Developer Guide for Unwired Server

Sybase Unwired Platform 2.0

DOCUMENT ID: DC01333-01-0200-03
LAST REVISED: May 2011
Copyright © 2011 by Sybase, Inc. All rights reserved.
This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.
To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.
Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.
Sybase trademarks can be viewed at the Sybase trademarks page at *http://www.sybase.com/detail?id=1011207*. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.
SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.
Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.
Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.
All other company and product names mentioned may be trademarks of the respective companies with which they are associated.
Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.
Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

Introducing Custom Development for Unwired Server 1
Server API 1
Interfaces 2
Javadocs 2
Documentation Roadmap for Unwired Platform 3
Result Set Filters 7
Result Set Filter Data Flow 8
Implementing Custom Result Set Filters 8
Writing a Custom Result Set Filter 9
Deploying Custom Filters to Unwired Server 11
Validating Result Set Filter Performance 12
Filter Class Debugging 13
Enabling JPDA 13
Result Checkers 15
Implementing Customized Result Checkers 15
Writing a Custom Result Checker 16
Adding a Result Checker 19
Default Result Checker Code 22
Data Change Notification 27
Data Change Notification Data Flow 28
Data Change Notification With Payload and Without Payload 29
Performance Considerations for DCN With Payload Versus Without Payload 31
Information Roadmap for Implementing Data Change Notification 31
Server Configuration for Data Change Notification 31
MBO Development for Data Change Notification 32

Mobile Workflow Application Configuration for Data Change Notification 33
Implementing Data Change Notification 33
Invoking upsert and delete Operations Using Data Change Notification 33
Extending Data Change Notification to Mobile Workflow Clients 36
Basic HTTP Authentication 38
Non HTTP Authentication Workflow DCN Request 39
Data Change Notification Requirements and Guidelines 39
Data Change Notification Results 42
Mobile Workflow DCN Request Response 43
Data Change Notification Filters 44
Implementing a Data Change Notification Filter 45
Workflow DCN Design Approach and Sample Code 46
Comparing Workflow DCN With and Without Payload 47
Sample Java Function for Generating Workflow DCN 50
Sample ABAP Function for Generating Workflow DCN With Payload 52
Sample ABAP Function for Generating Workflow DCN Without Payload 57
Custom XSLT Transforms 63
Custom XSLT Use Cases 63
Implementing Custom Transforms 63
XSLT Stylesheet Syntax 64
XSLT Stylesheet Example 65
Index 69

Introducing Custom Development for Unwired Server

This developer guide provides information about using the Sybase® Unwired Platform Server API to customize Unwired Server. The audience is advanced developers who are familiar working with APIs, but who may be new to Sybase Unwired Platform.

This guide describes the contents of the Server API, and how you can use packages to create custom server code.

Companion guides include:

- *System Administration of the Unwired Platform*
- *Sybase Unwired WorkSpace online help*
- *Javadocs, which provide a complete reference to the APIs.*

See *Fundamentals* for high-level mobile computing concepts, and a description of how Sybase Unwired Platform implements the concepts in your enterprise.

Server API

Sybase Unwired Platform includes several interfaces that open specific features and functionality of Unwired Server for custom development. Customizing Unwired Server allows you to better control behaviors of these features.

- Result set filter – use a custom Java class to filter the rows or columns of data returned from a read operation for a mobile business object (MBO). You can write a filter to add, delete, or change columns, or to add and delete rows.

  Result set filters depend on the `sup-ds.jar` file, located in `\UnwiredServer\lib\ext\`.
- Result checkers – use the custom Java class to implement custom error checking for enterprise information system (EIS) business objects.

  Result checkers depend on the `sup-ds.jar` file, located in `\UnwiredServer\lib\ext\`.
- Data change notifications (DCNs) – use an HTTP interface to immediately propagate EIS changes to Unwired Server, rather than using the built-in cache refresh mechanism configured by the Unwired Server administrator.

  DCN requests contain the changed data (delta) of the EIS for which the Unwired Server cache needs to be updated. This request can be in the standard JSON format that Unwired Server expects or in a different format along with a translation logic to convert it into the standard format. This translation logic is coded in a DCN Filter.

- Custom transforms – create a transform to modify the structure of generated Web Services message data, so it can be used by an Unwired Platform MBO.

You can program these functions in any order; each class is implemented independently.

Interfaces

There are several Server API interfaces that developers can invoke.

Interface	Includes methods that
com.sybase.uep.eis.ResultSetFilter	Define how a custom filter for the data is called, and perform the filtering of data.
com.sybase.uep.eis.ResultSetFilterMetaData	Obtain output column and datatype information without executing a chain of mobile business object operations and filters with real data.
com.sybase.sup.ws.rest.RestResultChecker	Implement a result checker for a RESTful Web service datasource.
com.sybase.sup.ws.soap.WSResultChecker	Implement a result checker for a SOAP Web service datasource.
com.sybase.sup.sap.SAPResultChecker	Implement a result checker for a SAP® datasource.
com.sybase.sup.server.dcn.DCNFilter	Preprocess – digests the DCN request as blob, converts it into a valid JSON DCN request format and returns the DCN. PostProcess – takes the DCN result in a valid JSON format, converts to the EIS format and returns it.

For details on these classes, and the methods that implement them, see the Javadocs for `com.sybase.sup.admin.client`.

Javadocs

The Server API installation includes Javadocs. Use the Sybase Javadocs for your complete API reference.

As you review the contents of this document, ensure you review the reference details documented in the Javadoc delivered with this API. By default, Javadocs for Result Set Filters, Result Checkers, and Data Change Notifications are installed in `<UnwiredPlatform_InstallDir>\UnwiredPlatform\Servers\UnwiredServer\APIdocs\index.html`.

Documentation Roadmap for Unwired Platform

Learn more about Sybase® Unwired Platform documentation.

Table 1. Sybase Unwired Platform Documentation

Document	Description
Sybase Unwired Platform Installation Guide	Describes how to install or upgrade Sybase Unwired Platform. Check the *Sybase Unwired Platform Release Bulletin* for additional information and corrections. Audience: IT installation team, training team, system administrators involved in planning, and any user installing the system. Use: during the planning and installation phase.
Sybase Unwired Platform Release Bulletin	Provides information about known issues, and updates. The document is updated periodically. Audience: IT installation team, training team, system administrators involved in planning, and any user who needs up-to-date information. Use: during the planning and installation phase, and throughout the product life cycle.
New Features	Describes new or updated features. Audience: all users. Use: any time to learn what is available.
Fundamentals	Describes basic mobility concepts and how Sybase Unwired Platform enables you to design mobility solutions. Audience: all users. Use: during the planning and installation phase, or any time for reference.

Document	Description
System Administration	Describes how to plan, configure, manage, and monitor Sybase Unwired Platform. Use with the *Sybase Control Center for Sybase Unwired Platform* online documentation. Audience: installation team, test team, system administrators responsible for managing and monitoring Sybase Unwired Platform, and for provisioning device clients. Use: during the installation phase, implementation phase, and for ongoing operation, maintenance, and administration of Sybase Unwired Platform.
Sybase Control Center for Sybase Unwired Platform	Describes how to use the Sybase Control Center administration console to configure, manage and monitor Sybase Unwired Platform. The online documentation is available when you launch the console (**Start > Programs > Sybase > Sybase Control Center**, and select the question mark symbol in the top right quadrant of the screen). Audience: system administrators responsible for managing and monitoring Sybase Unwired Platform, and system administrators responsible for provisioning device clients. Use: for ongoing operation, administration, and maintenance of the system.
Troubleshooting	Provides information for troubleshooting, solving, or reporting problems. Audience: IT staff responsible for keeping Sybase Unwired Platform running, developers, and system administrators. Use: during installation and implementation, development and deployment, and ongoing maintenance.

Document	Description

Tutorials	Tutorials for trying out basic development functionality.
	Audience: new developers, or any interested user.
	Use: after installation.
	• Learn mobile business object (MBO) basics, and create a mobile device application:
	  • *Tutorial: Mobile Business Object Development*
	• Create native mobile device applications:
	  • *Tutorial: BlackBerry Application Development*
	  • *Tutorial: iOS Application Development*
	• Create a mobile workflow package:
	  • *Tutorial: Mobile Workflow Package Development*

Sybase Unwired WorkSpace – Mobile Business Object Development	Online help for developing MBOs. Audience: new and experienced developers. Use: after system installation.
Sybase Unwired WorkSpace – Mobile Workflow Package Development	Online help for developing mobile workflow applications. Audience: new and experienced developers. Use: after system installation.
Developer guides for device application customization	Information for client-side custom coding using the Client Object API. Audience: experienced developers. Use: to custom code client-side applications. • *Developer Guide for BlackBerry* • *Developer Guide for iOS* • *Developer Guide for Mobile Workflow Packages* • *Developer Guide for Windows and Windows Mobile*

Document	Description
Developer guide for Unwired Server side customization – *Developer Guide for Unwired Server*	Information for custom coding using the Server API. Audience: experienced developers. Use: to customize and automate server-side implementations for device applications, and administration, such as data handling. Dependencies: Use with *Fundamentals* and *Sybase Unwired WorkSpace – Mobile Business Object Development*.
Developer guide for system administration customization – *Developer Guide for Unwired Server Management API*	Information for custom coding using administration APIs. Audience: experienced developers. Use: to customize and automate administration at a coding level. Dependencies: Use with *Fundamentals* and *System Administration*.

Result Set Filters

A result set filter is a custom Java class an experienced developer writes in order to specifically manipulate the rows or columns of data returned from a read operation for an MBO.

When a read operation returns data that does not completely suit the business requirements for your MBO, you can write and add a filter to the MBO to customize the data into the form you need.

You can chain multiple filters together. Multiple filters are processed in the order they are added, each applying an incremental change to the data. Consequently, Sybase recommends that you always preview the results, taking note that the MBO has a different set of attributes than it would have had directly from the read operation. You can map and use the altered attributes in the same way you would do so for a regular attribute from an unfiltered read operation.

Example: a simple SELECT statement filter

Suppose you have an MBO based on this query, and you do not want fname and lname divided between two columns:

```
SELECT * FROM sampledb.customer
```

Instead, write a filter that replaces these columns with a single concatenated "commonName" column.

Note: You could also implement the above example with a more advanced SQL statement with additional computation in the MBO definition:

```
SELECT id, commonName=fname+' '+lname, address, city, state,
zip, phone, company_name FROM customer
```

Example: two separate data sources filter

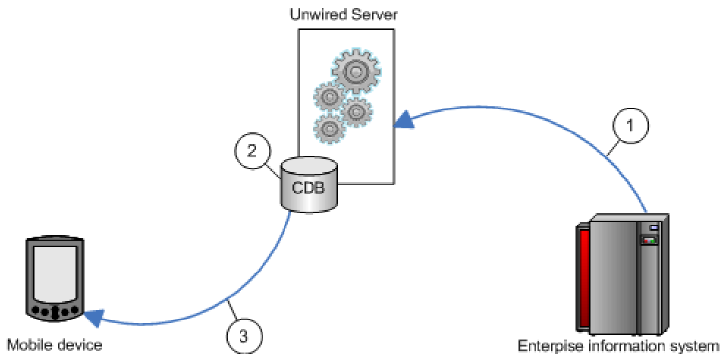
Suppose you have customer data in two data sources: basic customer information is in an SAP repository, and more complete details are contained in another database on your network, for example, SQL Anywhere™. You can use a result set filter to combine the SAP customer data with detailed customer data from the database, so that the MBO displays a complete set of information in a single view. You can accomplish this by:

1. Creating a filter for the SAP backend and add it to an SAP MBO.
2. Add a JDBC connection for the SQL Anywhere backend in the filter, then use the SQL Anywhere data to filter the SAP result.
3. Validate the results are what you expect upon completion. When you synchronize the SAP MBO, you should see data from both SAP backend and SQL Anywhere backend.

Result Set Filter Data Flow

A `ResultSetFilter` is a custom Java class deployed to Unwired Server that manipulates rows and columns of data before synchronization.

Result set filters are more versatile (and more complicated to implement) than an attribute filter implemented through a synchronization parameter, since you must write code that implements the filter, instead of simply mapping a parameter to a column to use as the filter. See *Developers Reference: Server API*.



1. Enterprise information system (EIS) data is sent to Unwired Server.
2. The result set filter filters the results, and applies those results to the CDB for a given MBO. For example, the result set filter combines two columns into one.
3. The device application synchronizes with the results contained in the CDB.

   The client cannot distinguish between MBOs that have had their attributes transformed through a `ResultSetFilter` from those that have not.

Implementing Custom Result Set Filters

Developers can write a filter to add, delete, or change columns as well as to add and delete rows.

Prerequisites

To write a filter, developers must have previous experience with Java programming — particularly with the reference implementations for `javax.sql.RowSet`, which is used to implement the filter interface and described in the *JDBC RowSet Implementations Tutorial*.

