Installing the Mobile Workflow Application Using iTunes

Install the Mobile Workflow application using iTunes.

1. Launch iTunes.
2. Download the application from your corporate network to your Applications library.
3. Sync the Mobile Workflow application to your Apple mobile device.
4. Specify the connection settings in **Settings > Workflow**.

Preparing BlackBerry Devices for the Mobile Workflow Package

Install the Mobile Workflow container on the BlackBerry device using BlackBerry Desktop Manager.

Prerequisites

For prerequisites and complete information about provisioning BlackBerry devices see *BES Provisioning for BlackBerry* in *System Administration > Device Provisioning*.

Task

1. Connect the BlackBerry device to the computer that contains the Mobile Workflow container for BlackBerry.
2. Run the BlackBerry Desktop Manager following the instructions in the RIM documentation.
3. In the BlackBerry Desktop Software, select **Application Loader**.
4. Under Add/Remove Applications, select **Start**.
5. Browse to the location on your local machine or network that contains the mobile workflow `.cod` and `.alx` container files, `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK<version>\HybridWeb\BB`.
   - `CommonClientLib`
   - `MessagingClientLib`
   - `MocaClientLib`
   - `Workflow`
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 Mobile Workflow Container on BlackBerry Devices Over the Air

Your system administrator must provide the appropriate information for installing the Mobile Workflow container over the air, and the BlackBerry Exchange Server (BES) must be available.

Note: For complete information about provisioning BlackBerry devices see *BES Provisioning for BlackBerry* in *System Administration > Device Provisioning*.

The administrator stages the OTA files in a Web-accessible location and notifies BlackBerry device users via 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 `SybaseMobileWorkflow.jad` file. This single JAD and associated files for this type of deployment are located in `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK<version>\HybridWeb\BB\OTA`.

Configuring the BlackBerry Simulator for Mobile Workflow Packages

Copy the `.cod` files 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 `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK<version>\HybridWeb\BB`.
4. Select the required .cod files, then click **OK**. These include:
   - `CommonClientLib.cod` – shared code that can be used by native Sybase Unwired Platform BlackBerry applications.
   - `MessagingClientLib.cod` – the main Messaging library, shared code that can be used by native SUP BlackBerry applications.
   - `MocaClientLib.cod` – messaging library.
   - `Workflow.cod` – the main Mobile Workflow application and Sybase Settings, where the user enters the server connection information.

Installing the Mobile Workflow Container on Windows Mobile Devices

Install and configure the Hybrid Web Container required to prepare a Windows Mobile device to run Mobile Workflow packages.

1. Navigate to `<UnwiredPlatform_InstallDir>\UnwiredPlatform\MobileSDK\HybridWeb\WM`.
2. Copy the Windows Mobile Professional device file, `SybaseMobileWorkflow.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 SybaseMobileWorkflow.`cab` file from the Windows Mobile device. This installs the container.
6. In Programs, click the Workflow Settings icon to open the Mobile Workflow settings connection screen.
7. In the Workflow Settings Connection screen, enter the connection settings. These settings should match the values you used when you registered the device in Sybase Control Center.

   Note: Select the right arrow icon ( ) to view the container log. This is useful for checking the connection, or retrieving other debugging information.

8. From the Start menu, select Sybase Mobile Workflow.
9. Select a Mobile Workflow package from the list of available packages.

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 Mobile Workflow application.

1. Click the **Workflows** 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. Enter the settings for the Mobile Workflow application:
   - Server Name – the machine that hosts the server where the mobile application project is deployed.
   - Server Port – Unwired Server port number. The default is 5001.
   - Farm ID – the farm ID you entered when you registered the application connection in Sybase Control Center.
   - User Name – the user you registered in Sybase Control Center.
   - Activation Code – the activation code for the user, for example, `123`.
   - Protocol – HTTP or HTTPS. The protocol with which to connect to the Relay Server or the reverse proxy server. The default is HTTP.
   - Password – enter your password so that the container registers using the automatic registration option.

     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.
   - (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 System Administration > System Reference > Application Connection Properties > Device Advanced Properties*.

   Select **Save** to save the settings.
4. Start the Mobile Workflow application, then view the settings log to verify that the connection is active.

   From the Mobile Workflow application, tap **Settings > Show Log**.

Configuring BlackBerry Connection Settings

Configure the connection settings for the Mobile Workflow application.

1. Click the **Workflow** icon on the applications screen, then press the **Menu** key and select **Settings**.
2. Enter the settings for the Mobile Workflow application:
   - Server Name – the machine that hosts the server where the mobile application project is deployed.
   - Server Port – Unwired Server port number. The default is 5001.
   - Farm ID – the farm ID you entered when you registered the device in Sybase Control Center.
   - User Name – the user you registered in Sybase Control Center.
   - Activation Code – the activation code for the user, for example, `123`.

- HTTP – the protocol with which to connect to the Relay Server or the reverse proxy server.
- Enable Automatic Registration – when you select this option, the Registration 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.

- URL Suffix (Optional) – the 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*.

3. Select **Menu > Save** to save the settings.
4. Start the Mobile Workflow application, then view the settings log to verify that the connection is active.

   In Workflows, select **Settings**. On the Connection settings screen, select **Show Log**.

Configuring iOS Connection Settings

Configure the settings for the Mobile Workflow application.

1. Go to the device Settings screen and click **WorkFlows**.
2. In the basic authentication screen, enter the user name and password if you are prompted.
3. Enter the settings for the Mobile Workflow application:
   - Server Name – the machine that hosts the server where the mobile application project is deployed.
   - Server Port – Unwired Server port number. The default is 5001.
   - Farm ID – the farm ID you entered when you registered the application connection in Sybase Control Center.
   - User Name – the user you registered in Sybase Control Center.
   - Activation Code – the activation code for the user, for example, `123`.
   - Protocol – HTTP or HTTPS. The protocol with which to connect to the Relay Server or the reverse proxy server. The default is HTTP.
   - Password – enter your password so that the container registers using the automatic registration option.

     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.

   - (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 System*

Administration > System Reference > Application Connection Properties > Device Advanced Properties.

Configuring Windows Mobile Connection Settings

Configure the connection settings.

Prerequisites

Install the Mobile Workflow Container `CAB` file.

Task

1. Select **Start > Programs > Sybase Settings**.
2. Click **Connection**.
3. 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 – Unwired Server port number. The default is 5001.
   - Farm ID – the farm ID you entered when you registered the device in Sybase Control Center.
   - User Name – the user you registered in Sybase Control Center.
   - Activation Code – the activation code for the user, for example, `123`.
   - HTTP – the protocol with which to connect to the Relay Server or the reverse proxy server.
   - Enable Automatic Registration – when you select this option, the Registration 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.
   - URL Suffix (Optional) – the 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*.

   Note: The URL Suffix setting is in Advanced settings.
4. Click **Done**.
5. Start the Mobile Workflow application, then view the settings log to verify that the connection is active.

   Select tap **Start > Programs > Sybase Settings > 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

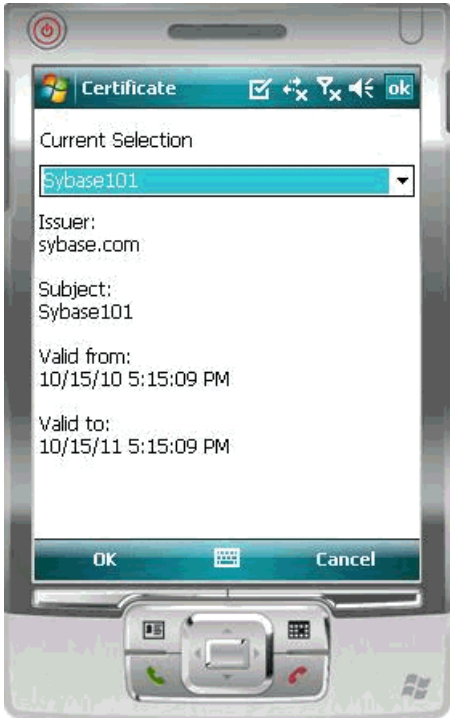
Select an X.509 certificate to use for Mobile Workflow application user authentication.

Prerequisites

1. Create a Mobile Workflow application that prompts the user to specify a certificate as credentials.
2. Package and assign the Mobile Workflow application to a Windows Mobile device user.

Task

1. From **Inbox** > **Workflows**, select the Mobile Workflow.
2. Select the **Specify Certificate Credentials** menu item from the Certificate Picker.
3. Select the certificate (for example, Sybase101) and continue with the Mobile Workflow.



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 Mobile Workflow application user authentication.

Prerequisites

1. Create a Mobile Workflow application that prompts the user to specify a certificate as credentials.
2. Package and assign the Mobile Workflow application to an Android device user.

Task

1. On the Android device or emulator, in applications, click **Workflow**.
2. Select the Mobile Workflow 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 intstall 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

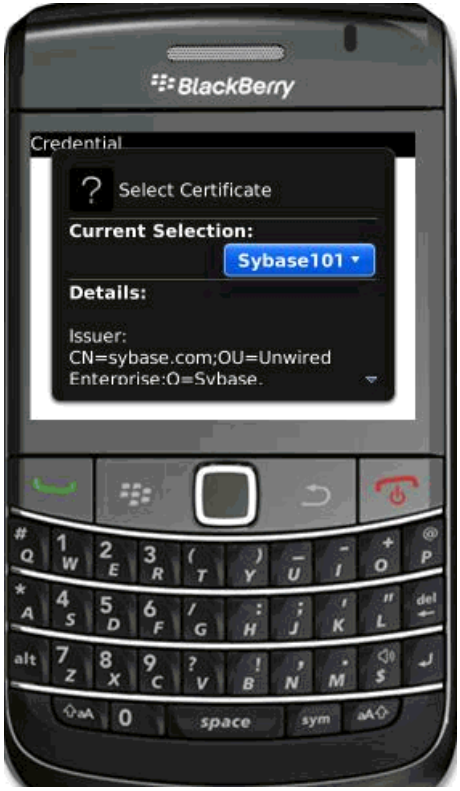
Select an X.509 certificate to use for workflow application user authentication.

