

HP Operations Orchestration

for the Windows and Linux operating systems

Software Version: OO Content Pack 7

Web Services Wizard Guide

for the Windows operating system

Document Release Date: January 2012

Software Release Date: January 2012



Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© Copyright 2012 Hewlett-Packard Development Company, L.P.

Trademark Notices

For information on open-source and third-party software acknowledgements, see *Open-Source and Third-Party Software Acknowledgements* (3rdPartyOpenNotices.pdf) in the documentation set for this OO 9.00 release.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

<http://h20230.www2.hp.com/selfsolve/manuals>

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Web site at:

www.hp.com/go/hpsoftwaresupport

This Web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support Web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

- 1 Introduction 6**
 - Overview of the Web Services Wizard.....7
 - Downloading OO Releases and Documents on HP Live Network.....7

- 2 Wizard Processing Details..... 9**
 - How the Web Services Wizard Uses soapUI.....10
 - Processing Templates.....10
 - Locating Inputs and Creating the inputMap.....10
 - Locating Outputs and Creating Operation Outputs13
 - Populating InvokeMethod2 Default Values for All Operations.....14

- 3 The Invoke Method 2 Operation..... 15**
 - Overview of the Invoke Method 2 Operation16
 - Building a SOAP Request16
 - Complete Set of Inputs17

- 4 Using the Web Services Wizard Installer..... 20**
 - About the Web Services Wizard Installer21
 - System Requirements.....21
 - Running the Web Services Wizard Installer21
 - Web Services Wizard Code Dependencies.....22
 - Configure Logging Settings.....22
 - Uninstalling the Web Services Wizard23

- 5 Using the Web Services Wizard to Create Web Services Flows 24**
 - Using the Web Services Wizard to Create OO Flows from Selected WSDL Operations25
 - After Running the Web Services Wizard31

- 6 Troubleshooting..... 32**
 - General Troubleshooting Principles33
 - Troubleshooting Steps33

1 Introduction

This section includes the following topics:

- [Overview of the Web Services Wizard](#)
- [Downloading OO Releases and Documents on HP Live Network](#)

Overview of the Web Services Wizard

When you run the Web Services Wizard (wswizard.exe), you provide it with the WSDL for a Web service. The Web Services Wizard creates OO flows based on the API described in the Web Service Definition Language (WSDL) of the Web service that you identify in the wizard. The WSDL string you provide as a pointer can be a file's location and name or a URL.

The Web Services Wizard helps you create OO flows when:

- An OO integration does not exist.
- An OO integration does exist, but the customer has modified the application. For example, a customer using Remedy may have modified a form or added a field. To take advantage of the customer's modifications, the Remedy Web Service is updated. You can use the Web Services Wizard to create OO flows from the modified Web service.
- If a new version of an application with an OO integration comes out and the integration content does not support the new version, you can use the Web Services Wizard to create new OO flows.

Example

Suppose you have an application called **MyAlert** that creates a ticket through a Web service and API, and you want to tell **MyAlert** to create a ticket. The Web Services Wizard extracts, from the Web service's WSDL, the application's APIs for the actions that can be performed with the application, such as creating or changing a ticket. The WSDL defines the Web service's methods, the inputs that each method needs, and the required format for each input.

When you provide the wizard with the WSDL (in our example, for **MyAlert**) and run the wizard, it generates flows that can run against the Web service. All flows created with the Web Services Wizard have a single step which is built from the **Invoke Method 2** operation in the Library/Operations/Wizards/Web Services Wizard/ folder. The flows are created in the repository specified by the user in the **Library/Wizards/Web Services Wizard/<ServiceName>/** folder.

Running the flows requires a Remote Action Service (RAS) that has access to the Web service. For information on creating and configuring RAS references, see "Operating outside Central with Remote Action Services" in the Guide to Authoring Operations Orchestration Flows.

Downloading OO Releases and Documents on HP Live Network

HP Live Network provides an **Operations Orchestration Community** page where you can find and download supported releases of OO and associated documents.

To download OO releases and documents, visit the following site:

<https://hpln.hp.com/>

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

<http://h20229.www2.hp.com/passport-registration.html>

Or click the **New users - please register** link on the HP Passport login page.

On the **HP Live Network** page, click **Operations Orchestration Community**.

The Operations Orchestration Community page contains links to announcements, discussions, downloads, documentation, help, and support.