Note: Sybase strongly encourages developers to initially create filters in Unwired WorkSpace: a wizard assists you by autogenerating required imports, and methods correctly generated so the implementation already compiles and runs. Then to customize the code, you

can cut and paste fragments from the sample, and make the required changes to get the desired end result.

Task

Once the filter has been implemented and deployed to the server, the mobile business object (MBO) developer can use the filter created from Unwired WorkSpace. See *Sybase Unwired WorkSpace - Mobile Business Object Development > Develop > Developing a Mobile Business Object > Binding Mobile Business Objects to Data Sources > Adding a Result Filter > Deploying Result Filter Classes to Unwired Server*.

Note: Validate the performance of any custom result set filters, before deploying packages to Unwired Server.

Task

1. *Writing a Custom Result Set Filter*

   Write a custom result set filter to define specific application processing logic. Save the compiled Java class file to location that is accessible from Unwired WorkSpace.
2. *Deploying Custom Filters to Unwired Server*

   Deploy custom filters as part of a deployment unit.
3. *Validating Result Set Filter Performance*

   After you deploy the filters to Unwired Server, synchronize data and ensure that filters are performing as you expect.

Writing a Custom Result Set Filter

Write a custom result set filter to define specific application processing logic. Save the compiled Java class file to location that is accessible from Unwired WorkSpace.

In the custom filter, configure attribute properties so that the returned record set can be better consumed by the device client application. Sometimes, a result set returned from a data source requires unique processing; a custom filter can perform that function before the information is downloaded to the client.

Data in the cache is shared by all clients. If you need to identify data in the cache to a specific client, you must define a primary key attribute that identifies the client (such as remote_id or username).

1. (Required) Create a record set filter class that implements the `com.sybase.uep.eis.ResultSetFilter` interface.

   This interface defines how a custom filter for the data is called.

   For example, this code fragment sets the package name and imports the required classes:

```
package com.mycompany.myname;
import java.sql.ResultSet;
import java.util.Map;
```

2. (Recommended) Implement the `com.sybase.uep.eis.ResultSetFilter` and `com.sybase.uep.eis.ResultSetFilterMetaData` interface on your filter class as required by your business requirements.

   If you choose to implement this interface, you must instead execute a chain of mobile business object operations and filters with real data before you can get actual results of the output columns and their datatypes. This can impact information on the data source, which may eventually need to be reverted. By first implementing these interfaces, the operation does not need to be executed first. Instead, the `getMetaData` obtains the necessary column or data type information.

   This example sets the package name but uses a different combination of classes than in the example for step 1:

```
package com.mycompany.myname;
import java.sql.ResultSetMetaData;
import java.util.Map;
```

3. Call the appropriate method, which depends on the interfaces you implement.

   `ResultSetFilter` filters the data in the first option documented in step 1. Each filter defines a distinct set of arguments. Therefore, use only the arguments with the appropriate filter that defines these arguments in `getArguments()`, rather than use all filters and data source operations.

   The result set passed in contains the grid data, which should be considered read-only—do not use operations that change or transform data.

```
public interface ResultSetFilter {
   ResultSet filter(ResultSet in, Map<String, Object> arguments)
throws
     Exception;
   Map <String, Class> getArguments();
}
```

   Next, use `ResultSetFilterMetaData` to format the data from step 1. Use this interface to avoid executing an extraneous data source operation to generate a sample data set.

```
public interface ResultSetFilterMetaData {
  ResultSetMetaData getMetaData(ResultSetMetaData in, Map<String,
      Object> arguments) throws Exception;
}
```

Note: If the filter returns different columns depending on the argument values supplied, the filter may not work reliably. Ensure that any arguments that affect metadata have constant values in the final mobile business object definition, so the schema does not dynamically change.

4. Implement the class you have created, defining any custom processing logic.
5. Save the classes to an accessible Unwired WorkSpace location. This allows you to select the class, when you configure result set filters for your mobile business object.
6. In Unwired WorkSpace, refresh configured MBO attributes, to see the result.

MBO load operations can take parameters on the enterprise information system (EIS) side. These load parameters are defined from Unwired WorkSpace as you create the MBO. For example, defining an MBO as:

```
SELECT * FROM customer WHERE region = :region
```

results in a load parameter named "region".

As an example, if you want a filter that combines fname and lname into commonName, add `MyCommonNameFilter` to the MBO. When `MyCommonNameFilter.filter()` is called, the "arguments" input value to this method is a `Map<String, Object>` that has an entry with the key "region". Your filter may or may not care about this parameter (it is the backed database that requires the value of region to execute the query). But your filter may need some other information to work properly, for example the remote user's zipcode. The `ResultSetFilter` interface includes `java.util.Map<java.lang.String,java.lang.Class> getArguments()` that you must implement. In order to arrange for the remote user's zipcode (as a String) to be provided to the filter, write some custom code in the body of the `getArguments` method, for example:

```
public Map<String, Object> getArguments {
       HashMap<String, Class> myArgs = new HashMap<String, Class>();
       myArgs.put("zipcode", java.lang.String.class);
       return myArgs;
    }
```

This informs Unwired WorkSpace that the "zipcode" parameter is required, and is of type String. Unwired WorkSpace automatically adds the parameter for the load operation, so this MBO now has two (region and zipcode). Your filter gets them both when its `filter()` method is called, but can ignore region if it wants.

See also

- *Deploying Custom Filters to Unwired Server* on page 11

Deploying Custom Filters to Unwired Server

Deploy custom filters as part of a deployment unit.

There are two methods that are supported.

1. Create a JAR of the class.
2. Deploy the JAR, by packaging it in a deployment unit using either:

- Unwired WorkSpace development tooling. See either *Unwired WorkSpace - Developing Mobile Business Objects > Packaging and Deploying Mobile Business Objects*.
- The Deploy command line utility. See *System Administration of the Unwired Platform > System Reference > Command Line Utilities > Unwired Server Runtime Utilities > Package Administration Utilities > Deploy Application Package (deploy) Utility*.

The packaged classes are copied to `<UnwiredPlatform_InstallDir>\UnwiredPlatform\Servers\UnwiredServer\deploy\sup\<deployment-packageName>\lib` by the tool you use. In this case, the deployed package automatically refreshes, so no server restart is required.

See also

- *Writing a Custom Result Set Filter* on page 9
- *Validating Result Set Filter Performance* on page 12

Validating Result Set Filter Performance

After you deploy the filters to Unwired Server, synchronize data and ensure that filters are performing as you expect.

1. Confirm that the columns appear correctly after the filter has been added to the mobile business object.
   a) Refresh the object.
   b) In the Properties view, select the **Attribute Mapping** tab.
   c) Verify that columns are correctly listed in the **Map to** column.
2. From the device client or the device simulator, open the mobile object, and check that the new column appears.
3. Synchronize the object from the device client or simulator.
4. Troubleshoot filters if issues arise:
   - During synchronization, all `System.out` statements are printed to the Unwired Server log.
   - If you started Unwired WorkSpace with the -consoleLog in java.exe, `System.out` statements are also printed to the console window.

See also

- *Deploying Custom Filters to Unwired Server* on page 11

Filter Class Debugging

Sybase Unwired Platform supports various debugging models: instrumented code, and JPDA (Java Platform Debugger Architecture).

You can also instrument code by including **System.out.println()** in the filter class, output from the class is captured in the Unwired Server log when the filter is being executed by the server.

Alternatively, you can use the standard Java debugger to debug the filter class.

Enabling JPDA

To enable JPDA for Unwired Platform debugging, the Unwired Server needs to be started in JPDA mode.

This task describes how to setup JPDA and attache the Java standard debugger to Unwired Server.

Alternatively, you can enable Eclipse debugging in Unwired WorkSpace by first setting up a project and switching to the Debug perspective. Within the filter source code, set breakpoints from the context menu in the default Java editor. Then, with the breakpoints in place, a debugging session can be created. When this is completed, double-click the remote Java application of the Debug Configurations wizard and configure the connection type as:

- use a standard connection (Socket Attach)
- use host 0.0.0.0
- set the the port to matches the one enabled in Unwired Server (by default 5005)

1. Change to the `<UnwiredPlatform_InstallDir>\Servers\UnwiredServer\bin`.
2. By default JPDA connects over port 5005. Change the port by running **djc-setenv.bat** from the same folder and issuing this command:
   ```
   set DJC_JPDA_PORT=5005
   ```
3. Start Unwired Server in JPDA mode . How to do this varies, depending on whether or not Unwired Server is installed as a service:
   - If Unwired Server is not a service, run:
     ```
     start-unwired-server.bat -jpda
     ```
   - If Unwired Server is installed as a Windows service:
     1. Remove the service:
        ```
        sup-server-service.bat remove
        ```
     2. Recreate the service to run in JPDA mode:
        ```
        sup-server-service.bat install -jpda
        ```
4. Once Unwired Server is restarted, verify that JPDA mode is working by running:
   ```
   netstat -an | grep 5005
   ```

Look for these results:

```
TCP  0.0.0.0:<JPDAport>   0.0.0.0:0  LISTENING
```

5. Use a standard Java debugger and attach it to Unwired Server by specifying the correct host and the JPDA port used.

Begin debugging the result filter class with the Java debugger.

Result Checkers

Use the custom Java class to implement custom error checking.

A custom result checker can throw errors for both a scheduled cache refresh as well as an on demand cache refresh:

- For a scheduled refresh – the result checker writes a log message that describes the nature of the error to the Unwired Server log. As a consequence of this error, the transaction for the entire cache group is rolled back. The device client user is not notified of these errors; no client log records are generated.
- On demand refresh – instead of writing the error to the server log, the log message is written to the Unwired Server. Services in the server handle the exception. As a consequence of this error, the transaction for the cache group is rolled back. But in this case, a client log record is generated, which is visible to the client application after synchronization.

Both cases send the `OperationStatusEvent`. This event indicates that an operation failed to execute properly. The server uses `OperationStatusEvent` to populate a statistics repository that tracks the success or failure of EIS operation invocations. An administrator can review these statistics in Sybase Control Center, by clicking the Monitor node in the left navigation pane. See *System Administration of the Unwired Platform > System Maintenance and Monitoring > Status and Performance Monitoring > Reviewing System Monitoring Data*.

Implementing Customized Result Checkers

Implement a custom result checker with the required Java class to implement custom error checking for EIS-specific business objects.

1. *Writing a Custom Result Checker*

   A result checker is a custom Java class that implements error checking for mobile business objects (MBOs).
2. *Adding a Result Checker*

   Add a result checker when you edit Attribute or Operation properties for a mobile business object derived from a data source. Add a result checker after you have either written a custom one or use a predefined one in Unwired WorkSpace (the latter of which can be configured when you create an object).

Writing a Custom Result Checker

A result checker is a custom Java class that implements error checking for mobile business objects (MBOs).

Not all MBO operations use a "standard" error reporting technique; you may want to implement your own custom result checker. Doing so allows you to check any field for errors, or implement logic that determines what constitutes an error, and the severity of the error. The error code and message can be influenced in the result checker by throwing a `DSException`

1. Provide a Java class that implements the appropriate interface.

<table>
<tr><th>Data source</th><th>Interface</th></tr>
<tr><td>SAP</td><td>

```
package com.sybase.sup.sap;
public interface SAPResultChecker
{
    /**
     *
     * @param f - JCO function that has already been
executed.
     * Use the JCO API to retrieve returned values and
determine if the RFC has executed
     * successfully.
     * @return a single Map.Entry. The boolean "key"
value should be set to true if the
     * RFC is deemed to have succeeded. Normal result
processing will ensue.<P>
     * If the String value is not empty/null, that
value will be treated as a warning message,
     * which will be logged on the server,
     * and returned as a warning in transaction logs
to the client.<P>
     * Set the key value to false if it is deemed the
RFC has failed. The String value will
     * be thrown in the body of an exception. The error
will be logged on the server, and the
     * client will receive a transaction log indicat-
ing failure, including the string value.
     */
    Map.Entry<Boolean, String> checkReturn(JCO.Func-
tion f);
}
```

</td></tr>
</table>

<table>
<tr><th>Data source</th><th>Interface</th></tr>
<tr><td>Web service (SOAP)</td><td>

```
package com.sybase.sup.ws.soap;
public interface WSResultChecker
{
    /**
     * @param is the method for passing a parameter, 
and does not support setting a
       default value.
     * @param response - the SOAP Envelope response 
from a Web service execute.
     * Use the SOAP API to retrieve values and deter-
mine if the SOAP request
     * has executed successfully.
     * @return a single Map.Entry. The boolean "key" 
value should be set to true if the
     * SOAP request is deemed to have succeeded. Nor-
mal result processing will ensue.<P>
     * If the String value is not empty/null, that 
value will be treated as a warning message,
     * which will be logged on the server,
     * and returned as a warning in transaction logs 
to the client.<P>
     * Set the key value to false if it is deemed that 
SOAP has failed. The String value will
    * be thrown in the body of an exception. The error 
will be logged on the server, and the
     * client will receive a transaction log indicat-
ing failure, including the string value.
     */
    Map.Entry<Boolean, String> checkReturn(jav-
ax.xml.soap.SOAPEnvelope response);
}
```

</td></tr>
</table>

<table>
<tr><th>Data source</th><th>Interface</th></tr>
<tr><td>RESTful Web service</td><td><pre>
package com.sybase.sup.ws.rest;

import java.util.List;
import java.util.Map;

public interface RestResultChecker
{
    /**
     * REST Result Checker.
     *
     * @param responseBody HTTP response body.
     *
    * @param responseHeaders HTTP response headers in
the form
     * {{header1,value1}, {header2,value2}, ...}.
     *
     * @param httpStatusCode HTTP status code.
     *
     * @return Single Map.Entry whose boolean "key"
value is true if the
     * HTTP request succeeded, after which normal re-
sult processing will
     * ensue.<P>
     *
     * If the String value is not empty/null, that
value will be treated
    * as a warning message which will be logged on the
server and returned
    * as a warning in the transaction log sent to the
client.<P>
     *
    * Set the key value to false if it is deemed that
the service has failed.
     * The String value will be thrown in the body of
an exception. The error
     * will be logged on the server, and the client
will receive a transaction
     * log indicating failure, including the string
value.
     **/
    Map.Entry<Boolean, String> checkReturn(String
responseBody,
            List<List<String>> responseHeaders, int
httpStatusCode);
}
</pre></td></tr>
</table>

When using the Server API, result checkers depend on the `sup-ds.jar` file, in `\UnwiredServer\lib\ext\`.