Prerequisites

1. Create a Mobile Workflow application that prompts the user to specify a certificate as credentials.
2. Package and assign the Mobile Workflow application to a BlackBerry device user.

Task

1. From **Inbox** > **Workflows**, select the mobile workflow.
2. From the Certificate Picker, select the **Specify Certificate Credentials** menu item.
3. Select the certificate (for example, Sybase101) and continue with the workflow application.



Installing X.509 Certificates on iOS Devices

Add an authentication screen to the workflow client from which you can authenticate with a generated X.509 certificate instead of a user name and password combination.

1. Copy the X.509 certificate used for authentication into a directory on the same host as Unwired Server. For example, `c:\certs`.
2. Create a registry string value on Unwired Server at `HKLM\Software\Sybase\Sybase Messaging Server\CertificateLocation` and populate it with the path. For example, `c:\certs`.
3. Name the X.509 certificate file as `domain_user.p12`, where *domain* is the Unwired Server domain and *user* is the certificate user. The user must have read permission for `.p12` file.
4. The system administrator must ensure the specified domain\user has “log on as batch job” permission on the Windows machine on which Unwired Server runs:
   a) Double-click **Control Panel** > **Administrative Tools** > **Local Security Policies**.
   b) Expand **Local Policies** and select **User Rights Assignment**.
   c) Right-click **Log on as a batch job** and select **Properties**.

d) Select **Add User or Group** and add the domain\user.

5. The account under which Unwired Server runs must have adequate permissions to impersonate the domain\user, for example, the Administrator account for the domain.

Testing X.509 Certificates on iOS Devices and Simulators

Select an X.509 certificate for Mobile Workflow application user authentication to test.

Prerequisites

1. Create a Mobile Workflow application that prompts the user to specify a certificate as credentials.
2. Package and assign the Mobile Workflow application to an iOS device user.

Task

1. During device application development, define and add a screen that has a Certificate Picker menu item.
2. Generate and deploy the application to the iPhone client.
3. Select **Certificate Picker** from the iPhone client.
4. Enter Windows credentials and certificate password in the dialog and click **Done**. Make sure the format is *domain**user*.
5. Submit the credentials to Unwired Server.

Manage a Mobile Workflow Package

The Workflows node in Sybase Control Center allows administrators to view and manage deployed Mobile Workflow packages, including Mobile Workflow display name, module name, and module version.

Administrators deploy Mobile Workflow packages into the Unwired Platform cluster through this node, as well as manage e-mail settings configuration.

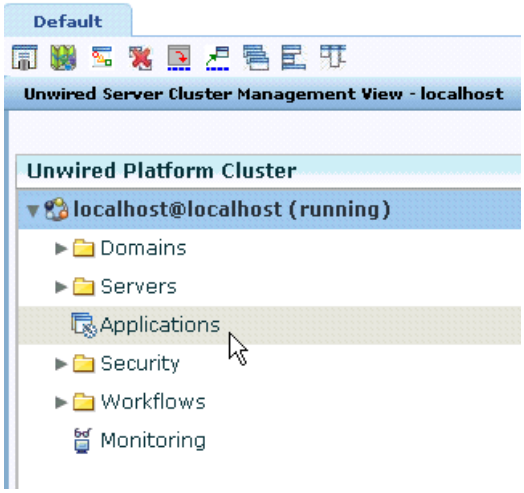
Registering and Reregistering Mobile Workflow Application Connections

Use Sybase Control Center to trigger the registration and application activation process for Mobile Workflows.

Note: When using a Windows Mobile emulator or BlackBerry simulator to register an application connection in Sybase Control Center, the device ID changes each time you reset the emulator to factory settings and reinstall the client. Before reinstalling, you must delete the original application connection from Unwired Server. Then, reregister the application connection. Otherwise, the device log shows a `Wrong Device for Code` error when the

device attempts to connect after registration. This problem occurs with Windows Mobile emulators and BlackBerry simulators.

1. Log in to Sybase Control Center.
2. In the left navigation pane, click the **Applications** node.



3. In the right administration pane, click the **Application Connections** tab.
4. Click **Register** to register the Mobile Workflow application connection to a device, or **Reregister** to update the application connection for an existing application connection.
5. In the **Register Application Connection** or the **Reregister Application Connection** dialog:
   a) For new registrations only, type the name of the user that will activate and register the Mobile Workflow application. For reregistrations or clones, the same name is used and cannot be changed.

      Users can use an e-mail address as a username, however, those users must ensure that e-mail addresses are processed correctly, especially when a security configuration is paired with the e-mail address. See *Security > Server Security > Enabling Authentication and RBAC for User Logins > Supported Providers and Credential Types > Considerations for Using E-mail Addresses as User Names*.
   b) Select the **HWC** template.
      The template you choose supplies initial values in the subsequent activation fields.
6. Change the default activation field values for the template you have chosen.

   If you are using a relay server, ensure the correct values are used.

   - **Server name** – the DNS name or IP address of the primary Unwired Server, such as "myserver.mycompany.com". If using relay server, the server name is the IP address or fully qualified name of the relay server host.

- **Port** – the port used for messaging connections between the device and Unwired Server. If using relay server, this is the relay server port. Default: 5001.
- **Farm ID** – a string associated with the relay server farm ID. Can contain only letters A – Z (uppercase or lowercase), numbers 0 – 9, or a combination of both. Default: 0.

  Note: If the device uses relay server to connect to Unwired Server, the farm ID should be the name of the Unwired Server farm configured in the relay server for messaging-based synchronization applications. If the device connects to Unwired Server directly, the farm ID should be 0.
- **Activation code length** – the number of characters in the activation code. If you are reregistering or cloning an application connection, this value cannot be changed.
- **Activation expiration** – the number of hours the activation code is valid.

7. (Optional) Select the check box adjacent to **Activation Code** to enter the code sent to the user in the activation e-mail. This value can contain letters A – Z (uppercase or lowercase), numbers 0 – 9, or a combination of both. Acceptable range: 1 to 10 characters.

   If the activation code is automatically generated, the code for the application connection can be retrieved from the **Connections** group of the **Application Connection Properties** dialog
8. Click **OK**.

Enabling and Configuring the Notification Mailbox

Configure the notification mailbox settings that allow Unwired Server to transform e-mail messages into mobile workflows.

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 mobile workflow 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 Sybase Control Center.
2. In the left navigation pane, click **Workflows**.
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 Unwired Server and the e-mail server in your environment.

- **Server** and **Port** – configure these connection properties so Unwired 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 Unwired 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 Mobile Workflows

Assign mobile workflow packages to make them available to a device user. Unassign them when a package is no longer required.

1. In the left navigation pane of Sybase Contorl Center, click **Workflows > <Mobile_WorkFlow_Package>**.
2. In the right administration pane, click the **Application Connections** tab.
3. Locate the device to assign a mobile workflow package to, then:
   a) Click **Assign Workflow**.
   b) List the activation users to assign the mobile workflow package to.

      By default, no users are listed in this window. 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.
   c) Select the user or users from the list to which to assign the mobile workflow package.
   d) Click **OK**.
4. To unassign a mobile workflow package, select the User and click **Unassign Workflow**.

Activating the Workflow

The menu items on a Workflow screen can be either a Submit Workflow (asynchronous) or Online Request (synchronous) menu item type.

To complete the mobile workflow activation process, the last screen in the mobile workflow application must have a Submit Workflow menu item. This is necessary for the device and server-side to activate the mobile workflow for the device.

Mobile workflows are considered to have been processed and/or activated only if they are closed with a Submit Workflow menu item, and which may, or may not, have a corresponding mobile business object (MBO) operation tied to it.

Configuring Context Variables for Mobile Workflow 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, click **Workflows > <Workflow_Package>:<Workflow_Version>**.
2. In the right administration pane, click the **Context Variables** tab.
3. Select the context variable to configure, then click **Modify**.

Context Variable	Description
SupUser	The valid mobile workflow application user ID that Unwired Server uses to authenticate the user. Depending on the security configuration, Unwired Server may pass that authentication to an EIS.
SupUnrecoverableErrorRetryTimeout	After sending a JSON request to Unwired Server, if you receive an EIS code that indicates an unrecoverable error in the response log, the mobile workflow client throws an exception. A retry attempt is made after a retry time interval, which is set to three days by default. Select this property to change the retry time interval.
SupThrowCredentialRequeston401Error	The default is **true**, which means that an error code 401 throws a `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.
SupRecoverableErrorRetryTimeout	After sending a JSON request to Unwired Server, if you receive an EIS code that indicates a recoverable error in the response log, the mobile workflow 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.

Context Variable	Description
SupPassword	The valid mobile workflow application user password that Unwired Server uses to authenticate the user. Depending on the security configuration, Unwired Server may pass that authentication to an EIS. An administrator must change development/test values to those required for a production environment.
SupPackages	The name and version of the MBO packages that are used in the workflow. This cannot be changed.
SupMaximumMessageLength	Use this property to increase the allowed maximum Mobile Workflow message size. Limitations vary depending on device platform: • For BlackBerry 5, the limit is 512KB. • For Windows Mobile the limit is 500KB. • For BlackBerry 6 and Android, the limit depends on the memory condition of the device. Large message may result in an out of memory error.

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, click **Workflows** > **<Workflow_Package>:<Workflow_Version>**.
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 Mobile Workflow Package

The administrator can change the credential certificate assigned at design time.