1. On the left-hand side, click **Operations Orchestration Content Packs**.
2. In the **Operations Orchestration Content Packs** box, click **Content**. The HP Passport and sign-in page appears.
3. Enter your user ID and Password to access to continue.
4. Click **HP Operations Orchestration 9.00**.
5. Search for HP Operations Orchestration Content Pack 7

2 Wizard Processing Details

This section includes the following topics:

- [How the Web Services Wizard Uses soapUI](#)
- [Processing Templates](#)
- [Locating Inputs and Creating the inputMap](#)
- [Locating Outputs and Creating Operation Outputs](#)
- [Populating InvokeMethod2 Default Values for All Operations](#)

How the Web Services Wizard Uses soapUI

soapUI is an open-source Web service testing tool. Its functionality includes Web service inspection, invoking, development, and simulation. The Web Services Wizard uses soapUI to parse the WSDL and create a template SOAP request (OO Content Pack 7 now uses SoapUI version 4.0.1). This template is an XML with placeholder tokens that are replaced with real data in order to make a request to the server. If you run soapUI manually and create a project referencing a WSDL, you will see it create these request templates in the tree as nodes called **Request 1** for every operation in the WSDL. This is the template that the Web Services Wizard gets from soapUI, and it uses to populate the **xmlTemplate** input.

In a similar way, the Web Services Wizard (in OO versions 9.00 and later) retrieves a SOAP response template with tokens that indicate what the response will look like. This is a little more difficult to reproduce in the soapUI GUI, but it requires creating a Mock Response and then using the Open Editor function to look at the XML.

For OO 9.0, support was added to specify a Web proxy via the properties **http.proxyHost** and **http.proxyPort** in the `wsw.properties` file in the OO Home folder in `/Studio/tools/conf/`. This configuration is now read from this file and prepopulated in the wizard GUI because this information need only be entered once (the first time you run the wizard against a WSDL outside the firewall). You may change it in the file or the wizard GUI and the values are saved for the next time you run the wizard.

After retrieving the templates for the request and the response, the WSDL is basically discarded. No further information is obtained from the WSDL, and all subsequent logic in both the Web Services Wizard and the **Invoke Method 2** operation is based entirely on the templates returned from soapUI.

Processing Templates

The template processing logic parses through a SOAP template (either a request template or a response template) looking for tokens. It is called in different ways for different purposes — for processing the request template and for processing the response template:

- 1 Locating input tokens in the request template to create the input map (in the wizard).
- 2 Locating output tokens in the response template (in the wizard).
- 3 Replacing input tokens with actual values to build the SOAP request (in the **Invoke Method 2** operation).

In all cases, the logic skips past any leading xml elements until it finds an element whose namespace prefix is either **soap** or **soapenv** and whose element name is not **envelope**, and then begins with the content of that element; this effectively ends up arriving at the topmost element under the outermost **Body** element.

Locating Inputs and Creating the inputMap

In the wizard, the request template is processed, and for each token that is found, a pipe-delimited value is returned indicating its path in the template, but with the outermost SOAP

envelope information removed. For example, if the template looks like this:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <Name?</Name>
      <Address?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>
```

it returns the values **Test|Name** and **Test|Address**. Note that the whole path is needed, since it is possible that an element (like **Name**) can appear in more than one place in a template, and there needs to be a unique path to each.

If during this input processing the wizard encounters a comment that indicates that it is at the beginning of an array ("x or more repetitions" or "m to n repetitions"), the value zero (0) is inserted at that point. For example:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <!--1 or more repetitions-->
      <Name?</Name>
      <!--1 or more repetitions-->
      <Address?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>
```

returns the values **Test|0|Name** and **Test|0|Address**. Since arrays may be nested, there may be templates whose values contain more than one zero (0).

The next task is to figure out a meaningful set of input names that should be created. This is accomplished using an input map. An input map permits a user-friendly name to be associated with each value. For example, **Address** is mapped to **Test|Address** and **Name** is mapped to **Test|Name**. The **inputMap** input that is generated in the operation is a list of these mappings between pipe-delimited paths and user-friendly names. In the first example above, the **inputMap** would contain:

Test|Name=Name

Test|Address=Address

The creation of the **inputMap** is somewhat complicated. When determining which user-friendly name to assign to each path name, this is the basic process:

- Use the last portion of the path (for example, **Name** or **Address**) if it is unique within the template.
- Avoid using a friendly name that is already one of the input names to the **Invoke Method 2** operation, such as **xmlTemplate**.