Result checkers depend on the `sup-ds.jar` file, in `com.sybase.uep.tooling.api/lib` subdirectory. For example, `C:\Sybase`

```
\UnwiredPlatform-1_5\Unwired_WorkSpace\Eclipse
\sybase_workspace\mobile\eclipse\plugins
\com.sybase.uep.tooling.api_1.5.0.200909281740\lib
```

2. Save any classes you create to an accessible Unwired WorkSpace location. This allows you to select the class when you configure the result checker for your mobile business object.

See also

- *Adding a Result Checker* on page 19

Adding a Result Checker

Add a result checker when you edit Attribute or Operation properties for a mobile business object derived from a data source. Add a result checker after you have either written a custom one or use a predefined one in Unwired WorkSpace (the latter of which can be configured when you create an object).

1. In the New Attributes or New Operation wizard, in the Result checker section, select from these options:

Option	Description

Default	The result checker depends on the data source type:
	• SAP – `com.sybase.sup.sap.DefaultSAPResultCh-eck`. If a RETURN parameter is found in the selected operation, this option is automatically selected.
	• Web service (SOAP) – `com.sybase.sup.ws.soap.De-faultWSResultCheck`. The default checker always returns the status as successful.
	`DefaultWSResultCheck Passed.`
	• Web service (RESTful) – `com.sybase.sup.ws.rest.De-faultRestResultCheck`. The default checker always returns the status as successful.
	`DefaultRestResultCheck Passed.`

None	Return the status as successful with no message. The result checker used depends on the data source type: • SAP – `com.sybase.sup.sap.NoOpSAPResultCheck` • Web service (SOAP) – `com.syb-ase.sup.ws.soap.NoOpWSResultCheck` • Web service (RESTful) – `com.sybase.sup.ws.rest.NoO-pRestResultCheck`
Custom	Specify a custom result checker.

2. (Optional) If you have not yet created the result checker classes, select **Custom** in the Result checker area of the New Attributes or New Operation dialog, and click **Create** to run the New Java Class wizard.
3. If prompted, add a Java nature.
   a) (Recommended) Click **Yes** to add a Java nature. In Eclipse, a Java nature adds Java-specific behavior to projects.

   In the New Java Class wizard, enter:

Option	Description
Source folder	By default, this is the `Filters` folder from your project. Click **Browse** to locate the source folder for the Java class.
Package	Click **Browse** to locate the package for the new Java class. **Note:** Sybase recommends that you do not leave this field blank. Otherwise, the JDK 1.4 Java class in the default package cannot be resolved in other packages.
Enclosing type	Choose a type in which to enclose the new class. You can select either this option or the Package option, above. Enter a valid name or click **Browse**.
Name	Enter a name for the result checker class.
Modifiers	Select the Java class modifiers. The default modifier is public.
Superclass	**1.** Click **Browse**. **2.** In the Superclass Selection dialog, enter: • Choose a Type • Matching Items **3.** Click **OK**.

Option	Description
Interfaces	By default, this is populated with the corresponding interface: • SAP – `com.sybase.sup.sap.SAPResultChecker` • Web service (SOAP) – `com.sybase.sup.ws.soap.WSResultChecker` • RESTful Web services – `com.sybase.sup.ws.rest.ResultChecker` Click **Add** to select interfaces implemented by the new class.
Which Method Stubs Would You Like to Create	• **Public Static Void Main** • **Constructors From Superclass** • (Default) **Inherited Abstract Methods**
Do You Want to Add Comments	Select **Generate Comments** to add comments. From here, you can modify the preferences of the code templates by clicking **Configure templates and default values**.

b) Click **No** if you do not want to add the Java nature to the selected mobile application project.

c) Click **Finish** to compile the java skeleton source file and add the skeleton Java checker class to the MBO.
The result checker appears next to the Custom option.

4. In the Result checker section, next to the Custom option, click **Browse** to find an existing result checker class.

a) In the Select Result Checker Class dialog, select the result checker class and click **OK**.

The result checker class appears next to the Custom option.

5. Validate the result checker:

a) To reuse input values you have already saved for previous previews, select **Existing Configuration**. Otherwise, load defaults, or create a new set of input values expressly for this preview instance.

b) Click **Preview**.

If the data runs successfully, `Execution Succeeded` appears at the top of the Preview dialog and data appears in the **Preview Result** window.

See also

- *Writing a Custom Result Checker* on page 16

Default Result Checker Code

This result checker is a default result checker and is used to check results in SAP data sources.

```
package com.sybase.sap;

import java.util.AbstractMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import com.sap.mw.jco.JCO;
import com.sybase.sup.sap.SAPResultChecker;
import com.sybase.vader.utils.logging.SybLogger;

public class DefaultSAPResultCheck implements SAPResultChecker
{
    private static Set<String>              nonErrorMessages;
    static
    {
        nonErrorMessages = new HashSet<String>();
        nonErrorMessages.add("No data found");
        nonErrorMessages.add("Data was not found for the document");
    }

    public Map.Entry<Boolean, String> checkReturn(JCO.Function f)
    {
        JCO.Record returnStructure = null;
        JCO.ParameterList jpl = f.getExportParameterList();
        String msg = null;
        boolean success = true;
        if ( jpl != null )
        {
            try
            {
                returnStructure = jpl.getStructure("RETURN");
                if ( returnStructure != null )
                {
                    String type = returnStructure.getString("TYPE");
                    // generally TYPE is S for success, I for
informational,
                    // or empty
                    if ( !(type.equals("") || type.equals("S") ||
type.equals("I")) )
                    {
                        String message =
returnStructure.getString("MESSAGE");
                        /*UWPLogger.LogWarning*/
                       SybLogger.warn("SapUtils.execute: TYPE: <<" +
type + ">>, MESSAGE: <<" + message + ">>");
```

```
                        if ( !type.equals("W") && !
nonErrorMessages.contains(message) )
                        {
                            success = false;
                       msg =  "TYPE: <<" + type + ">>, MESSAGE: <<" +
message + ">>";
                        }
                        else
                        {
                       msg =  "TYPE: <<" + type + ">>, MESSAGE: <<" +
message + ">>";
                        }
                    }
                    else
                    {
                        if (SybLogger.isDebugEnabled())
                        {
                            String message =
returnStructure.getString("MESSAGE");
                       SybLogger.debug("SapUtils.execute: TYPE: <<"
+ type + ">>, MESSAGE: <<" + message + ">>");
                        }
                    }

                }
            }
            catch (Exception e)
            {
                /*
                if (UWPLogger.isTrace())
                    UWPLogger.LogTrace
                */
             SybLogger.debug("SapUtils::execute: Unable to retrieve
RETURN structure - Will try to retrieve RETURN table next.", e);
            }
        }
        // if there is no RETURN structure, look for RETURN table
        if ( returnStructure == null )
        {
            jpl = f.getTableParameterList();
            if ( jpl != null )
            {
                try
                {
                    StringBuilder retMessage = new StringBuilder();
                    JCO.Table returnTable = jpl.getTable("RETURN");
                  for (int i = 0; i < returnTable.getNumRows(); i++)
                    {
                        returnTable.setRow(i);
                        String type = returnTable.getString("TYPE");
                        // generally TYPE is S for success, I for
                        // informational, or empty
                      if ( !(type.equals("") || type.equals("S") ||
type.equals("I")) )
                        {
                            String message =
```

```
returnTable.getString("MESSAGE");
                            /*UWPLogger.LogWarning*/
SybLogger.warn("SapUtils.execute[" + i + "]: TYPE: <<" + type + ">>,
MESSAGE: <<"
                                    + message + ">>");
                            if ( !type.equals("W") && !
nonErrorMessages.contains(message) )
                            {
                                success = false;
                                retMessage
                                    .append("[" + i + "]TYPE: <<" +
type + ">>, MESSAGE: <<" + message + ">>");
                            }
                            else
                            {
                                retMessage
                                    .append("[" + i + "]TYPE: <<" +
type + ">>, MESSAGE: <<" + message + ">>");
                            }
                        }
                        else
                        {
                            if (SybLogger.isDebugEnabled())
                            {
                                String message =
returnTable.getString("MESSAGE");
                           SybLogger.debug("SapUtils.execute[" + i +
"]: TYPE: <<" + type + ">>, MESSAGE: <<" + message + ">>");
                            }
                        }

                    }
                    if( retMessage.length() > 0 )
                    {
                        msg = retMessage.toString();
                    }
                }
                catch (Exception e)
                {
                    /*UWPLogger.LogWarning*/
SybLogger.warn("SapUtils::execute: error in execution while
retrieving RETURN table: ", e);
                    success = false;
                    msg = e.toString();
                }
            }
        }

        return new CheckReturnMapEntry<Boolean, String>(success,
msg);
    }

    class CheckReturnMapEntry<Boolean, String> extends
java.util.AbstractMap.SimpleImmutableEntry<Boolean, String> {

        public CheckReturnMapEntry(Boolean success, String msg) {
```

```
            super(success, msg);
        }
    }

}
```

Data Change Notification

Data change notification (DCN) is an update mechanism that fully reconciles enterprise information system (EIS) data changes with Unwired Server over an HTTP or HTTPS connection using JavaScript Object Notation (JSON).

To implement DCN from the EIS to the device, there are two changes that must be reconciled: EIS to Unwired Server (DCN), and the client database with the Unwired Server cache (synchronization).

Both replication- and message-based clients support DCN:

- **Regular DCN (DCN) –** provides DCN to replication-based synchronization (RBS) and message-based synchronization (MBS) clients
- **Workflow DCN (WF-DCN) –** provides DCN to workflow clients by extending regular DCN for MBS clients

All DCN commands support both GET and POST methods. The EIS developer creates and sends a DCN to Unwired Server through HTTP GET or POST operations. The portion of the DCN command parameters that come after `http://host:8000/dcn/DCNServlet`, can all be in POST; any *var=name* can be in either the URL (GET) or in the POST. The HTTP POST method is more secure than HTTP GET methods; therefore, Sybase recommends that you include the authenticate.password parameter in the POST method, as well as any sensitive data provided for attributes and parameters.

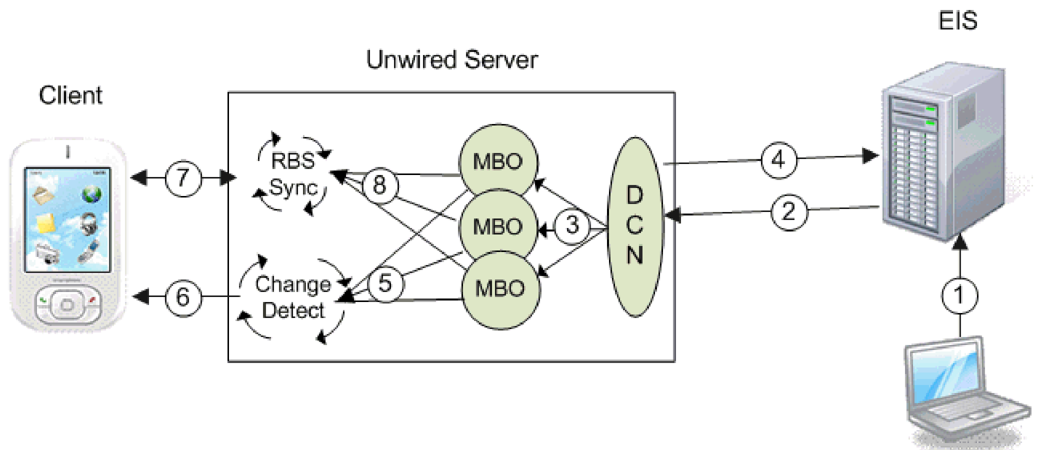
You must be familiar with the EIS data source from which the DCN is issued. You can create and send DCNs that are based on:

- Database triggers
- EIS system events
- External integration processes

Data Change Notification Data Flow

Replication-based synchronization (RBS) data flow for regular DCN differs slightly from message-based synchronization (MBS).