Note: Sybase recommends that you use Internet Explorer to perform this procedure.

1. In the left navigation pane, click **Workflows** > ***<Workflow_Package>:<Workflow_Version>***.
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.

Security

Set up static or dynamic authentication, and configure the Mobile Workflow application to use credentials.

Credentials

You can use either dynamic or static credentials in a mobile workflow form.

See *Security* and *System Administration* for more detailed information about implementing security and certificates.

The user name and password values are required when the mobile workflow application 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.

As an example of this, you might define a server-initiated workflow 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 mean that the user sets the user name and password on a screen that is pointed to by the credential request starting point. The text fields must have the corresponding Credential Cache User Name and Password checkbox checked to indicate the value is to be used to provide the user name and password on the client. When the user logs in, the credentials are authenticated using the stored credentials.

Note: If an e-mail triggered workflow has dynamic cached credentials, the cached credentials are not cached between invocations of the workflow form through an email 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 mobile workflow application can be extracted from the selected Sybase Unwired Platform profile user name and password when the mobile workflow application is generated, or they can be hard-coded using the Properties view. After deployment, you can change static credentials in Sybase Control Center.

The application can also have a credential screen (Credential Request) that appears if the mobile workflow application 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 SUP Server connection profile authentication – selected by default. When the code is generated for the mobile workflow application, the user name and password associated with the SUP connection profile are used.
   - 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 – when you select this option, you can use a certificate to generate authentication credentials.
4. (Optional) If you selected the **Use hard-coded credentials** option 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 when you want 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 mobile workflow, 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 mobile workflow.

1. In the Mobile Development perspective, select **File** > **New** > **Mobile Workflow Forms Editor**.
2. Follow the instructions in the New Mobile Workflow Forms Editor wizard:

Field	Description
Enter or select the parent folder	Select the mobile application project in which to create the mobile workflow form.
File name	Enter a name for the mobile workflow form. The extension for mobile workflow forms is .xbw.
Advanced	Link the mobile workflow form 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 mobile workflow form. 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 Work-Space 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. Continue with the New Mobile Workflow wizard as appropriate to create the type of mobile workflow application you want to create. Click **Finish**.
5. When the Mobile Workflow Forms Editor opens, click **Flow Design**.

   The Credential Request starting point and its associated Credential Request screen appear on the Flow Design page.

   Select the Credential Request starting point. You see the two pre-defined keys (cc_username and cc_password) in the Properties view.
6. Double-click the **Credential Request** screen to go to the Screen Design page.

   Two editbox controls, which are bound to the pre-defined cc_username and cc_password keys appear on the screen.
7. Select the **Username** editbox, then click **Advanced** on the left side of the Properties view.

   The Username editbox has the **Credential cache username** checkbox selected. Select the Password editbox and note that is has the **Credential cache password checkbox** checked.

   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 the Properties.
   c) In the Properties view, from Type, choose **Select Certificate**.

      When the user selects the menu item on the device, a dialog is opened that allows the user to select a certificate to use for credentials.
9. Select **File** > **Save**.

   The first time the mobile workflow is started following deployment, the credential screen is shown. The username and password values are then cached in the credential cache.

   Note: If an e-mail triggered workflow form has dynamic cached credentials, the cached credentials are not cached between invocations of the workflow form through an email 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 *Security > Server Security » Enabling Authentication and RBAC for User Logins > Authentication in Unwired Platform > Built-in Security Providers for User Authentication and Authorization > HTTP Authentication Security Provider* for more information.

Single Sign-on

Android, BlackBerry, and iOS Workflow applications 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 workflow application. 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 *Security > Server Security > Enabling Authentication and RBAC for User Logins > Authentication in Unwired Platform > Built-in Security Providers for User Authentication and Authorization > HTTP Authentication Security Provider* 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 Sybase Unwired Platform.
3. Sybase Unwired 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. Sybase Unwired 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 crdentials again.

Configuring the Workflow Application to Use Credentials

Configure a Mobile Workflow application to pass user credentials, which are authenticated by Unwired Server and the EIS.

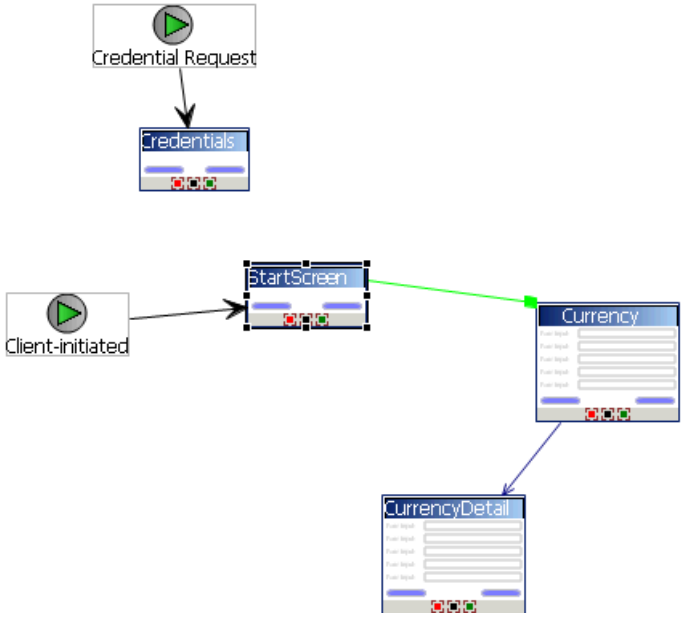
For information about configuring and implementing X.509 and SSO2 on the server, see the Sybase Unwired Platform *Security* documentation.

Configuring the Workflow Application 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 workflow application user selects a certificate to gain access to the MBO and related resources.

1. In the Mobile Workflow Forms Editor, add a **Credential Request** starting point to the workflow application.
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 Workflow type.

   On the device, the Specify Certificate Credentials action prompts the user for a *.p12 certificate and passes it to Unwired 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 Workflow Application to Use Static X.509 Credentials

When using static credentials, the workflow application does not prompt the user for credentials, instead it passes the credentials to Unwired Server automatically and displays the workflow application's start screen.

1. Remove the Credential Request starting point and screen from the workflow application (so the client is no longer prompted for credentials).
2. From Flow Design, select **Authentication**, **Use static credentials**, and **Use certificate-based credentials**.
3. Click **Generate from Certificate**.
4. Browse to the location of the `*.p12` certificate file.
5. Enter the certificate's password, select the alias and click **OK**.
6. Save and regenerate the Mobile Workflow package, and reassign it to a device.

Propagating a Client's Credentials to the Back-end Data Source

Use client credentials 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 Unwired 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.

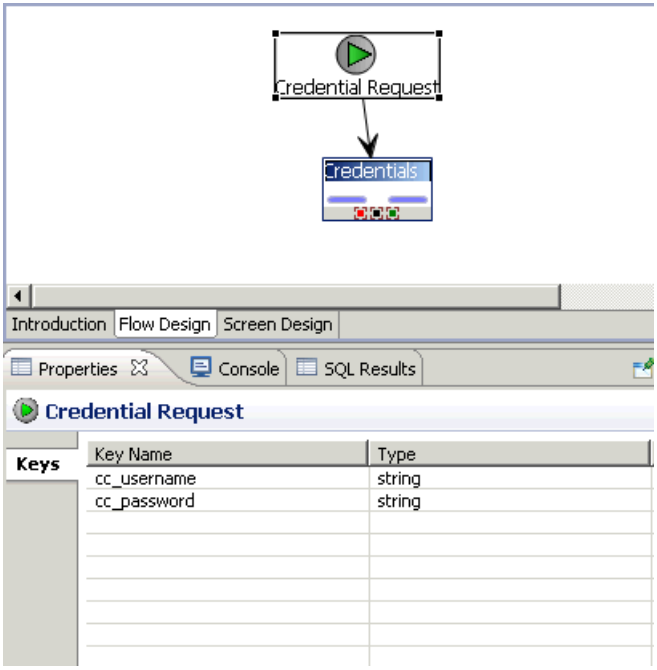
   Note:

- 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 Sybase Control Center (SCC).
- When you create a new security configuration that includes the SAPSSOTokenLoginModule, and deploy it to a new domain, if the mobile workflow application uses the MBOs associated with the new security configuration, you must specify an Unwired Server domain that corresponds to the domain using the security configuration. See the Sybase Unwired Platform *Security* guide for more information about security configurations

Configuring a Workflow Application to Use SSO2 Tokens

Configure a Credential Request starting point from which a workflow application user can pass a user name and password to gain access to the MBO and related resources.

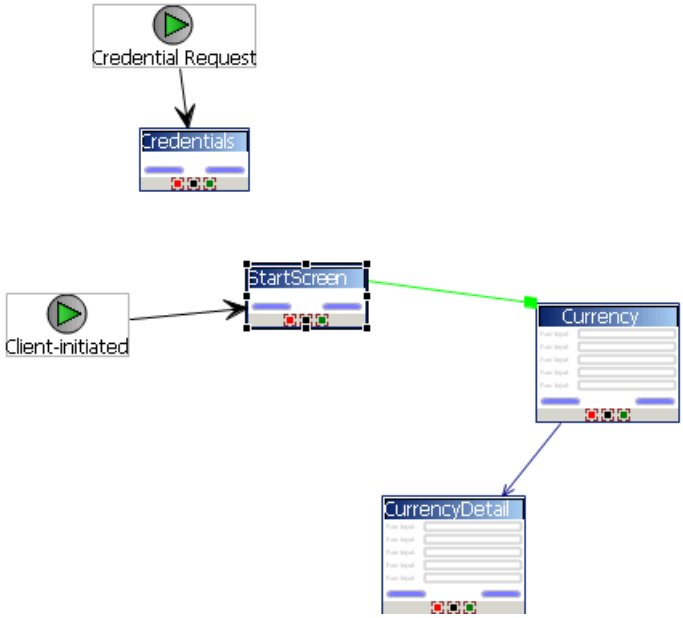
1. In the Mobile Workflow Forms Editor, add a **Credential Request** starting point to the workflow application.
2. Add two keys to the Credential Request named `cc_username` and `cc_password`.
3. Add a screen named `Credentials` and connect it to the Credential Request starting point.



4. Double-click **Credentials** to open it in the Screen Design. 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 workflow application. You could also add a **Submit workflow** menu item instead of **Save screen**.

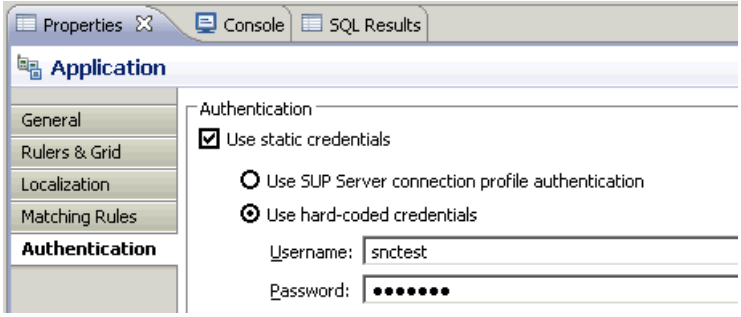
5. Add a Client-initiated starting point to which you add screens that contain the Submit menu items used to run MBO operations and object queries, return and display results, and so on. These actions all use the same credentials created in the previous steps.



Configuring the Workflow Application to Use a Static SSO2 Token

When using static credentials, the workflow application does not prompt the user for credentials, instead it passes the credentials to Unwired Server automatically and displays the workflow application's start screen.

1. Remove the Credential Request starting point and screen from the workflow application (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 Sybase Control Center for the server connection (for example: snctest/snctest).



3. Save and regenerate the workflow package, and reassign it to a device.

Modify Certificate Information for Workflow Packages

If using static credentials, either SSO token or static x.509 certification, you can replace the workflow package certificate using either Sybase Control Center or the SUPMobileWorkflow.replaceMobileWorkflowCertificate() API. To replace a certificate, you must have access to the certificate file and password.

Replacing the Mobile Workflow Certificate Through Sybase Control Center

If using static credentials, you can set or modify the context variable certificate settings for a mobile workflow package from Sybase Control Center.

The mobile workflow certificate password context variable is read-only. You can modify this only by using the Admin Java API method `SUPMobileWorkflow.replaceMobileWorkflowCertificate()`.

1. From Sybase Control Center, navigate to **Workflows** > **WorkflowName**, where *WorkflowName* is the name of the workflow package.
2. On the Context Variables tab, verify that `SupUser` and `SupPassword` contain valid credentials for the specified security configuration, for workflow packages that do not use certificate-based authentication.
3. For workflow packages that use certificate based authentication, you can view these context variables:
   - SupCertificateIssuer
   - SupCertificateSubject
   - SupCertificateNotAfter
   - SupCertificateNotBefore

Replacing the Mobile Workflow Certificate Using the Admin API

Use the SUPMobileWorkflow.replaceMobileWorkflowCertificate() method to set or modify the certificate password context variable for the workflow package.

```
InputStream is = workflowRL.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();
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setWID(4);
workflowID.setVersion(1);

workflow.replaceMobileWorkflowCertificate(workflowID,
    baos.toByteArray(), "password");
```

Content Security on Devices

This explains how the files that make up the mobile workflow container are protected when stored on the device, and under what circumstances the files are stored in plaintext.

Content Security on BlackBerry Devices

In general, all Hybrid Web Container files and extra data entered by the user, or retrieved from the server, are stored on the BlackBerry device's PersistentStore.

This is the same storage area used by e-mail, calendar entries, and applications. See your BlackBerry documentation for information about persistent store APIs.

The BlackBerry Hybrid Web Container uses the RIM PersistentContent APIs when reading and writing of data from PersistentStore is required. This ensures that the content being written is stored at the device's current encryption level. See your BlackBerry documentation for information about content protection strength settings.

When content protection is turned on, content on the BlackBerry device is protected using the 256-bit Advanced Encryption Standard (AES) encryption algorithm.

- Use 256-bit AES encryption to encrypt stored data when the BlackBerry device is locked
- Use an Elliptic Curve Cryptography (ECC) public key to encrypt data that the BlackBerry device receives when it is locked

These settings apply to the encryption of data that the BlackBerry device receives while locked:

Content protection strength setting	ECC encryption key length (in bits)
Strong	160
Stronger	283
Strongest	571

The BlackBerry Hybrid Web Container also registers a PersistentContentListener, which allows it to be notified when the device's encryption level changes. This also enables previously stored content to be re-encoded to the new encryption level setting. The device's encryption level setting can be changed by a BlackBerry Enterprise Server Administrator remotely, or by the user, from the device.

Hybrid Web Container Files

Hybrid Web Container files include all the files contained in the `<workflow_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 Workflow 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 `<workflow_package_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 workflow application 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 mobile workflow application 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 mobile workflow application 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 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 into a SQLite database on the device.

The crypto 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 `<workflow_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 SQLite database, 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. The Content Provider URL is further obfuscated with a randomly generated number that is required on the URL when the files are requested.

Attachments

If attachments, such as *.docx, *.pdf, and so on, are part of the `<workflow_package_name>.zip` deployed to the device, they are stored in the encrypted SQLite database after they have been encrypted through the Google/Android crypto libraries.

- When the JavaScript requests these attachments for viewing, they are read from the SQLite database, and temporarily written unencrypted to the device's flash memory for the external viewers to display them.
- Once the mobile workflow 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 mobile workflow 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 crypto 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 crypto 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 Workflow 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 crypto 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 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 <workflow_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 Workflow 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 package is finished loading.

Attachments

If attachments, such as *.docx, *.pdf, and so on, are part of the <workflow_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’s sandbox for the viewer to display them.
- Once the mobile workflow 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 mobile workflow 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 Workflow 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 a Workflow 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 Containerontainer 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, Sybase 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\Sybase\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\Sybase\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 `<workflow_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 Workflow 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 <workflow_package_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 mobile workflow application closes, these temporary attachment files are immediately removed.

Images

Images are stored unencrypted on the file system, then removed when the Workflow application 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 Workflow 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 Mobile Workflow Forms Editor, such as the names of screen controls, screens, and mobile business objects.

You can localize the mobile workflow by creating locale properties files. You can then load, update, and generate localized mobile workflow applications.

All the localizable strings in the Mobile Workflow Forms Editor XML model work as resource keys in the localization properties file. All the localization properties files are in the same directory as the Mobile Workflow packages (`.xbw` files).

Resource keys are divided into these categories, which include all the elements of the Mobile Workflow Forms Editor XML model:

- Menus
- Controls
- Screens

Localization consists of two levels of localization—the Mobile Workflow Forms Editor XML model localization and the Mobile Workflow client localization.

All locale properties files are saved in the same directory as the Mobile Workflow package.

To ensure that the correct locale is picked up for the Mobile Workflow 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\workflow*.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\workflow*.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\workflow*.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\workflow*.html`, that file is used and the HTTP lang parameter is set to "".

Localization Limitations

Some restrictions for the locale properties files apply:

- Traditional Chinese characters are not supported on iOS.
- Mobile workflow applications 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 mobile workflow 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 Mobile Workflow Package

Use the Mobile Workflow Forms Editor to complete these tasks to localize Mobile Workflow packages (.xbw files).

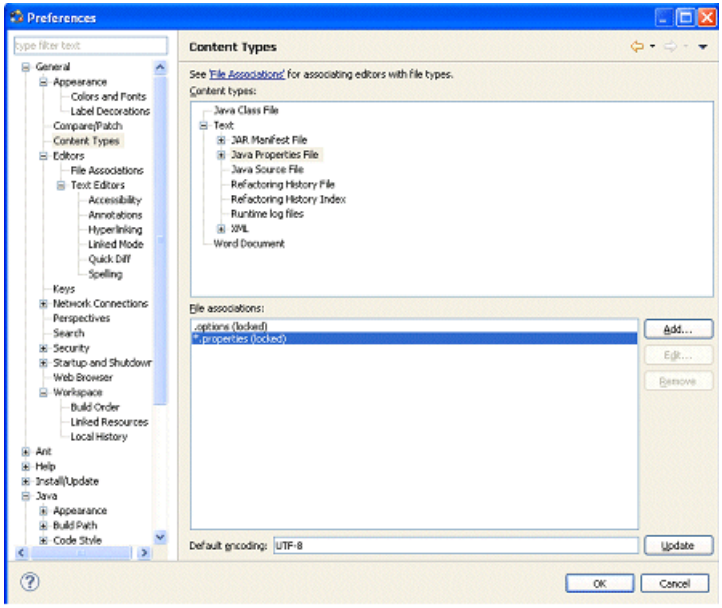
Changing the Encoding Type

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 Sybase Unwired 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(locked)`.
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

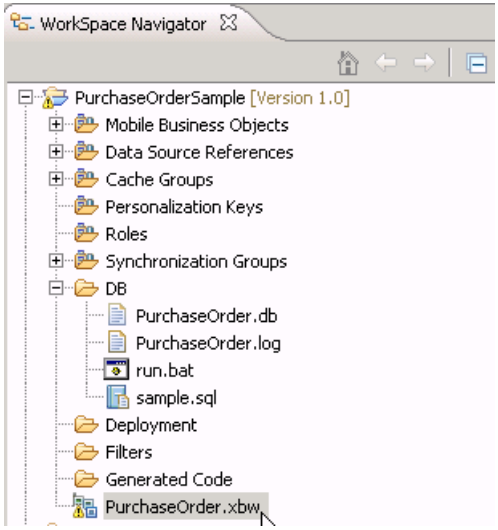
You must have an existing Mobile Workflow package before you create the locale properties file.