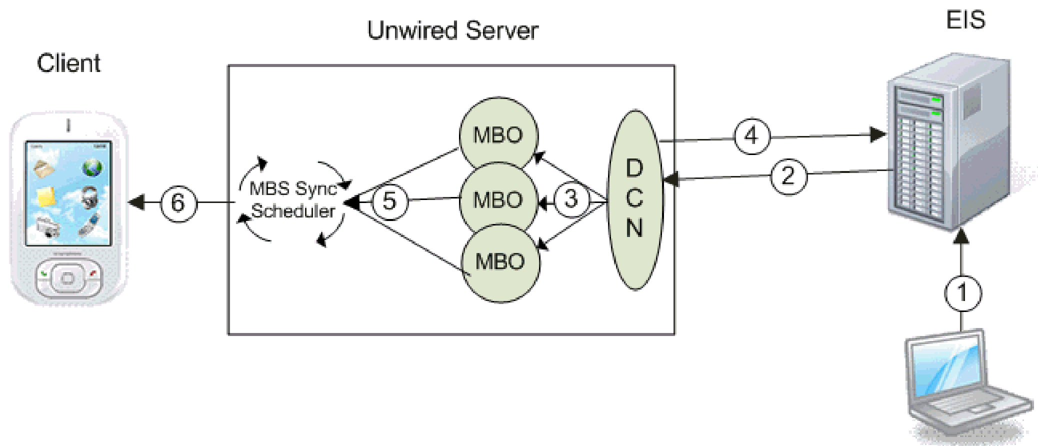
RBS DCN data flow:



Steps one through four describe DCN, while steps five through eight provides an example of how the change itself is synchronized with the client using server-initiated synchronization with an RBS client using synchronization parameters.

1. EIS update – a program or some other process updates data in the EIS which is associated with a DCN.
2. HTTP(S) push – the EIS pushes a DCN message with new or changed MBO data contained in the message on the configured HTTP(S) port.
3. DCN operation – the DCN service receives the message and implements the upsert/update function into the CDB tables of the corresponding MBOs. DCN upsert/delete operations also sets the changed flag of an MBO package to true.
4. Unwired Server response – Unwired Server sends a response message back to the EIS that contains the status of each DCN in the submitted message.
5. Change detection – for a particular device that returns new data, Unwired Server generates a message for the client indicating that it should synchronize.
6. Change notification – the message is pushed from Unwired Server to the device.
7. RBS synchronization – the client receives the message and issues a synchronization request to Unwired Server.
8. Synchronization – Unwired Server retrieves the new/updated data based on the client synchronization parameter and returns it to the client.

MBS DCN data flow:



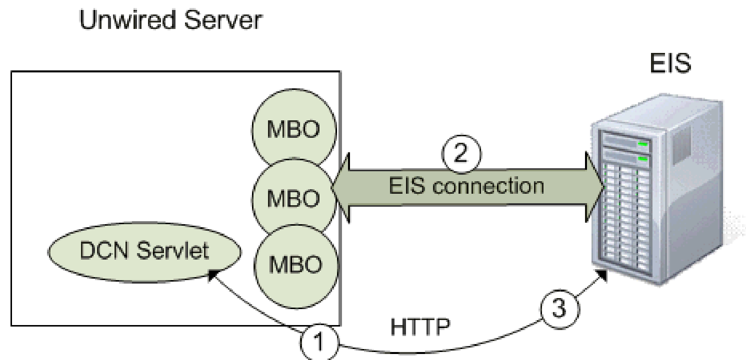
1. **EIS update** – a program or some other process updates data in the EIS which is associated with a DCN.
2. **HTTP(S) push** – the EIS pushes a DCN message with new or changed MBO data contained in the message on the configured HTTP(S) port.
3. **DCN operation** – (with payload) the DCN service receives the message and implements the upsert/update function into the CDB tables of the corresponding MBOs. DCN upsert/ delete operations also sets the changed flag of an MBO package to true.
4. **Unwired Server response** – Unwired Server sends a response message back to the EIS that contains the status of each DCN in the submitted message.
5. **synchronization triggered** – Unwired Server checks the changed flag status for the package when the synchronization group schedule time is reached. If the flag is true, Unwired Server retrieves a list of all subscribed devices for this package and synchronizes. If synchronization logic for a particular device returns new data, an MBS message is generated for that device.
6. **MBS push** – the MBS message is pushed from Unwired Server to the device.

   When the client receives this message, it performs the corresponding create, update, or delete operation on the device's mobile database.

Data Change Notification With Payload and Without Payload

Understand the differences between DCN with payload and DCN without payload.

- DCN without payload – calls MBO operations, where the name used in the DCN request matches that of the MBO definition.

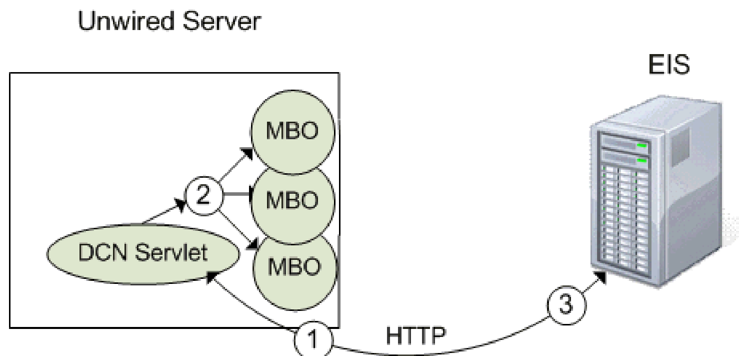


1. The DCN requester sends an MBO operation execution request, along with operation parameters, to Unwired Server.
2. Unwired Server executes the operation, (effectively calling the EIS operation), and updates the consolidated database (CDB), if needed, depending on the operation's cache update policy.
3. Unwired Server returns a DCN status message to the requester.

Note: Be careful when naming MBO operations in Unwired WorkSpace, for example, an EIS-affecting operation named "delete" may be easily confused with the direct cache-affecting operation named **:delete**.

- DCN with payload – calls only the two direct cache-affecting operations (**:upsert** or **:delete**), which always exist for an MBO, and are not related to user-defined MBO operations.
  - :upsert – the message must contain name/value pairs for every required attribute, and the name must exactly match the MBO attribute name.
  - :delete – provide only the name/value pairs for the primary key column(s).

  These operations respectively insert or update, or delete a row in the CDB. Calling either of these operations does not trigger any other refresh action. A typical sequence of events might be:



1. Some event initiates the DCN request (a database trigger for example).
2. The Unwired Server cache could be updated directly from the EIS, or the DCN request could originate from a source other than the EIS. The actual data (payload) is applied to the cache, through either an **:upsert** (update or insert) or a **:delete** operation.
3. Unwired Server returns a DCN status message to the requester.

Performance Considerations for DCN With Payload Versus Without Payload

Performance is one factor to consider when determining whether to implement DCN with or without payload.

If your application requires business data details from the EIS, including those business objects inside your DCN message reduces the number of calls made to the EIS. In these cases, DCN with payload generally provides better performance. But keep in mind that when using DCN with payload, the EIS spends more time retrieving the object data and converting it into the JSON format.

Information Roadmap for Implementing Data Change Notification

This section describes additional tasks you must perform to effectively implement DCN and WF-DCN in an end-to-end unwired enterprise environment.

Server Configuration for Data Change Notification

Sybase Control Center online help and the System Administration Guide contains information about setting up Unwired Server to accept DCN requests from the EIS,

configuring authentication and authorization, and monitoring DCN statistics and performance.

Goal	Procedures required to achieve the goals
Understand DCN's affect on Unwired Server cache	*Cache Data Management* in the *System Administration Guide*
Configuring security profiles for secure DCN communication	*Configuring Security Profiles* in Sybase Control Center online help While DCN requests can be sent to Unwired Server via HTTP, Sybase recommends configuring an HTTPS port for secure communication. The topic *Securing the DCN Transport* in the *Administration Workbook* provides a task flow for configuring a secure DCN connection and security profile.
Verify the DCN request has proper authorization by mapping the SUP DCN User role to a domain or package	*SUP DCN User Role* in Sybase Control Center online help
Optionally configure synchronization for message- or replication-based DCN	DCNs are frequently associated with a synchronization mechanism that allows Unwired Server to push data changes or notifications to the device application: • For replication-based synchronization see *Configuring a Synchronization Group for RBS Packages* • For message-based synchronization see *Configuring a Synchronization Group for MBS Packages* In Sybase Control Center online help.
Optionally configure and assign security configurations that support single sign-on authorization for DCN	*Stacking LoginModules in SSO Configurations* in Sybase Control Center online help
Optionally monitor DCN activity	*Checking System Statistics*, *Data Change Notification Performance Statistics*, and *Data Change Notification Statistics* in Sybase Control Center online help

MBO Development for Data Change Notification

Unwired WorkSpace online help contains details about configuring MBOs to enable DCN to refresh cached MBO data.

MBOs belong to a single cache group, although MBOs in the same project are not necessarily in the same cache group. The cache group policy determines the data refresh behavior of all

MBOs within the group. DCN can be used as the sole mechanism of refreshing cached data in Unwired Server by specifying the DCN cache refresh policy. DCN may also be used in parallel with a scheduled or on-demand cache refresh policy. See *Cache Groups*.

Mobile Workflow Application Configuration for Data Change Notification

The Developer Guide for Mobile Workflow Packages 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* in the Developer Guide for Mobile Workflow Packages.

Implementing Data Change Notification

Follow the syntax desribed in this section to implement DCN with payload, DCN without payload, and WF-DCN.

Invoking upsert and delete Operations Using Data Change Notification

Data change notifications (DCNs) with payload directly update the Unwired Server cache, either with the built-in, direct cache-affecting operations **:upsert** (update or insert), or with **:delete**.

Syntax

DCN with payload requires a JavaScript Object Notation (JSON) string (dcn_request) that contains one or more **:upsert** and **:delete** operations that are applied to the Unwired Server cache (CDB).

See *Extending Data Change Notification to Mobile Workflow Clients* and *Workflow DCN Design Approach and Sample Code* for WF-DCN differences and examples.

```
http://unwired_server_host:unwired_server_port(default 8000)/dcn/
DCNServlet
? cmd=dcn
&username=userName
&password=password
&domain=domainName
&package=unwired_server_PackageName
&dcn_request={"pkg":"dummy","messages":
[{"id":"1","mbo":"CustomerWithParam","op":":upsert","cols":
{"id":"10001","fname":"Adam"}}]}
&dcn_filter=fully_qualified_name_of_dcn_filter
```

Parameters

- **unwired_server –** Unwired Server host name to which the DCN is issued.
- **unwired_server_port –** Unwired Server port number. The default port is 8000.
- **username –** authorized Unwired Server user with permission to modify the MBO and permission to submit DCN requests (controlled by possessing the "SUP DCN User" logical role).
- **password –** authorized user's password.
- **domain –** Unwired Server domain that contains the package.
- **package –** Unwired Server package that contains the MBO. The format is package:version. For example, e2e_package:1.0.
- **dcn_request –** the JSON string that contains operation name and parameters, which must include:
  - Package name (pkg) – this package name is required to support backwards compatibility but ignored. The package value supplied in the header is the package value used by DCN.
  - A list of messages (messages). Each message includes:
    - A unique message ID (id) used to report back the status. The values provided for the "id" element of each DCN statement within a DCN request message are used only to identify the corresponding status message in the DCN response, which means you can select any value, including nonnumeric characters. Use unique values, so that responses to the correlated requests can be clearly identified.
    - Mobile business object name (mbo).
    - Operation name (op): either one of the direct cache-affecting operations (:upsert or :delete) or one of the user-defined MBO operations.