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 `<mobile_workflow>.xbw` file to open the Mobile Workflow Forms Editor.



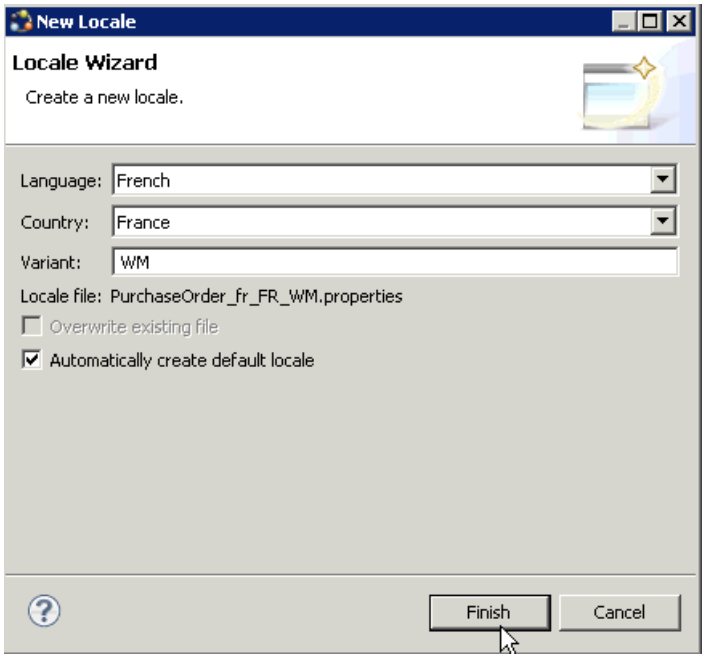
2. Click the **Flow Design** tab.
3. Right-click in a blank area on the Flow Design page, and select **Show Properties View**.
4. In the Properties view, on the left, click the **Localization** tab.
5. In the right pane, click **New**.
6. Select or enter the information for the new locale, select **Automatically create default locale**, and click **Finish**.

Option	Description
Language	Select the language.
Country	Select the country.
Variant	Enter the variant, which is the vendor or browser-specific code. For example, enter "WIN" for Windows, "MAC" for Macintosh and "POSIX" for POSIX. If there are two variants, separate them with an underscore, and put the most important one first. For example, a Traditional Spanish collation might construct a locale with parameters for language, country, and variant as: "es", "ES", "Traditional_WIN".
Overwrite existing file	Overwrite an existing localization file.

Option	Description
Automatically create default locale	Automatically create the default locale properties file. For example, if you specify the language as "English" and the country as the "United States" for a device application called `test`, then both `test_en_uS.properties` and `test.properties` files are created.

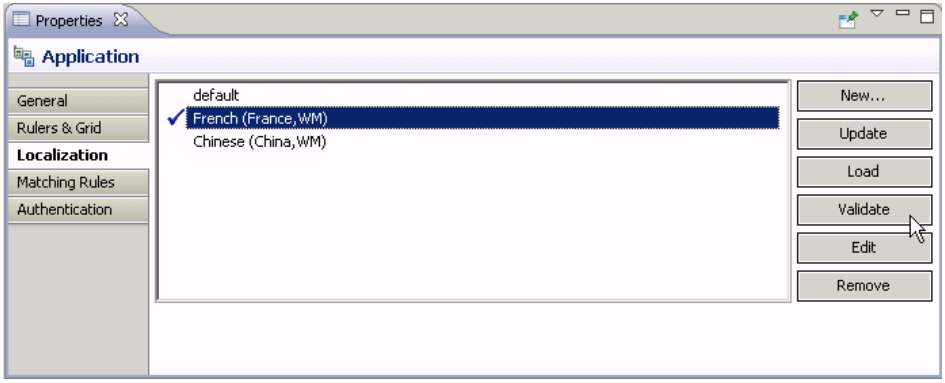
For example:

- Language – select **French**.
- Country – select **France**.
- Variant – enter a value to make this locale file unique from others, for example, `WM` for Windows Mobile.



This locale file is now the default locale file, and will be used when the regional setting of the device does not match that of any supplied locale file.

7. In the Properties view, in the Localization page, select the file to validate and click **Validate**.



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 Mobile Workflow Designer.

If the locale properties file does not already exist, it is created. If the current locale is not defined in the mobile workflow application file, the updated locale is used as the default, and the file name is *{device_application}*.properties. Otherwise, the locale defined in the mobile workflow application 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.

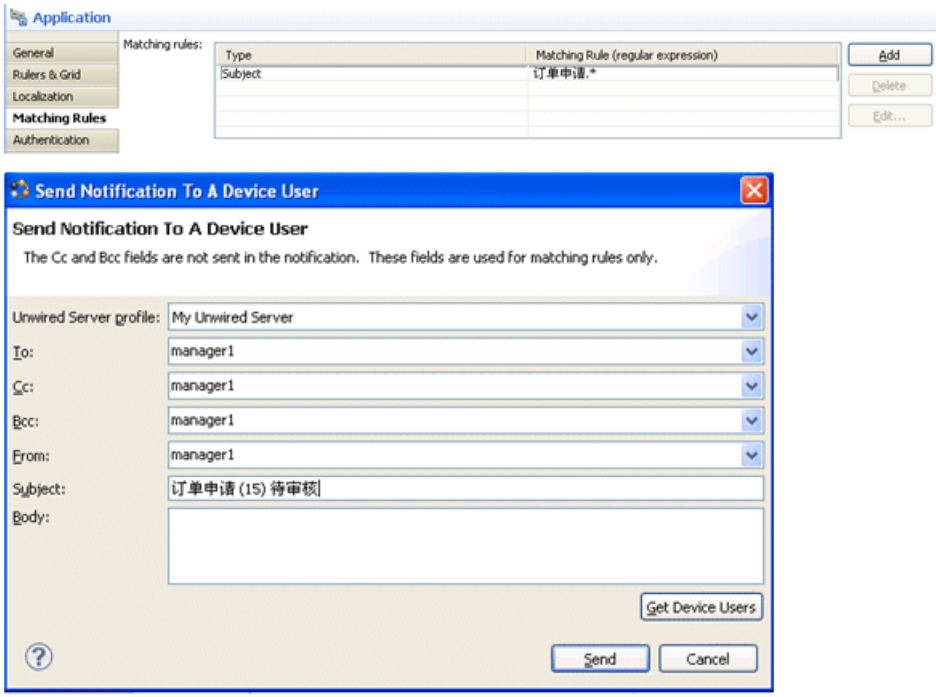
1. In the Screen Design page Properties view, click **Localization**.
2. Click **Update**.

Mobile Workflow Package Internationalization

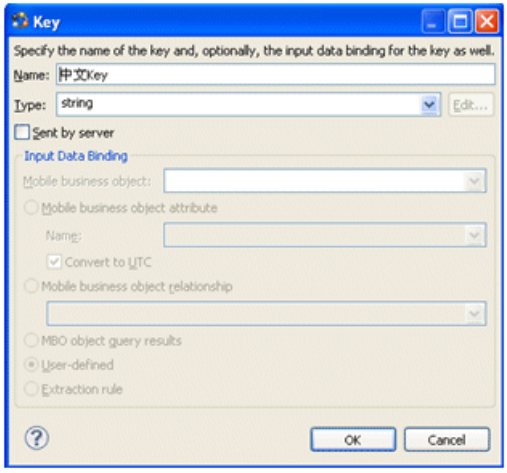
The internationalization feature depends on the internationalization setting on the operating system where Sybase Unwired Platform Mobile Workflow is running.

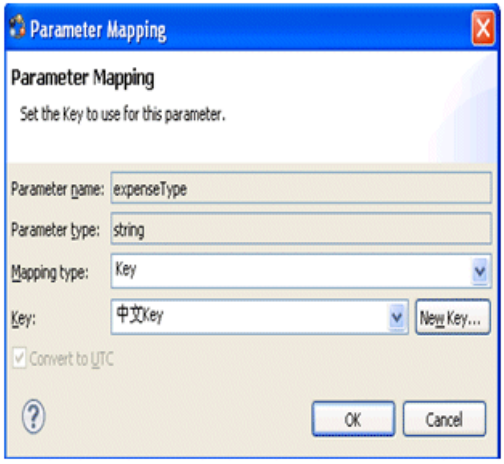
In the Mobile Workflow Forms Editor, you can use international data in:

- Matching rules for notifications.



- Key names – you can create keys with names in other languages and map them to mobile business object parameters.





- 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 Mobile Workflow 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 Mobile Workflow Packages

Test a Mobile Workflow on a device or simulator.

1. Launch and/or connect to the mobile device or emulator.
2. Deploy the Mobile Workflow package to the device.
3. Establish the connection to the server on the device.
4. For user-initiated Mobile Workflow packages, go to the Workflows menu of the e-mail inbox and click on the appropriate Workflow package.

5. For e-mail subscription Mobile Workflow 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 Mobile Workflow Packages

Test a server-initiated Mobile Workflow package.

1. In the Mobile Workflow Forms editor, open the Mobile Workflow form, `<workflow_form>.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 Unwired 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 Mobile Workflow form.
5. Click **Send**.
6. On the client, from the applications screen, open SUPWorkflows.
7. In the client application, click **WorkFlow Messages**. This contains the server-initiated Mobile Workflow form.

Viewing Workflow Messages on the Device

Where Workflow 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 Workflow connection settings before any Workflow packages appear on the device.