      Note: The specified cache update policy property of the operation still applies when DCN is used to invoke a user-defined MBO operation.
    - Bindings (cols): name and values of operation parameters which are mapped to MBO attributes.
- **dcn_filter –** (optional) the custom filter used to pre-process the DCN request and post-process the DCN status message. By default, Unwired Server requires the value of the dcn_request field to be a valid JSON string. A DCN filter is used to convert the dcn_request field from a client-specific format to a valid JSON string, before processing in the SUP server. The filter can also reformat the status message returned in the DCN response into a custom format defined by the user.
- **ppm –** personalization parameters (for either the server or client side) that need to be explicitly defined in the DCN request. The format must conform to the JSON messaging synchronization format, which is a Base64-encoded map of personalization parameters. For example, for runtime credentials sent via DCN, the PPM might be:

  ```
  base64encode("{\"username\":\"supAdmin\",\"password\":\"test
  \"}");
  ```

Examples

- **Upsert example with header –** this DCN contains a single `:upsert` operation that updates or inserts (upserts) data in the Unwired Server cache for the Department MBO.

```
http://dsqavm5:8000/dcn/DCNServlet?cmd=dcn&username=
supAdmin&password=s3pAdmin&package=dept:
1.0&domain=default&dcn_request=
{"pkg":"dummy","messages":
[{"id":"1","mbo":"Department","op":":upsert",
"cols":{"dept_id":"2","dept_name":"D2","dept_head_id":"501"}}]}
```

- **Upsert example without header –** this JSON string included in a DCN contains a single `:upsert` operation that updates or inserts (upserts) data in the Unwired Server cache for the Department MBO.

```
dcn_request={"pkg":“TestPackage",
"messages":
    [{"id":"1","mbo":"Department",
        "op":":upsert",
        "cols":{"DepartmentID":"3333",
                 "DepartmentName":"Test Value",
                                    "DepartmentHeadID":"501"}}]
}
```

- **Delete example with header –** this DCN example deletes a row of data from the Unwired Server cache for the Department MBO:

```
http://dspevm5:8000/dcn/DCNServlet?cmd=dcn&username=
supAdmin&password=s3pAdmin&package=dept:
1.0&domain=default&dcn_request=
{"pkg":"dummy","messages":
[{"id":"1","mbo":"Department","op":":delete",
"cols":{"dept_id":"2"}}]}
```

- **Delete example without header –** this example JSON string included in the DCN sent to Unwired Server, deletes a row of data from the Unwired Server cache for the Department MBO:

```
dcn_request={"pkg":“TestPackage",
    "messages":[{"id":"1","mbo":"Department",
"op":":delete",
    "cols":{"DepartmentID":"3333"}}]}
```

Usage

Follow these guidelines when constructing a DCN:

- For timestamp values use this format: 2009-03-04T17:12:45.

Note: Time zone information should not be included since it is ignored by the server. Convert timestamps to the corresponding UTC value before submitting them.

- The **:upsert** operation requires:

- All MBO primary key attributes to be present in the payload.
- Any other MBO attributes used in the upsert.
- All columns in the operation use attribute names (not the column names to which they are mapped).

- The **:delete** operation requires:
  - The MBO primary key attribute be present in the payload.
  - All columns in the operation use attribute names (not the column names to which they are mapped).

Extending Data Change Notification to Mobile Workflow Clients

Mobile Workflow data change notification (WF-DCN) requests enable you to process the general DCN request and send notification to the device to keep the workflow application synchronized with the enterprise information system (EIS) when a change to the EIS occurs.

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 e-mail, 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 **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.

Basic HTTP Authentication

When you use http://<host>:8000/dcn/DCNServlet, the user authentication is done by Unwired Server extracting the user information from the request parameter.

```
username=<username>
password=<password>
```

Alternatively, you can use HTTP BASIC authentication instead of sending the username and password as part of the URL. To use HTTP BASIC authentication, the URL is `http://<hostname>:<port>/dcn/HttpAuthDCNServlet`, as this example illustrates:

```
URL url = new URL("http://<host>:8000/dcn/HttpAuthDCNServlet?
cmd=dcn&package=<package_name>:<package_version>");
        HttpURLConnection huc = (HttpURLConnection)
url.openConnection();
        huc.setDoOutput(true);
        huc.setRequestMethod("POST");
        final String login = "<login_name_of_user_with_DCN_role>";
        final String pwd = "<password_of_user_with_DCN_role>";
        Authenticator.setDefault(new Authenticator()
        {
            protected PasswordAuthentication
getPasswordAuthentication()
            {
                return new PasswordAuthentication(login,
pwd.toCharArray());
            }
        });
        String dcnRequest = "{\"pkg\":
\"<package_name>:<package_version>"\","
              + "\"messages\":[{\"id\":\"1\",\"mbo\":\"CustomerState
\",\"op\":\":upsert\","
              + "\"cols\":{\"id\":\"1020\",\"fname\":\"Paul\",\"city
\":\"Rutherford\"}}]}";
        StringBuffer sb = new StringBuffer();
        sb.append(dcnRequest);
        OutputStream os = huc.getOutputStream();
        os.write(sb.toString().getBytes());
        os.close();
        BufferedReader br = new BufferedReader(new
InputStreamReader(huc.getInputStream()));
        System.out.println(huc.getURL());
        huc.connect();
        String line = br.readLine();
        while (line != null)
```

```
        {
            System.out.print(line);
            line = br.readLine();
        }
```

HTTP POST and DCN

You can also use the URL `http://<hostname>:8000/dcn/HttpAuthDCNServlet` if you do not want to send the DCN request as a request parameter but as an HTTP POST body instead.

If you are using HTTP BASIC authentication, the JSON encoded DCN request is always sent as the HTTP POST body.

Non HTTP Authentication Workflow DCN Request

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

The URL is:

```
http://host:8000/dcn/DCNServlet?
cmd=wf&security=admin&domain=default&username=supAdmin&password=s3p
Admin&dcn_filter=aa.bb&dcn_request=<wfrequestdata>
```

Data Change Notification Requirements and Guidelines

Familiarize yourself with data change notification (DCN) requirements before implementing DCN.

Personalization parameters in DCN

Personalization parameters of the MBO need to be specified separately in the **ppm** parameter. The required ppm parameter in the `dcn_request` has to be a `string` which should be a Base64-encoded map of personalization parameters. This example shows how you must use `ppmString` to define the value for **ppm** parameter in the `dcn_request`:

```
Map<String, String> ppm = new HashMap<String, String>();
        ppm.put("myCompany", "Sybase");
        String ppmString =
Base64Binary.toString(gson.toJson(ppm).getBytes());
```

DCN upsert operations and MBO relationships

When using the DCN payload mode to upsert rows to MBOs where there is a relationship between rows of data, you must provide the data in the correct order so Unwired Server can properly create the metadata in the cache (CDB) to reflect the data relationship. However, when you are using DCN to insert data into the cache, the concept of child and parent may be different from what is reflected in the graphical model of the package used in the design tooling. Also, one-to-many relationships differ, as noted below.

When using DCN to upsert rows to both the parent and child MBOs in a relationship, the order for the upserts can change depending on the nature of the relationship. This is due to the implementation details of the cache metadata. In these examples, the Department MBO is the parent MBO in both relationships, but notice the order of the upsert operations:

- For a one-to-one relationship between:

  ```
  Dept.dept_head_id - > Employee.emp_id
  ```

  (from a department to the department head) the order in which you upsert a new department and new department head is:
  1. Employee
  2. Department

  The foreign surrogate key reference is contained in the cache table used to hold the data for the Department MBO.
- For a one-to-many relationship between:

  ```
  Dept.dept_id - > Employee.dept_id
  ```

  (from a department to all of the employees in the department) the order in which you upsert a new department and a new employee is:
  1. Department
  2. Employee

  The foreign surrogate key reference is contained in the cache table used to hold the data from the Employee MBO.

Message autonomy

Unwired Server expects serialized DCN message updates to MBO instances. Therefore if concurrent DCN clients or processes are used, insure that all updates to all rows of MBO(s) are contained within a single DCN request in order to avoid a possible deadlock condition.

Unwired Server expects an entire graph when sending updates to MBOs within a composite relationship.

DCN upsert operations and binary data

When using DCN to upsert binary data to the cache (CDB), the string used for the value of the binary type attribute of the MBO in the request message must conform to a very specific encoding for the DCN request to be processed correctly. Read the binary data into a byte array, then use the following code to obtain it in the correctly encoded format:

```
byte[] picByteArray = < < user code to read binary data into byte[] >
>
String picStringBase64Encoded =
com.sybase.djc.util.Base64Binary.toString(picByteArray);
String picStringUrlEncoded =
java.net.URLEncoder.encode(picStringBase64Encoded, "UTF-8");
```

Use the **picStringUrlEncoded** string as the value for the binary attribute in the DCN request message.

Note: The com.sybase.djc.util.Base64Binary class is in the sup-server.jar from the `C:\Sybase\UnwiredPlatform\Servers\UnwiredServer\lib` directory of the SUP installation.

DCN and date, time, and datetime datatypes

DCN accepts date, time, and datetime attribute and parameter values using this format:

- `date` – yyyy-MM-dd
- `time` – HH:mm:ss
- `datetime` – yyyy-MM-dd'T'HH:mm:ss

For example, Unwired Server parses `string` or `long` values and upserts a valid timestamp object:

```
http://localhost:8000/dcn/DCNServlet?
cmd=dcn&username=supAdmin&password=
s3pAdmin&package=testdatetime:1.0&domain=default&dcn_request=
{"pkg":"testdatetime","messages":
[{"id":"1","mbo":"TestDateTimeStamp","op":":upsert",
"ppm":null,"cols":
{"testTimestamp":"2009-08-09T12:04:05","testDate":"2009-08-09","c_i
nt":"0",
"testDateTime":"2009-08-09T12:04:05","testSmalldt":"2009-08-09T12:0
4:05","testTime":"12:04:05"},
}]}
```

Complex types

Special care must be taken when using DCN to populate MBOs which form the sub-types of a larger complex type. For example a PurchaseOrder type is composed of POHeader and POLineItem MBOs. To successfully populate the rows of the POHeader and POLineItem MBOs with data derived from a PurchaseOrder, the DCN code must properly set the primary key attributes and insert the rows in the correct order so that the relationship between rows from the POHeader and the POLineItem MBOs is successfully defined. See also the "DCN upsert operations and MBO relationships" topic.

DCN and Cache refresh policy

If DCN is the exclusive mechanism for loading and maintaining the data for the MBOs from a cache group, set the cache refresh policy to DCN.

DCN with payload directly updates the CDB by inserting the record into the MBO cache table. Using a DCN cache group policy ensures that MBO data is updated only through DCN and not another refresh mechanism. When designing your MBOs, keep in mind that if you use other methods (for example, a Scheduled cache group policy), DCNs as well as the scheduled refresh update the MBO.

If the cache group relies on a combination of DCN with On-demand or Schedule cache refresh policies, then you cannot use DCN with MBOs that define more than one partition (that is, a

load operation mapped to synchronization parameters in Unwired WorkSpace). The cache refresh policy does not support updating data inserted via DCN when the configured cache policy triggers a refresh of the partitioned data.

If the load operation initializes the MBO, and you use DCN to maintain the MBO, then associate the MBO with a cache group that implements an infinite schedule. Do not send DCN messages until the cache is initialized.

Cache update policies and DCN

Do not use a cache update policy that invalidates the cache if you use a DCN to populate the MBO.

DCN and deadlocks

The requirements described above (*Message autonomy* and *Send DCN messages only to MBOs with load operations that do not take parameters*) are designed to prevent deadlock situations. However, if you do not define an order of operation execution, deadlocks might occur depending on the DCN implementation or the locking mechanism used by the enterprise information system (EIS). In a deadlock situation, the entire transaction is rolled back (if there are multiple operations in a single DCN) and a `replayFailed` result is returned.

Data discrepancies and deleted data

If any attribute values differ from the actual EIS values, those values are updated with the actual value when a cache refresh occurs. Any rows inserted into the cache which contain a primary key value which is not present in the EIS are marked as logically deleted when a cache refresh occurs. Once a row has been marked as logically deleted from the cache, attempts to upsert data using that same primary key value fail until the logically deleted row is purged from the cache.

Data Change Notification Results

Each binding in a data change notification (DCN) request is associated with an ID. The result status of the DCN request is returned in JavaScript Object Notation (JSON) format, and includes a list of IDs followed by a Boolean success field and status message, in case of error.

The processing of the individual messages within a DCN request is done as a single transaction. A failure of one message results in the changes from all preceding messages being rolled back and all following messages are skipped. In response to payload and MBO operation DCNs, Unwired Server sends the requester a JSON string containing details about the success and/or failure of the operations. These examples show the JSON-formatted result for a multi-message DCN request, and has been formatted using newlines, and indentations, which are not present in an actual response.