BlackBerry

To see Workflow messages on BlackBerry devices and simulators:

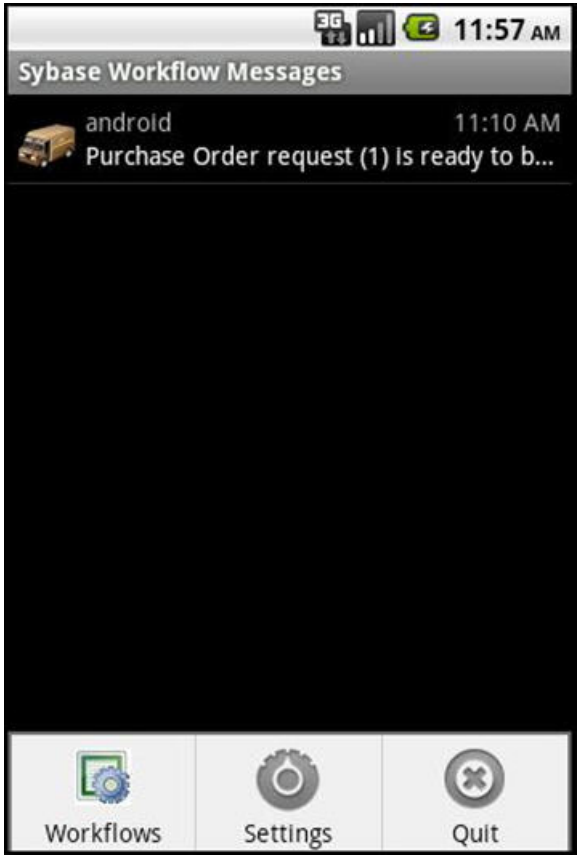
1. Close the Mobile Workflow application.
2. Go to the Messages inbox. The mobile workflow messages are in the inbox.
3. Select **Workflows** from the menu.

Android

To see Workflow messages on Android devices and simulators:

1. Open the **Workflows** application.
2. Open the Mobile Workflow application for which you want to view messages.

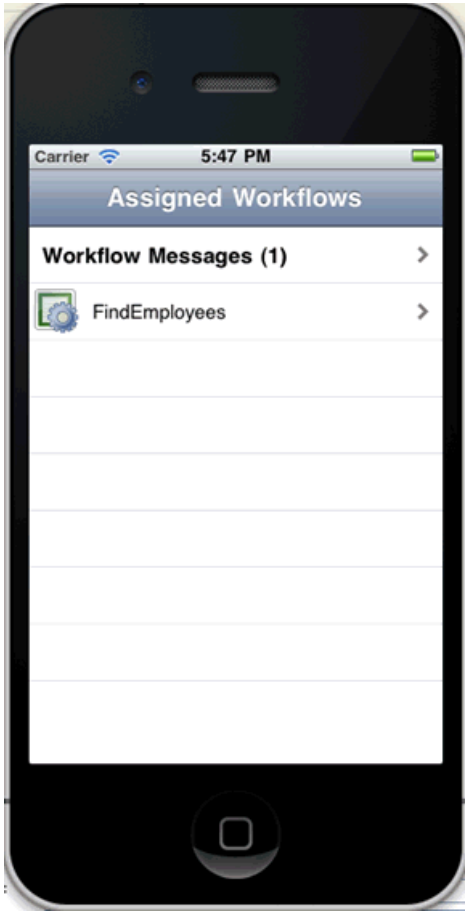
3. Click the message to view.



iOS

To see Workflow messages on iOS devices and simulators:

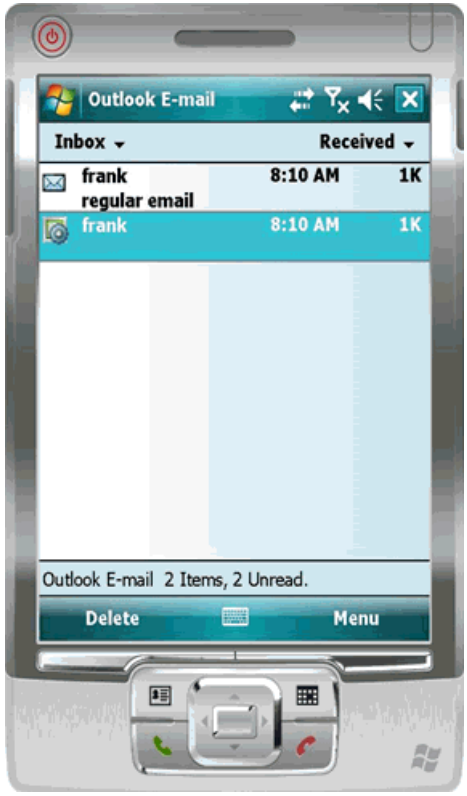
1. Open the **Workflows** application.
2. Click **Mobile Workflow Messages** to view messages.



Windows Mobile

To see Workflow messages on Windows Mobile devices and emulators:

1. Open the Outlook E-mail inbox.
2. Workflow messages are in the inbox, with a mobile workflow icon.



Launching a Server-initiated Mobile Workflow on the Device

Server-initiated Mobile Workflow messages show up as an e-mail message on the Windows Mobile or BlackBerry device inbox.

On Windows Mobile platforms, Mobile Workflow packages are integrated with the Outlook Mail inbox. On iOS, messages are sent to a container that is provided by the Workflow device client.

Click the e-mail message to launch the Mobile Workflow container and display the Mobile Workflow associated with the e-mail message.

When you click the **Workflows** menu item in the device inbox, only the latest version of the Workflows appear. When you click the Workflow icon for a particular workflow, the Workflow version that is associated with the e-mail notification is launched, whether it is the latest version or not.

Example

You develop a Mobile Workflow application that has both client-initiated and server-initiated starting points. You deploy the initial version, which is called version 1, and a Mobile Workflow e-mail notification is sent.

Next, make some changes and deploy a second version, version 2. Again, a Mobile Workflow e-mail notification is sent.

There are now three ways that this Mobile Workflow application can be launched, and the way that it is launched determines which version of the Workflow is launched:

- If you launch the application from the **Workflows** menu item, the last deployed version of the Workflow, in this case, version 2, is launched. Although version 1 of the Mobile Workflow application still exists somewhere on the device it is never used as long as you launch the Workflow from the Workflows menu.
- If you launch the Workflow by opening the initial e-mail 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 Workflow 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 Mobile Workflow package html and js 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 Mobile Workflow to Debug.
2. Open the browser to use for debugging and open the Java Console.

   If you are using Chrome:

   a) Add the following command line option to the shortcut used to start Chrome:

   ```
   ..\chrome.exe" --allow-file-access-from-files
   ```

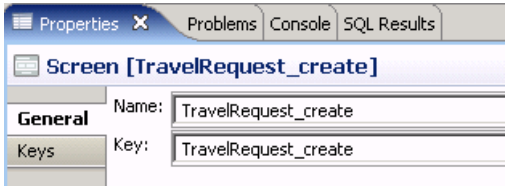
3. You can debug a client-initiated Mobile Workflow application up until the point where a menu item of the Submit Workflow type is performed. If the menu item action is an Online Request, place the XMLWidgetMessage (available in the WorkflowClient trace log located in `<UnwiredPlatform_InstallDir>\UnwiredPlatform\Servers\UnwiredServer\logs\WorkflowClient`) that is the expected response message into an `rmi.xml` file and place it at the same level as the generated `workflow.html` file.

   Note: Control characters are not parsed correctly when using rmi.xml and Chrome to debug the Mobile Workflow. Do not format the content of the `rmi.xml` when debugging the Mobile Workflow 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 `workflow.html` file for the Mobile Workflow application 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/MobileWorkflow101/Generated%20Workflow/
travelrequest/html/workflow.html?
screenToShow=TravelRequest_create
```

7. To simulate an e-mail message triggered Mobile Workflow application:
   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 Mobile Workflow application you are debugging, place the `transform.xml` file at the same level as the generated `workflow.html` file (`Generated Workflow\<Workflow_application_name>\html`).
   c) Add a **?loadtransformdata=true** parameter to load the data into the Workflow application.

Configuring Mobile Workflow Tracing in SCC

Change the tracing level for Mobile Workflow packages in SCC.

1. Log on to Sybase Control Center.
2. In the Unwired Platform Cluster view, select **Servers** > **<host_name>** > **Log**.
3. Click **Message Server** > **Settings**.
4. Select **WorkflowClient** and click **Properties**.
5. From Level, select **Debug** and click **OK**.

Create a Mobile Workflow Package Manually

While using the Mobile Workflow Forms Editor is the easiest and fastest way to develop and customize mobile workflow applications, it is also possible to develop a mobile workflow package outside of the constraints of the Mobile Workflow Forms Editor.

Developing a mobile workflow application this way allows you to use of a greater variety of application designs, from using different HTML formatting to using different Web application frameworks, and beyond. It should be emphasized, however, that the Mobile Workflow Forms Editor does a lot of the work to seamlessly support multiple platforms, which you must duplicate if you choose not to use it.

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

```
function processWorkflowMessage(incomingWorkflowMessage)
```

The Workflow container needs to call this function when online request processing is complete. The incoming workflow message is an XML-formatted string.

Mobile Workflow URL Parameters

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

An example of how to use these URL parameters can be found in the `onWorkflowLoad()` function in the `Utils.js` file.

URL parameter	Description
loglevel	Current device log level.
screenToShow	Name of the screen which should be displayed.
supusername	Username of the current Workflow (if available).
lang	Current language of the device.
isalreadyprocessed	Indicates whether or not the Workflow 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.

URL parameter	Description
loadtransformdata	Indicates that the JavaScript should request the transform data (contents of the e-mail message) from the Container using the loadtransformdata querytype. For information about the query types, see the topic *Calling the Hybrid Web Container*.
ignoretransformscreen	Indicates that the JavaScript should ignore the RequestScreen tag in the transform data (contents of the e-mail message). This is set to true when the screen that needs to be shown is either the Activation or Credentials screen.

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 that the Mobile Workflow Forms editor generates.

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

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

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

setscreentitle
Sets the native screen title on the Web Container.

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

addMenuItem
Adds a single menu item to the Web Container.

removeallmenuitems
Removes all the menu items from the Web Container.

clearrequestcache
Clears the entire Online Request cache for the current Mobile Workflow application.

clearrequestcacheitem
Clears a single Online Request cache entry for the current Mobile Workflow application.

logtoworkflow
Logs a message to the `AMPHostLog.txt` (`mocalog.txt` for iOS) on the device. This log file can be retrieved remotely from Sybase 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 Workflow 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 Web Container.

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

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

rmi
This function executes an online request to the Unwired 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 Unwired 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 Unwired 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 Web Container for the transform data (the contents of the e-mail message) for the current Workflow message.

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

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

Mobile Workflow Package Files

To build a mobile workflow package manually, you should first familiarize yourself with its contents.

This section describes the contents of the Mobile Workflow 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 Mobile Workflow package contains Web application files.

When developing a Mobile Workflow package manually:

- Include HTML files that follow the same general pattern as the files generated when using the Mobile Workflow Forms editor to generate the Mobile Workflow 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
- Use `WorkflowMessage.js` to view and manipulate the workflow messages

HTML Format

This is the basic HTML format.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
        <meta name="HandheldFriendly" content="True" />
        <meta http-equiv="PRAGMA" content="NO-CACHE" />
        <link rel="stylesheet" href="css/MyStylesheet.css"
type="text/css" />
        […]
        <script src="js/API.js"></script>
        <script src="js/Utils.js"></script>
        <script src="js/WorkflowMessage.js"></script>
        <script src="js/MyJavaScript.js"></script>
        […]
        <script>
[…]
        </script>
    </head>
    <body onload="onWorkflowLoad();">
      <div id=Screen1KeyScreenDiv" sup_screen_title=”Screen1Title"
style="display: none"
sup_menuitems="NativeMenu1Key,NativeMenu1DisplayName,NativeMenu2Key
,NativeMenu2DisplayName" sup_okaction="myOKAction()">
[…]
        <form style="margin: 0px;" name="Screen1KeyForm"
id="Screen1KeyForm" onSubmit="return false;" autocomplete="on">
[…]
        </form>
[…]
      </div>
    </body>
     <script>
[…]
$(document).ready( function(){
        […]
 });
 […]
     </script>
</html>
```

Manifest.xml File

The `manifest.xml` file describes how the contents of the Mobile Workflow package `.zip` file are organized.

This file must reside at the root of the Mobile Workflow .zip package. This shows the outline of what the manifest.xml file contains.

Manifest.xml

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

  <TransformPlugin>
    <File>WorkflowClient.dll</File>
    <Class>Sybase.UnwiredPlatform.WorkflowClient.Transformer</Class>
  </TransformPlugin>

  <ResponsePlugin>
    <File>WorkflowClient.dll</File>
    <Class>Sybase.UnwiredPlatform.WorkflowClient.Responder</Class>
  </ResponsePlugin>

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

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

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

  <MatchRules>
    <SubjectRegExp>…</SubjectRegExp>
    <ToRegExp>…</ToRegExp>
    <FromRegExp>…</FromRegExp>
    <CCRegExp>…</CCRegExp>
    <BodyRegExp>…</BodyRegExp>
  </MatchRules>
</Manifest>
```

ModuleName

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

The `ModuleName` defines the name of the Mobile Workflow package.

ModuleVersion

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

The `ModuleVersion` defines the version of the Mobile Workflow package.

ModuleDesc

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

The `ModuleDesc` provides a short description of the Mobile Workflow package.

ModuleDisplayName

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

The name of the Mobile Workflow package that is displayed to the user in the Worfklow list on the device for Mobile Workflows that are client-invoked. When the Mobile Workflow 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 Mobile Workflow 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 Mobile Workflow 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 Mobile Workflow can be used without an associated e-mail. `1` = true, `0` = false. If 1 is specified, the Mobile Workflow is shown in the Workflow list on the device and can be used without the context of an e-mail message.

PersistAppDomain

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

States whether this Mobile Workflow 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 Workflow message shows a visual indication in the Inbox after it has been processed (1 = true, 0 = false).

Note: When a Workflow 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 Workflow message is deleted from the mobile device's Inbox after it has been processed (1 = true, 0 = false).

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

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

ProcessUpdates

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

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

CredentialsCache

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

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

RequiresActivation

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

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

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

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

TransformPlugin

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

Describes the server module implemented as a .NET assembly that implements the IMailProcessor interface. This module is responsible for processing the intercepted e-mail message before it gets delivered to the device.

Inner tags

`<File shared="true">WorkflowClient.dll</File>` The path, including the filename of the assembly that implements the IMailProcessor interface. The path is relative to the 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 workflows 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 workflow will use its own version of that DLL in the Workflow's own folder.

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

ResponsePlugin

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

Describes the server module implemented as a .NET assembly that implements the IResponseProcessor interface. This module is responsible for processing the response from the device.

Inner tags

`<File shared="true">WorkflowClient.dll</File>` The path, including the filename, of the assembly that implements the IResponseProcessor interface. The path is relative to the Mobile Workflow .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 workflows 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 workflow will use its own version of that DLL in the Workflow's own folder.

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

ClientWorkflows

```
<ClientWorkflows>
  <WindowsMobileProfessional>
    <HTMLWorkflow>
      <File>…</File>
      <HtmlFiles>
        <HtmlFile>…</HtmlFile>
```

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

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

Inner tags

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

`<File>…</File>`

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

```
<?xml version="1.0" encoding="utf-8"?>
<widget>
  <screens src="html/myAndroidWorkflow.html" default="Start_Screen">
    <screen key="html/myAndroidWorkflow.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 Mobile Workflow .zip package.

`<HtmlFile>…</HtmlFile>`

Indicates that the named file (html/js/API.js, html/myAndroidWorkflow.html) will be used on the specified platform. The referenced file must be present in the Mobile Workflow .zip package.

ContextVariables

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

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

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

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

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

Inner tags

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

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

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

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

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

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

MatchRules

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

Describes the collection of match rules that will be used to determine if an e-mail message should be sent to a TransformPlugin server module for processing. When Administrators deploy a Mobile Workflow, they have the ability to Add, Delete and /or 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 an e-mail message.

`<ToRegExp>…</ToRegExp>` The value to test for against the "To" line of an e-mail message.

`<FromRegExp>…</FromRegExp>` The value to test for against the "From" line of an e-mail message.

`<CCRegExp>…</CCRegExp>` The value to test for against the "CC" line of an e-mail message.

`<BodyRegExp>…</BodyRegExp>` The value to test for against the <Body> text of an e-mail message.

WorkflowClient.dll File

The `WorkflowClient.dll` file is used by the Unwired Platform messaging server to transform the data that is sent to the device as notifications, and to respond to online request and submit workflow actions from the device.

The supplied `WorkflowClient.dll` file loads the metadata in the WorkflowClient.xml file to determine how to map the data in the workflow message to and from calls to Mobile Business Object operations and object queries.

The `WorkflowClient.dll` is shared by all Mobile Workflows. It is installed only once, into the `UnwiredPlatform_InstallDir>\UnwiredPlatform\Servers\MessagingServer\bin` folder. Your `Manifest.xml` file must refer to the `WorkflowClient.dll` as a shared file. It does not need to be included in the Mobile Workflow .zip file.

WorkflowClient.xml File

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

WorkflowClient.xml

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

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

Globals

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

Describes the global information for the Mobile Workflow 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 workflow 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 2. Attributes

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

Attribute	Description
errorscreen	This attribute is optional. The screen to which the client will return, by default, if the MBO operation/object query fails. If left unspecified, the client application remains on the current screen. This attribute is applicable only to online request actions.
• errorlogskey • errorlogmessagekey • errorlogmessageaslistkey	The keys used to fill any error log messages.

Inner tags

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

Method

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

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

Table 3. Attributes

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

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 workflow 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 4. Attributes

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

Inner tags

`<Value> … </Value>` Contains the description of where to obtain the parameter values of the MBO operations to be executed when this online request or submit workflow 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 workflow message.

Table 5. Attributes

Attribute	Description
sourceType	The source of the data. Must be one of these types: • Key • BackEndPassword • BackEndUser • DeviceId • DeviceName • DeviceType • UserName • MessageId • ModuleName • ModuleVersion • QueueId • ContextVariable
workflowKey	If the sourceType is **Key**, the name of the key in the workflow message from which to obtain the value.
contextVariable	If the sourceType is **ContextVariable**, the name of the context variable from which to obtain the value.
paramName	If present, the name of the parameter the value is supplying.
pkName	If present, the name of the personalization key the value is supplying.
attribName	If present, the name of the attribute name the value is supplying. This value may, or may not, be present together with paramName.
parentMBO	The name of the parent MBO, if any.
relationShipName	The name of the relationship, if any.

Attribute	Description
mboType	The type of the value in MBO terms. Must be one of these types: • string • char • date • datetime • time • int • byte • short • long • decimal • boolean • binary • float • double • list • integer • structure
array	A boolean that indicates whether or not the value is an array. The default is false.
length	The length of the parameter/attribute/personalization key.
precision	The precision of the parameter/attribute/personalization key.
scale	The scale of the parameter/attribute/personalization key.
convertToLocalTime	A boolean that indicates whether or not to convert the value to a local time before passing it to the MBO. The default is false.