This is an example of a response message for successful processing of a request containing four individual messages, using id values {1, 2, 3, 4)

```
[{"recordID":"1","success":true,"statusMessage":""},
{"recordID":"2","success":true,"statusMessage":""},
{"recordID":"3","success":true,"statusMessage":""},
{"recordID":"4","success":true,"statusMessage":""}]
```

In this example the third message in the request contains an error:

```
[{"recordID":"1","success":false,"statusMessage":
"Changes rolled back because dcn message with ID 3 in the DCN request
failed."},

{"recordID":"2","success":false,"statusMessage":
"Changes rolled back because dcn message with ID 3 in the DCN request
failed."},

{"recordID":"3","success":false,"statusMessage":
"VirtualTableName is null. MBO name Departments might be incorrect or
with missing
capitalization in the DCN request"},

{"recordID":"4","success":false,"statusMessage":
"Processing of DCN message skipped because dcn message with ID 3 in
the DCN request failed."}]
```

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"
    }
  ]
}
```

Data Change Notification Filters

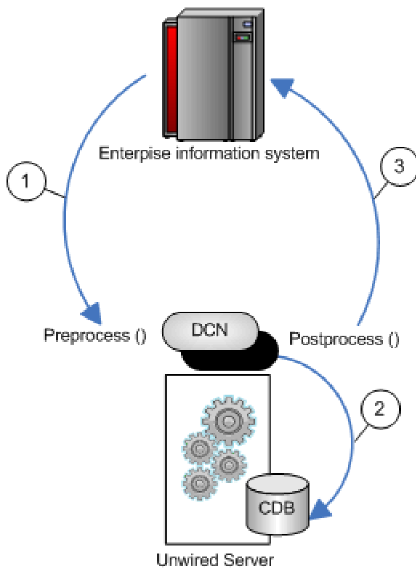
Data change notification (DCN) requests need not always be in the format Unwired Server expects.

You can deploy a DCN filter to Unwired Server and reference it in the DCN request. Unwired Server allows the filter to preprocess the submitted DCN. The filter converts raw data in the DCN request to the required JavaScript Object Notation (JSON) format. The filter can also postprocess the JSON response returned by the Unwired Server into the format preferred by the back end (which is governed by the implementation in the filter class).

The filter interface `DCNFilter` is in the `com.sybase.sup.server.dcn` package in the `sup-server-rt.jar` file. All classes that implement a DCN filter should implement this interface. The functions available in the interface are:

- **String preprocess(String blobDCNRequest, Map<String, String requestHeaders> requestHeaders);** – takes the DCN request as a binary large object (BLOB), converts it into a valid JSON DCN request format, and returns the same.
- **String postprocess(String jsonDCNResult, Map<String, String responseHeaders> responseHeaders);** – takes the DCN result in a valid JSON format, converts it to the EIS-specific format, and returns the same.

Figure 1: DCN filter flow



1. Changed data is sent from the EIS to Unwired Server via a DCN request, where any data preprocessing occurs. For example, the EIS data could be sent to Unwired Server as XML where the preprocess filter converts the data to JSON.
2. The DCN executes. For example, apply data changes directly to the Unwired Server cache.
3. Postprocessed DCN response is sent to the originating EIS as an HTTP response to the original DCN request. For example, the JSON response is converted to XML.

Implementing a Data Change Notification Filter

Write and deploy preprocess and postprocess DCN and WF-DCN filters to Unwired Server.

When specifying filters, add a dcn_filter parameter to the base URL, and to the parameters specified in the DCN request section. The **dcn_filter** parameter specifies the fully qualified name of the filter class, which must be in a valid CLASSPATH location so Unwired Server can locate it using its fully qualified name.

JSON requires colons to define the object structure, but since colons have a special function in HTTP URLs, use the tilda character "~" instead of colons ":" when implementing the DCN filter, so the JSON `dcn_request` string can be passed as an HTTP GET or POST parameter:

```
dcn_request={"pkg"~"TestPackage",
    "messages"~[{"id"~"1","mbo"~"Department","op"~"~upsert",
    "cols"~{"DepartmentID"~"3333",
          "DepartmentName"~"My Department",
             "DepartmentHeadID"~"501"}}]}
```

The `dcn_request` is in a format that is specific to the back end. The filter class can preprocess to the JSON format expected by Unwired Server.

1. Write the filter. For example:

```
import java.util.Map;
import com.onepage.fw.uwp.shared.uwp.UWPLogger;
import com.sybase.sup.dcn.DCNFilter;

public class CustomDCNFilter implements DCNFilter
{
  String preprocess(String blobDCNRequest, Map<String,String>
headers) {
         String result = blobDCNRequest.replace('~',':');
         return result;
  }

  String postprocess(String jsonDCNResult, Map<String,String>
responseHeaders) {
         String result = jsonDCNResult.replace(':','~');
         return result;
  }

    public static void main( String[] args ) { }
}
```

2. Package your DCN filter class in a JAR file.
3. Deploy the JAR file to Unwired Server by using the Deployment wizard from Unwired WorkSpace:
   a) Invoke the deployment wizard. For example, right-click in the Mobile Application Diagram and select **Deploy Project**.
   b) Select the JAR file that contains your DCN filter class files to deploy to Unwired Server in the third screen of the wizard (Package User-defined Classes).
   c) Click **Finish** after selecting the target Unwired Server.

   Note: Alternatively, you can bypass step b when deploying the DCN filter class JAR file to Unwired Server:
   1. Place the JAR file in `<UnwiredPlatform_InstallDir>\UnwiredPlatform\Servers\UnwiredServer\lib\ext`.
   2. At a command prompt, run: `<UnwiredPlatform_InstallDir>\Unwired Platform\Servers\UnwiredServer\bin\configure-mms.bat <hostname>`.
4. Restart Unwired Server.

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.

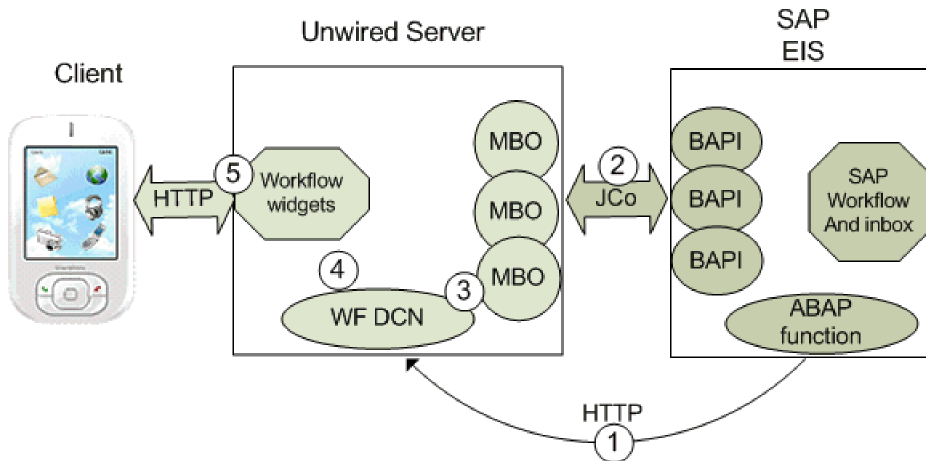
Comparing Workflow DCN With and Without Payload

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, as mobile workflow e-mails.
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 parameter to trigger an MBO data refresh.
5. The workflow engine converts MBO data and the WF-DCN message into a workflow e-mail, 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":":","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 retrives details from the EIS.

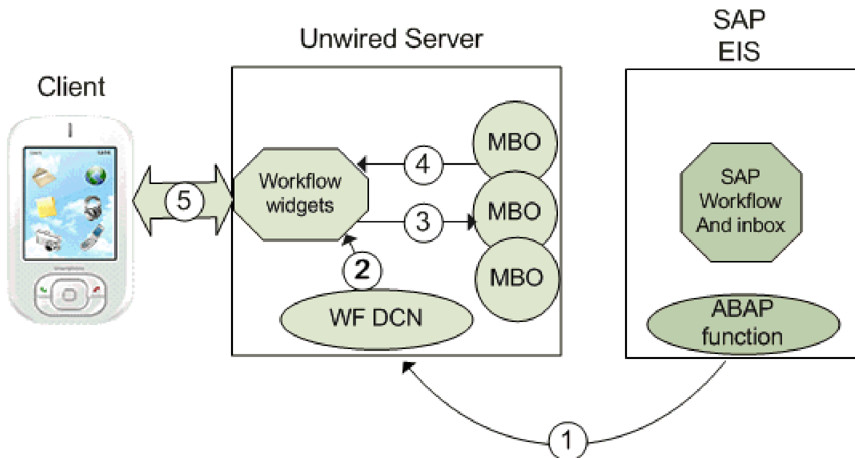
Processing the WF-DCN without payload message

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 parameter 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 e-mail, then pushes it to the device mobile inbox via the MOCA protocol.

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 package name, 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#\"}";
//            + "\"data\":[{\"id\":\"dcntest_1_data_1\",\"pkg\":"
//            + "\"wf_alrt:1.0\",\"messages\":[{\"id\":\"1\",\"mbo
```

```
\":"
//              + "\"Department\",\"op\":\":upsert\",\"cols\":
{\"dept_id\":\"2300\","
//              + "\"dept_name\":\"x\",\"dept_head_id\":\"501\"},"
//              + "\"parameters\":{}}]}]}";

        url = new URL("HTTP", "10.42.39.149", 8081,
                "/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 = "s3pAdmin";
        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);
        }

    }
}
```