Inner tags

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

OutputBinding

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

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

Table 6. Attributes

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

Inner tags

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

Mapping

```
<Mapping workflowKey="Department_dept_id_attribKey"
workflowType="number" attribName="dept_id" mboType="int"/>
```

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

Table 7. Attributes

Attribute	Description
workflowKey	The name of the key in the workflow message to fill with the results of the object query.
workflowType	The type of the data in the workflow message. Must be one of these types: • text • number • boolean • datetime • date • time • list • choice
attribName	If present, the name of the attribute name to which the key is mapped.

Attribute	Description
hardCodedValue	If the workflowType is not choice, and attrib-Name is not present, the hard-coded value to which the key is mapped.
keyWorkflowKey	If the workflowType is choice, the key to which to map the dynamic display names of the choice.
valueWorkflowKey	If the workflowType is choice, the key to which to map the dynamic values of the choice.
assumeLocalTime	A boolean to indicate whether or not to assume that the values coming back from the object query are in local time or not. The default is false.
array	A boolean that indicates whether or not the value is an array. The default is false.
mboType	The type of the value in MBO terms. Must be one of these types: • string • char • date • datetime • time • int • byte • short • long • decimal • boolean • binary • float • double • list • integer • structure
relationShipName	The name of the relationship, if any.

Inner tags

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

Notification

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

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

Table 8. Attributes

Attribute	Description
type	The type of the notification. Must be onEmail-Triggered.
targetscreen	The screen to which the client will be opened if the object query succeeds.
errorscreen	The screen to which the client will be opened, by default, if the object query fails.
• errorlogskey • errorlogmessagekey • errorlogmessageaslistkey	The keys to use to fill any error log messages.

Inner tags

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

`<Methods> … </Methods>` Contains the description for one or more object queries to be executed when this online request or submit workflow 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 workflow message.

Table 9. Attributes

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

Attribute	Description
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 workflow message to fill with the value extracted from the server notification.
workflowType	The type of the data in the workflow 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 (WindowsMobileProfessional, BlackBerry, BlackBerry6, iOS, and Android) provides a `<File>…</File>` tag, which refers to an .xml file in the Mobile Workflow .zip package.

The contents are similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<widget>
  <screens src="html/myAndroidWorkflow.html" default="Start_Screen">
    <screen key="html/myAndroidWorkflow.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 Mobile Workflow package is generated usuing the Mobile Workflow Forms editor, the with the **Optimized for appearance** option selected in Preferences, three look and feel .xml files are generated: `workflow.xml`, `workflow_CustomLookAndFeel.xml`, and `workflow_jQueryMobileLookAndFeel.xml`.

Using Third-party Files

To load external JavaScript and CSS files dynamically when creating a Mobile Workflow package manually:

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

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

Troubleshoot

Use troubleshooting tips to isolate and resolve common issues.

See *Troubleshooting Sybase Unwired Platform* for information about troubleshooting issues for Workflow package and other Sybase Unwired Platform components.

HTTP Error Codes

Unwired Server examines the EIS code received in a server response message and maps it to a logical HTTP error code, if a corresponding error code exists. If no corresponding code exists, the 500 code is assigned to signify either a Sybase Unwired Platform internal error, or an unrecognized EIS error. The EIS code and HTTP error code values are stored in log records.

These tables list recoverable and unrecoverable error codes. All error codes that are not explicitly considered recoverable are considered unrecoverable.

Table 10. Recoverable Error Codes

Error Code	Probable Cause
409	Backend EIS is deadlocked.
503	Backend EIS is down, or the connection is terminated.

Table 11. Unrecoverable Error Codes

Error Code	Probable Cause	Manual Recovery Action
401	Backend EIS credentials wrong.	Change the connection information, or backend user password.
403	User authorization failed on Unwired Server due to role constraints (applicable only for MBS).	N/A
404	Resource (table/Web service/BAPI) not found on backend EIS.	Restore the EIS configuration.
405	Invalid license for the client (applicable only for MBS).	N/A
412	Backend EIS threw a constraint exception.	Delete the conflicting entry in the EIS.

Error Code	Probable Cause	Manual Recovery Action
500	Sybase Unwired Platform internal error in modifying the CDB cache.	N/A

Error code 401 is not treated as a simple recoverable error. If the `SupThrowCredentialRequestOn401Error` context variable is set to true (the default), error code 401 throws a `CredentialRequestException`, which sends a credential request notification to the user's inbox. You can change this behavior by modifying the value of the `SupThrowCredentialRequestOn401Error` context variable in Sybase Control Center. If `SupThrowCredentialRequestOn401Error` is set to false, error code 401 is treated as a normal recoverable exception.

Recovering from EIS Errors

After sending a JSON request to Unwired Server, if you receive in the response log message an EIS code which is recoverable, the mobile workflow client throws a `TransformRetryException` or `ResponseRetryException`, as is appropriate.

A retry attempt is made after a retry time interval, which is set by default to 15 minutes for recoverable errors, and by default to 3 days for unrecoverable errors. You can configure the retry time interval by setting the `SupRecoverableErrorRetryTimeout` (default: 15 minutes) and `SupUnrecoverableErrorRetryTimeout` context variables through the Sybase Control Center admin console.

Only certain error codes are considered to be recoverable.

Table 12. Recoverable Error Codes

Error Code	Probable Cause
409	Backend EIS is deadlocked.
503	Backend EIS down or the connection is terminated.

Note: If the problem with the EIS is not corrected, the retry process can continue indefinitely. Ensure that you set an appropriate retry time interval.

Other error codes are considered to be non-recoverable. A retry attempt is made after a retry time interval, which is set to three days by default.

Table 13. Non-recoverable Error Codes

Error Code	Probable Cause	Manual Recovery Action
401	Backend EIS credentials wrong.	Change the connection information, or backend user password.
403	User authorization failed on Unwired Server due to role constraints (applicable only for MBS).	N/A
404	Resource (table/webservice/BAPI) not found on Backend EIS.	Restore the EIS configuration.
405	Invalid license for the client (applicable only for MBS).	N/A
412	Backend EIS threw a constraint exception.	Delete the conflicting entry in the EIS.
500	SUP internal error in modifying the CDB cache.	N/A

Mapping of EIS Codes to Logical HTTP Error Codes

A list of SAP® error codes mapped to HTTP error codes. By default, SAP error codes that are not listed map to HTTP error code 500.

Note: These JCO error codes are not applicable for DOE-based applications.

Table 14. Mapping of SAP Error Codes to HTTP Error Codes

Constant	Description	HTTP Error Code
JCO_ERROR_COMMUNICATION	Exception caused by network problems, such as connection breakdowns, gateway problems, or unavailability of the remote SAP system.	503
JCO_ERROR_LOGON_FAILURE	Authorization failures during login. Usually caused by unknown user name, wrong password, or invalid certificates.	401

Constant	Description	HTTP Error Code
JCO_ERROR_RESOURCE	Indicates that JCO has run out of resources such as connections in a connection pool.	503
JCO_ERROR_STATE_BUSY	The remote SAP system is busy. Try again later.	503

Credentials Are Lost after User Successfully Passes Activation Screen

User logs in successfully on Activation screen, but is no longer logged in at some point after that.

This happens when you do not execute a Save from the Activation screen, and then execute a Cancel on a subsequent screen, before a Save is executed.

Always execute a Save immediately after credentials are successfully validated on the Activation screen.

Mobile Workflow Exception Handling

Describes how to handle a blocked mobile workflow.

If a mobile workflow is not received or processed on a device, this may indicate the mobile workflow is blocked in the message queue. By default, Unwired Server retries actions that threw recoverable exceptions every 15 minutes, and it retries actions that threw unrecoverable exceptions every 3 days. Both types will continue to retry indefinitely, unless the administrator intervenes, either by fixing the error or by unblocking the mobile workflow in the Sybase Control Center queue.

This typically indicates that a workflow operation failed with a recoverable or unrecoverable error. To resolve the situation:

1. Check the mobile workflow trace log, which is located in `<UnwiredPlatform_InstallerDir>\UnwiredPlatform\Servers\UnwiredServer\logs\WorkflowClient`, for information.

   This log tracks incoming messages, either from the client or from DCN or e-mail notifications, what Sybase Unwired Platform calls get invoked as a result, what the output is from the Unwired server, and what message it is transformed into when a response is sent back to the client.
2. Check the Workflow client trace logs:
   - 1. Log in to Sybase Control Center.

- 2. In the left pane, select **Applications**.
- 3. Select the application for which you want to view the trace logs and click **Get Trace**.

  The trace files are located in `<UnwiredPlatform_InstallerDir>\UnwiredPlatform\Servers\UnwiredServer\logs\ClientTrace`.

3. Use information in the logs to resolve the problem.
4. Use Sybase Control Center to check message queue status, and to suspend, resume, unblock, or remove items in the queue. See:

   Sybase Control Center for Sybase Unwired Platform > Deploy > Mobile Workflow Packages > Configuring a Mobile Workflow Package > Checking Mobile Workflow Users and Queues.

Unable to Deploy Workflow

Problem: When generating the Mobile Workflow package, you get an error that the Mobile Workflow package cannot be deployed similar to this:

```
============================================================
Deployment to Unwired Server
============================================================
Deploying the workflow
Unable to deploy workflow:
System.Web.Services.Protocols.SoapException: Could not find a
part of the path 'C:\Sybase\UnwiredPlatform\Servers\MessagingServer
\Data\Mobile Workflow\117_1'.
at Admin.ReplaceWorkflow(Byte[] baZippedPackage)
```

Explanation: Some software, such as Microsoft Security Essentials, locks the temp folder, or files in the temp folder when they are written by the Workflow installation routine, thus preventing the Directory.Move from succeeding.

Solution: Disable Microsoft Security Essentials.

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W