Sample ABAP Function for Generating Workflow DCN With Payload

This WF-DCN sample illustrates WF-DCN with payload.

```
FUNCTION ZSUP_MWF_PUSH_WORKITEM.
*"-------------------------------------------------------------------
-----
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(DESTINATION) TYPE  ZSUP_MWF_CHAR20
*"     VALUE(QNAME) TYPE  ZSUP_MWF_CHAR20
*"     VALUE(QCOUNT) TYPE  ZSUP_MWF_INT4
*"     VALUE(SUPSECURITY) TYPE  ZSUP_MWF_CHAR255CS
*"     VALUE(ACTION) TYPE  ZSUP_MWF_CHAR1
*"     VALUE(WORKITEM) TYPE  ZSUP_MWF_CHAR32
*"     VALUE(COMPLETE) TYPE  ZSUP_MWF_CHAR1 OPTIONAL
*"     VALUE(PRIORITY) TYPE  ZSUP_MWF_CHAR1 OPTIONAL
*"     VALUE(MARKREAD) TYPE  ZSUP_MWF_CHAR1 OPTIONAL
*"     VALUE(SENDER) TYPE  ZSUP_MWF_CHAR255CS OPTIONAL
*"     VALUE(RECIPIENT) TYPE  ZSUP_MWF_CHAR20 OPTIONAL
*"     VALUE(SUBJECT) TYPE  ZSUP_MWF_CHAR255CS OPTIONAL
*"     VALUE(BODY) TYPE  ZSUP_MWF_STRINGCS OPTIONAL
*"  TABLES
*"      SENDERTEXTS STRUCTURE  ZSUP_MWF_PARAM OPTIONAL
*"      RECIPIENTS STRUCTURE  ZSUP_MWF_PARAM OPTIONAL
*"-------------------------------------------------------------------
-----

  "* if complete indicator is set, we directly generate
  "* json and execute the push to the http endpoint


  IF complete = 'X'.

    DATA lv_json_notif TYPE zsup_mwf_stringcs.
    "* generate a workitem notification body
    CALL FUNCTION 'ZSUP_MWF_GEN_NOTIF_JSON'
      EXPORTING
        notif_id = workitem
        delete = action
        markread = markread
        priority = priority
        to = recipient
        from = sender
        subject = subject
        body = body
      IMPORTING
        json_out = lv_json_notif.
```

```
  "* execute http push function module using specified destination
  CALL FUNCTION 'ZSUP_MWF_HTTP_POST'
    EXPORTING
      destination = destination
      supsecurity = supsecurity
      msg_body = lv_json_notif.

  RETURN.
ENDIF.

DATA lv_trfcqnam TYPE trfcqnam VALUE IS INITIAL.
DATA lv_xubname TYPE xubname VALUE IS INITIAL.
DATA ls_userdefault TYPE bapidefaul VALUE IS INITIAL.
DATA lv_lang_iso TYPE t002-laiso VALUE IS INITIAL.
DATA lv_wiid TYPE sww_wiid VALUE IS INITIAL.
DATA lv_wi_text TYPE witext VALUE IS INITIAL.
DATA lv_return_code TYPE sysubrc VALUE IS INITIAL.
DATA lv_subject TYPE zsup_mwf_stringcs VALUE IS INITIAL.
DATA lv_body TYPE zsup_mwf_stringcs VALUE IS INITIAL.
DATA lv_body2 TYPE zsup_mwf_stringcs VALUE IS INITIAL.
DATA lv_sender TYPE zsup_mwf_char255cs VALUE IS INITIAL.
DATA lv_sendertext TYPE zsup_mwf_param VALUE IS INITIAL.
DATA ls_recipient TYPE zsup_mwf_param VALUE IS INITIAL.
DATA lv_recipient TYPE zsup_mwf_char20 VALUE IS INITIAL.
DATA lt_message_struct TYPE TABLE OF swr_mstruc.
DATA lt_alternatives TYPE TABLE OF swr_decialts.
DATA lt_user_parameter TYPE TABLE OF bapiparam.
DATA ls_user_parameter TYPE bapiparam VALUE IS INITIAL.
DATA lt_bapiret2 TYPE TABLE OF bapiret2.

"* loop over all entries in the recipient list
"* and determine the user language, translate
"* the workitem text into the users language
"* and call the function module again with the
"* complete indicator set to push individual
"* notifications to configured destination
LOOP AT recipients INTO ls_recipient.

  IF ls_recipient-pkey = 'US'.
    "* read user defaults to determine user
    "* default language and if user is mobile
    "* enabled
    MOVE ls_recipient-pvalue TO lv_xubname.
    CALL FUNCTION 'BAPI_USER_GET_DETAIL'
      EXPORTING
        username = lv_xubname
      IMPORTING
        defaults = ls_userdefault
      TABLES
        parameter = lt_user_parameter
        return = lt_bapiret2.

    "* set output language to user default,
    "* if not set, set temporarily to default
    IF NOT ls_userdefault-langu IS INITIAL.
```

```
    SET LANGUAGE ls_userdefault-langu.
  ELSE.
    ls_userdefault-langu = sy-langu.
  ENDIF.

  "* check returned BAPIRET2 table for any error
  "* or abort entries, stops execution in case
  "* an error or abort entry is found
  PERFORM check_bapiret2 TABLES lt_bapiret2.

  "* initialize recipient name to username
  MOVE lv_xubname TO lv_recipient.

  "* check if user is mobile enabled (ZSUP_ENABLED)
  "* and set the recipient name to the mobile username
  lv_return_code = 1.
  LOOP AT lt_user_parameter INTO ls_user_parameter.
    IF ls_user_parameter-parid = 'ZSUP_MWF_ENABLED' AND
      ls_user_parameter-parva = 'X'.
        lv_return_code = 0.
    ENDIF.
  ENDLOOP.

  "* if the user is not mobile enabled
  "* continue with the next entry in the
  "* recipient list
  IF lv_return_code <> 0.
    CONTINUE.
  ENDIF.

  "* if the notification push is a delete skip the
  "* determination of further content for the push such
  "* as localized subject and sender, etc.
  IF NOT action = 'D'.

    "* translate user language code to iso
    CALL FUNCTION 'LANGUAGE_CODE_SAP_TO_ISO'
      EXPORTING
        sap_code  = ls_userdefault-langu
      IMPORTING
        iso_code  = lv_lang_iso
      EXCEPTIONS
        not_found = 1
        OTHERS    = 2.

    "* output MESSAGE to stop execution in case of
    "* an error during language code conversion
    IF sy-subrc <> 0.
      MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno.
    ENDIF.

    "* read decision in users configured default language
    MOVE workitem TO lv_wiid.
    CALL FUNCTION 'SAP_WAPI_DECISION_READ'
      EXPORTING
        workitem_id = lv_wiid
```

```
      language = ls_userdefault-langu
      user = lv_xubname
    IMPORTING
      return_code = lv_return_code
      decision_title = lv_wi_text
    TABLES
      alternatives = lt_alternatives
      message_struct = lt_message_struct.

  "* check returned MESSAGE_STRUCT table for any error
  "* or abort entries, stops execution in case
  "* an error or abort entry is found
  IF lv_return_code <> 0.
    PERFORM check_message_struct TABLES lt_message_struct.
  ENDIF.

  "* copy localized version of the decision title
  "* into local subject valiable
  MOVE lv_wi_text TO lv_subject.

  "* loop over all available configured l10n versions
  "* of the sender field and determine the match
  "* for the particular users default language or
  "* use the configured default sender text
  LOOP AT sendertexts INTO lv_sendertext.
    IF lv_sendertext-pkey CS lv_lang_iso.
      lv_sender = lv_sendertext-pvalue.
      EXIT.
    ELSEIF lv_sendertext-pkey = ''.
      lv_sender = lv_sendertext-pvalue.
    ENDIF.
  ENDLOOP.


  "* complete the fixed workitem notification body
  "* used for matching and paramater value extract
  "* in the SUP notification processing
 CONCATENATE body ', USER:' lv_xubname '*#END#*' INTO lv_body.

  DATA lv_decision TYPE ZSUP_MWF_CHAR255CS VALUE IS INITIAL.
  DATA lv_name TYPE ZSUP_MWF_CHAR20 VALUE IS INITIAL.
  DATA lv_description TYPE ZSUp_MWF_STRINGCS VALUE IS INITIAL.
  DATA lv_alter TYPE TABLE OF ZSUP_MWF_PARAM .
  DATA lv_return TYPE TABLE OF BAPIRET2.
  DATA ltater TYPE  ZSUP_MWF_PARAM VALUE IS INITIAL.
  Move ls_recipient-pvalue to lv_name.
  Call Function 'ZSUP_MWF_WI_GETDETAIL'
  Exporting
     USERNAME = lv_name
     WORKITEM = workitem
  IMPORTING
      DECISION = lv_decision
      Description = lv_description
  TABLES
    RETURN = lv_return
    ALTERNATIVES = lv_alter.
```

```
        CONCATENATE lv_body '","data":
[{"id":"'workitem'","pkg":"sup_mwf:1.2","messages":
[{"id":"'workitem'","mbo":"Workitem","op":":upsert","cols":
{"WORKITEM":"'workitem
        '","USERNAME":"'lv_xubname'","DESCRIPTION":"'lv_description'
","DECISION":"'lv_decision'"}}'Into lv_body.
        loop at lv_alter into ltater.

        Concatenate lv_body  ',
{"id":"'workitem'","mbo":"Alternatives","op":":upsert","cols":
{"WORKITEM":"'workitem
        '","USERNAME":"'lv_xubname'","PKEY":"'ltater-
PKEY'","PVALUE":"'ltater-PVALUE'"}}' Into lv_body.
        EndLoop.
        Concatenate lv_body ']}],'  INTO lv_body.

      ENDIF.

      "* if qname was not set skip
      "* and execute as tRFC
      IF qname <> ''.
        "* if qcount was specified hash username
        "* into one of the possible queues before
        "* setting the queue name for qRFC
        IF qcount > 0.
          PERFORM hash_user_to_queue USING lv_xubname qcount CHANGIN
G qname.
        ENDIF.
        MOVE qname TO lv_trfcqnam.
        CALL FUNCTION 'TRFC_SET_QUEUE_NAME'
          EXPORTING
            QNAME = lv_trfcqnam.
      ENDIF.

      "* execute push function module in the background
      "* with complete flag set
      CALL FUNCTION 'ZSUP_MWF_PUSH_WORKITEM' IN BACKGROUND TASK AS S
EPARATE UNIT
        EXPORTING
          destination = destination
          qname = qname
          qcount = qcount
          supsecurity = supsecurity
          action   = action
          workitem = workitem
          complete = 'X'
          priority = priority
          markread = markread
          sender = lv_sender
          recipient = lv_recipient
          subject = lv_subject
          body = lv_body.

    ENDIF.
```

```
  ENDLOOP.

  "* tRFC/qRFC requires explicit commit
  COMMIT WORK.

ENDFUNCTION.
```

Sample ABAP Function for Generating Workflow DCN Without Payload

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

```
FUNCTION ZSUP_MWF_PUSH_WORKITEM.
”-----------------------------------------------------------------
----
”*”Local Interface:
”  IMPORTING
”     VALUE(DESTINATION) TYPE  ZSUP_MWF_CHAR20
”     VALUE(QNAME) TYPE  ZSUP_MWF_CHAR20
”     VALUE(QCOUNT) TYPE  ZSUP_MWF_INT4
”     VALUE(SUPSECURITY) TYPE  ZSUP_MWF_CHAR255CS
”     VALUE(ACTION) TYPE  ZSUP_MWF_CHAR1
”     VALUE(WORKITEM) TYPE  ZSUP_MWF_CHAR32
”     VALUE(COMPLETE) TYPE  ZSUP_MWF_CHAR1 OPTIONAL
”     VALUE(PRIORITY) TYPE  ZSUP_MWF_CHAR1 OPTIONAL
”     VALUE(MARKREAD) TYPE  ZSUP_MWF_CHAR1 OPTIONAL
”     VALUE(SENDER) TYPE  ZSUP_MWF_CHAR255CS OPTIONAL
”     VALUE(RECIPIENT) TYPE  ZSUP_MWF_CHAR20 OPTIONAL
”     VALUE(SUBJECT) TYPE  ZSUP_MWF_CHAR255CS OPTIONAL
”     VALUE(BODY) TYPE  ZSUP_MWF_STRINGCS OPTIONAL
”  TABLES
”      SENDERTEXTS STRUCTURE  ZSUP_MWF_PARAM OPTIONAL
”      RECIPIENTS STRUCTURE  ZSUP_MWF_PARAM OPTIONAL
”-----------------------------------------------------------------
----

  “* if complete indicator is set, we directly generate
  “* json and execute the push to the http endpoint




  IF complete = ‘X’.

    DATA lv_json_notif TYPE zsup_mwf_stringcs.

    “* generate a workitem notification body
    CALL FUNCTION ‘ZSUP_MWF_GEN_NOTIF_JSON’
      EXPORTING
        notif_id = workitem
        delete = action
        markread = markread
        priority = priority
        to = recipient
        from = sender
```

```
      subject = subject
      body = body
    IMPORTING
      json_out = lv_json_notif.

  "* execute http push function module using specified destination
  CALL FUNCTION 'ZSUP_MWF_HTTP_POST'
    EXPORTING
      destination = destination
      supsecurity = supsecurity
      msg_body = lv_json_notif.

  RETURN.
ENDIF.

DATA lv_trfcqnam TYPE trfcqnam VALUE IS INITIAL.
DATA lv_xubname TYPE xubname VALUE IS INITIAL.
DATA ls_userdefault TYPE bapidefaul VALUE IS INITIAL.
DATA lv_lang_iso TYPE t002-laiso VALUE IS INITIAL.
DATA lv_wiid TYPE sww_wiid VALUE IS INITIAL.
DATA lv_wi_text TYPE witext VALUE IS INITIAL.
DATA lv_return_code TYPE sysubrc VALUE IS INITIAL.
DATA lv_subject TYPE zsup_mwf_stringcs VALUE IS INITIAL.
DATA lv_body TYPE zsup_mwf_stringcs VALUE IS INITIAL.
DATA lv_body2 TYPE zsup_mwf_stringcs VALUE IS INITIAL.
DATA lv_sender TYPE zsup_mwf_char255cs VALUE IS INITIAL.
DATA lv_sendertext TYPE zsup_mwf_param VALUE IS INITIAL.
DATA ls_recipient TYPE zsup_mwf_param VALUE IS INITIAL.
DATA lv_recipient TYPE zsup_mwf_char20 VALUE IS INITIAL.
DATA lt_message_struct TYPE TABLE OF swr_mstruc.
DATA lt_alternatives TYPE TABLE OF swr_decialts.
DATA lt_user_parameter TYPE TABLE OF bapiparam.
DATA ls_user_parameter TYPE bapiparam VALUE IS INITIAL.
DATA lt_bapiret2 TYPE TABLE OF bapiret2.

"* loop over all entries in the recipient list
"* and determine the user language, translate
"* the workitem text into the users language
"* and call the function module again with the
"* complete indicator set to push individual
"* notifications to configured destination
LOOP AT recipients INTO ls_recipient.

  IF ls_recipient-pkey = 'US'.
    "* read user defaults to determine user
    "* default language and if user is mobile
    "* enabled
    MOVE ls_recipient-pvalue TO lv_xubname.
    CALL FUNCTION 'BAPI_USER_GET_DETAIL'
      EXPORTING
        username = lv_xubname
      IMPORTING
        defaults = ls_userdefault
      TABLES
        parameter = lt_user_parameter
        return = lt_bapiret2.
```

```
"* set output language to user default,
"* if not set, set temporarily to default
IF NOT ls_userdefault-langu IS INITIAL.
  SET LANGUAGE ls_userdefault-langu.
ELSE.
  ls_userdefault-langu = sy-langu.
ENDIF.

"* check returned BAPIRET2 table for any error
"* or abort entries, stops execution in case
"* an error or abort entry is found
PERFORM check_bapiret2 TABLES lt_bapiret2.

"* initialize recipient name to username
MOVE lv_xubname TO lv_recipient.

"* check if user is mobile enabled (ZSUP_ENABLED)
"* and set the recipient name to the mobile username
lv_return_code = 1.
LOOP AT lt_user_parameter INTO ls_user_parameter.
  IF ls_user_parameter-parid = 'ZSUP_MWF_ENABLED' AND
    ls_user_parameter-parva = 'X'.
      lv_return_code = 0.
  ENDIF.
ENDLOOP.

"* if the user is not mobile enabled
"* continue with the next entry in the
"* recipient list
IF lv_return_code <> 0.
  CONTINUE.
ENDIF.

"* if the notification push is a delete skip the
"* determination of further content for the push such
"* as localized subject and sender, etc.
IF NOT action = 'D'.

  "* translate user language code to iso
  CALL FUNCTION 'LANGUAGE_CODE_SAP_TO_ISO'
    EXPORTING
      sap_code  = ls_userdefault-langu
    IMPORTING
      iso_code  = lv_lang_iso
    EXCEPTIONS
      not_found = 1
      OTHERS    = 2.

  "* output MESSAGE to stop execution in case of
  "* an error during language code conversion
  IF sy-subrc <> 0.
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno.
  ENDIF.

  "* read decision in users configured default language
```

```
    MOVE workitem TO lv_wiid.
    CALL FUNCTION 'SAP_WAPI_DECISION_READ'
      EXPORTING
        workitem_id = lv_wiid
        language = ls_userdefault-langu
        user = lv_xubname
      IMPORTING
        return_code = lv_return_code
        decision_title = lv_wi_text
      TABLES
        alternatives = lt_alternatives
        message_struct = lt_message_struct.

    "* check returned MESSAGE_STRUCT table for any error
    "* or abort entries, stops execution in case
    "* an error or abort entry is found
    IF lv_return_code <> 0.
      PERFORM check_message_struct TABLES lt_message_struct.
    ENDIF.

    "* copy localized version of the decision title
    "* into local subject valiable
    MOVE lv_wi_text TO lv_subject.

    "* loop over all available configured l10n versions
    "* of the sender field and determine the match
    "* for the particular users default language or
    "* use the configured default sender text
    LOOP AT sendertexts INTO lv_sendertext.
      IF lv_sendertext-pkey CS lv_lang_iso.
        lv_sender = lv_sendertext-pvalue.
        EXIT.
      ELSEIF lv_sendertext-pkey = ".
        lv_sender = lv_sendertext-pvalue.
      ENDIF.
    ENDLOOP.

    "* complete the fixed workitem notification body
    "* used for matching and paramater value extract
    "* in the SUP notification processing
   CONCATENATE body ', USER:' lv_xubname '*#END#*' INTO lv_body.





    TRANSLATE lv_body TO UPPER CASE.

  ENDIF.

  "* if qname was not set skip
  "* and execute as tRFC
  IF qname <> ".
    "* if qcount was specified hash username
    "* into one of the possible queues before
    "* setting the queue name for qRFC
```

```
        IF qcount > 0.
          PERFORM hash_user_to_queue USING lv_xubname qcount CHANGIN
G qname.
        ENDIF.
        MOVE qname TO lv_trfcqnam.
        CALL FUNCTION ‘TRFC_SET_QUEUE_NAME’
          EXPORTING
            QNAME = lv_trfcqnam.
      ENDIF.

      “* execute push function module in the background
      “* with complete flag set
      CALL FUNCTION ‘ZSUP_MWF_PUSH_WORKITEM’ IN BACKGROUND TASK AS S
EPARATE UNIT
        EXPORTING
          destination = destination
          qname = qname
          qcount = qcount
          supsecurity = supsecurity
          action   = action
          workitem = workitem
          complete = ‘X’
          priority = priority
          markread = markread
          sender = lv_sender
          recipient = lv_recipient
          subject = lv_subject
          body = lv_body.

    ENDIF.
  ENDLOOP.

  “* tRFC/qRFC requires explicit commit
  COMMIT WORK.

ENDFUNCTION.
```

Custom XSLT Transforms

If you are using data from a SOAP or REST Web service, you may need to use XSLT (Extensible Stylesheet Language Transformations) to modify the structure of the message data generated by the service, so it can be used by an Unwired Platform MBO. Unwired Workspace can create XSLT transforms automatically, however sometimes these generated transforms are not sufficient and do not yield the results you require.

MBOs typically require a flat and tabular message structure from a Web service. This tabular structure corresponds to the rows and columns that eventually materialize the MBO's instances and attributes, respectively. Therefore the message structure used by a Web service must align correctly. Transformation must be precise to avoid unexpected results in an MBO.

Therefore, always validate the transform before deploying it to a production environment.

Custom XSLT Use Cases

In most cases, the XSLT that is generated by Unwired WorkSpace is sufficient. However, in some cases, you may need to modify the generated XSLT file, or to create a new one manually.

Some of these cases include:

- Web service response messages do not precisely conform to the schema required by the WSDL schema.
  For example, the schema indicated that an integer field is not nullable, but the Web service response message failed to return a valid integer value. This omission triggers an error on the device application. even though the root issue is the data from the Web service, not Unwired Platform.
  In this scenario, it is simpler to modify the generated XSLT slightly, by changing the single op_nullable field from `false` to `true`.

Implementing Custom Transforms

When the generated transform does not yield expected results in the MBO, you need to either modify the generated transform or create a custom transform outside of Unwired WorkSpace.

1. Make changes to an existing transform or write a new one.
2. Save the changes and overwrite the file that already exists. This ensures that the binding remains intact for the MBO. See *Unwired WorkSpace > Develop > Developing Mobile Business Objects > Binding Mobile Business Objects to Data Sources*.
3. Redeploy the MBO so changes implemented to Unwired Server, and include the transform in the deployment package.

See *Unwired WorkSpace > Develop > Developing Mobile Business Objects > Packaging and Deploying Mobile Business Objects*.

Note: If you are redeploying to a production environment, ensure the administrator redeploys the MBO with the modified transform.

XSLT Stylesheet Syntax

XSLT stylesheet must follow Unwired Platform stylesheet syntax requirements so that the Web service response message is formatted correctly for MBOs bound to this data source.

The stylesheet is applied to different parts of the Web service response message, depending on the type:

- For SOAP web service response messages, the stylesheet is applied to the contents of the SOAP body.
- For REST web service messages, the stylesheet is applied to the contents of the HTTP response body.

Table 2. Stylesheet elements

Element	Description	Contains
`Data`	The root element of the stylesheet.	One or more `Record` elements.
`Record`	The element that corresponds to a row in the tabular MBO data structure. The first `Record` element resulting from the transformat describes the column using metadata (that is, names, data types, nullability, and so on). The `Record` element has no attributes, except when it is a metadata element. The contents of the `Field` elements should match the corresponding `op_label` values. The `Record` or `Field` values from this first Record element will not appear in the resulting tabular data structure.	One or more `Field` elements.
`Field`	The element that corresponds to the column value. The `Field` element has a number of attributes that can be used.	One or more attributes. See the *Attributes* table.

Table 3. Attributes

Attribute	Applicability	Description
`op_label`	Required by the stylesheet and the resulting transformed structure's metadata. Ignored by the data field elements.	The column name.
`op_position`	Required by all.	The attribute's position in the tabular structure. The first attribute is at position 1
`op_nullable`	Required by the stylesheet and the resulting transformed structure's metadata. Ignored by the data field elements.	Whether (TRUE) or not (FALSE) the attribute is nullable.
`op_datatype`	Required by the stylesheet and the resulting transformed structure's metadata. Ignored by the data field elements.	The data type. Supported values inlclude STRING, INT, LONG, BOOLEAN, DECIMAL, BINARY, FLOAT, DOUBLE, DATE, TIME, DATETIME, CHAR, BYTE, SHORT, INTEGER. See *Sybase Unwired WorkSpace - Mobile Business Object Development > Develop > Working with Mobile Business Objects > Mobile Business Object Data Properties > Datatype Support*.
`op_xsdtype`	Required by the stylesheet and the resulting transformed structure's metadata. Ignored by the data field elements.	The XML schema primitive type name corresponding to this attribute.

XSLT Stylesheet Example

Use the example XSLT stylesheet to understand the structure required by Unwired Platfrom.

The bolded elements are required. The `<xsl:stylesheet>` needs a `<xsl:template>` element. The first child element of `<xsl:template>` must be the `<data>` that also requires the a metadata `<Record>` element.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"; xmlns:ns1="urn:Sample_Enrollments" exclude-result-
```

```
prefixes="ns1">
        <xsl:template match="//ns1:OpGetListResponse">
                <data>
                        <Record>
                                <Field op_label="Class_Cost"
op_position="1" op_datatype="DECIMAL"
op_nullable="false">Class_Cost</Field>
                                <Field op_label="Class_ID"
op_position="2" op_datatype="STRING" op_nullable="false">Class_ID</
Field>
                           <Field op_label="Class_Start_Date___Time"
op_position="3" op_datatype="DATETIME"
op_nullable="false">Class_Start_Date___Time</Field>
                                <Field op_label="Class_Title"
op_position="4" op_datatype="STRING"
op_nullable="false">Class_Title</Field>
                                <Field op_label="Enrollee_Login"
op_position="5" op_datatype="STRING"
op_nullable="false">Enrollee_Login</Field>
                                <Field op_label="Temp_Number"
op_position="6" op_datatype="INT" op_nullable="true">Temp_Number</
Field>
                        </Record>
                        <xsl:for-each select="ns1:getListValues">
                                <Record>
                                        <Field>
                                                <xsl:attribute
name="op_position">1</xsl:attribute>
                                                <xsl:value-of
select="ns1:Class_Cost"/>
                                        </Field>
                                        <Field>
                                                <xsl:attribute
name="op_position">2</xsl:attribute>
                                                <xsl:value-of
select="ns1:Class_ID"/>
                                        </Field>
                                        <Field>
                                                <xsl:attribute
name="op_position">3</xsl:attribute>
                                                <xsl:value-of
select="ns1:Class_Start_Date___Time"/>
                                        </Field>
                                        <Field>
                                                <xsl:attribute
name="op_position">4</xsl:attribute>
                                                <xsl:value-of
select="ns1:Class_Title"/>
                                        </Field>
                                        <Field>
                                                <xsl:attribute
name="op_position">5</xsl:attribute>
                                                <xsl:value-of
select="ns1:Enrollee_Login"/>
                                        </Field>
                                        <Field>
```

```
                                                <xsl:attribute
name="op_position">6</xsl:attribute>
                                                <xsl:value-of
select="ns1:Temp_Number"/>
                                        </Field>
                                </Record>
                        </xsl:for-each>
                </data>
        </xsl:template>
</xsl:stylesheet>
```

If you use this style sheet, the output generated by this transform would be:

```
<data>
        <Record>
                <Field op_label="Class_Cost" op_position="1"
op_datatype="DECIMAL" op_nullable="false">Class_Cost</Field>
                <Field op_label="Class_ID" op_position="2"
op_datatype="STRING" op_nullable="false">Class_ID</Field>
                <Field op_label="Class_Start_Date___Time"
op_position="3" op_datatype="DATETIME"
op_nullable="false">Class_Start_Date___Time</Field>
                <Field op_label="Class_Title" op_position="4"
op_datatype="STRING" op_nullable="false">Class_Title</Field>
                <Field op_label="Enrollee_Login" op_position="5"
op_datatype="STRING" op_nullable="false">Enrollee_Login</Field>
                <Field op_label="Temp_Number" op_position="6"
op_datatype="INT" op_nullable="true">Temp_Number</Field>
        </Record>
        <Record>
                <Field op_position="1">100.00</Field>
                <Field op_position="2">00001</Field>
                <Field op_position="3">2010-07-02T10:27:35-07:00</
Field>
              <Field op_position="4">Managing Within the Law</Field>
                <Field op_position="5">Demo</Field>
                <Field op_position="6"/>
        </Record>
        <Record>
                <Field op_position="1">150.00</Field>
                <Field op_position="2">00005</Field>
                <Field op_position="3">2005-11-17T08:00:00-08:00</
Field>
               <Field op_position="4">Microsoft Word for Beginners</
Field>
                <Field op_position="5">Demo</Field>
                <Field op_position="6"/>
        </Record>
        <Record>
                <Field op_position="1">299.00</Field>
                <Field op_position="2">00006</Field>
                <Field op_position="3">2005-11-15T08:00:00-08:00</
Field>
                <Field op_position="4">Meeting Planning and
Facilitation</Field>
                <Field op_position="5">Demo</Field>
```

```
                <Field op_position="6"/>
        </Record>
</data>
```

Index

C

companion documentation 1
custom development features 1
custom filters
    See result set filters

D

data change notification 39
    filters 44
    GET 36
    JSON format 36
    POST 36
    request response 43
data change notification filter
    example 45
    implementing 45
data change notification interface 27
data change notification parameters 33
data change notification syntax 33
data change notification with payload 33
data change notification, results 42
DCN 43
DCN task flow 31
documentation roadmap
    document descriptions 3

F

filters
    data change notification 44
    result set 13

G

guide, introducing 1

H

HTTP interface for data change notification 27

I

Implementing DCN 31
interfaces 2
introduction 1

J

Javadocs 2
JPDA
    enabling 13

M

messages, transforming 64

N

non HTTP authentication request 39

P

parameters, data change notification
    dcn_request 33
    domain 33
    package 33
    password 33
    unwired_server 33
    unwired_server_port 33
    username 33

R

response messages, transforming 64
REST
    transforming data
        See Transforms
result checker 19
    default SAP code 22
result checker, customizing 16
result checker, implementing 15
result checkers 15
result set filters 8, 11
    debugging 13
    deploying 11

S

SAP result checker 15
server API features 1
single sign-on task flow 32, 33
SOAP
    transforming data
        See Transforms
stylesheet syntax, XSLT 64
syntax, XSLT 64

T

transforms
    custom, introducing 63
    implementing 63
    stylesheet example 65
    stylesheet syntax 64
    when to use custom files 63

W

Web services
    transforming data
        See Transforms

X

XSLTs
    See Transforms