- If there are duplicate names, prepend additional levels (with a period separator) onto the user-friendly name until they are unique. For example, if the template yielded **One | Name** and **Two | Name**, the following input map would be created:

One | Name=One . Name

Two | Name=Two . Name

since both would otherwise map to the same value of **Name**.

- Single zeros in the pipe-delimited path (indicating the beginning of an array) are replaced with wildcards (*). This is a new feature with OO 9.0 in order to avoid having to make manual edits to the **inputMap** when dealing with arrays. The position of the wildcard in the user-friendly name is moved to the end of the next element. In the above example, **Test | 0 | Name** and **Test | 0 | Address**, the following input map would be created:

Test | * | Name=^Name*\$

Test | * | Address=^Address*\$

Note:

- The purpose of shuffling the wildcard position is to allow more intuitive input names like **Name0** and **Name1**.
- The value on the right side of the equal sign for array types is surrounded by the ^ and \$ symbols as a workaround for resolving the issue of parameters having similar names. These values are used as regex patterns for array types and similarly-named parameters without these symbols corrupting the algorithm.
- The simplification of friendly names (see the first bullet in this list) only applies to the portion of an array to the left of the wildcard; all elements to the right will remain. For example, the items **Test | 0 | Extra | Stuff | Name** and **Test | 0 | Extra | Stuff | Address** will result in:

Test | * | Extra | Stuff | Name=^Extra* . Stuff . Name\$

Test | * | Extra | Stuff | Address=^Extra* . Stuff . Address\$

in spite of the fact that the **Extra** and **Stuff** are otherwise unnecessary.

The wizard then uses the **inputMap** to create step-level and flow-level inputs for each item in the map. Any occurrences of wildcards are replaced with zeros in the input names. If the flow developer wants to provide additional elements (beyond just the 0th), then he would need to add them both as step level inputs and flow level inputs. Using our previous example:

Test | * | Name=^Name*\$

Test | * | Address=^Address*\$

Name0 and **Address0** will be created as inputs to the step and the flow.

The Web Services Wizard now accepts JSON formatted arrays for the array types that were found in the WSDL. So, instead of entering a new input for each element in the array, you can now enter a JSON-formatted array as the input value instead of creating additional inputs.

When you run the Web Services Wizard, you must check the **Use JSON arrays for WSDL array type** option on the **Select operation(s)** screen. This will add the input field "usesJSON" with a value of "true" to the created Invoke Method 2 step. Then for the inputs, use a JSON format array for the "0" element and the Invoke Method 2 operation to create the required elements to send in the request.

For example, for an array structure defined by the following in the xmlTemplate:

```

<ns:AffectedCI type="Array">
  <!--Zero or more repetitions:-->
  <ns:AffectedCI type="String" mandatory=""
  readonly=""></ns:AffectedCI>
</ns:AffectedCI>

```

- The inputMap entry for this array must use the following wildcard format:
CreateChangeTaskOORequest|model|instance|middle|AffectedCI|*|AffectedCI=^AffectedCI*\$
- The associated Web Services Wizard created AffectedCI0 input field JSON array formatted value would be something similar to:
["C1value1" ,"C1value2" ,"C1value3"]

Locating Outputs and Creating Operation Outputs

Locating outputs in the XML template uses the same logic as finding inputs, but instead of returning a pipe-delimited path, the process returns an XML XPath expression. This is nearly the same thing except with a slash as a delimiter rather than a pipe. There are, however a few differences:

- /text() is appended to the XPath in order to properly extract the text of the simple elements. For example, the following template:

```

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <Name>?</Name>
      <Address>?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>

```

corresponds to the outputs **/Test/Name/text()** and **/Test/Address/text()**.

- Nothing is appended to the XPath of array elements. Downstream, this will cause the entire portion of the XML document to be returned in a single output, and it will be left to the flow developer to use other operations (like XML or JSON ones) to extract the items of interest. This difference is due to the fact that arrays can become arbitrarily nested, and returning such structured data in a simple variable is not an easy task. For example, the following template

```

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <!--1 or more repetitions-->
      <Name>?</Name>

```

```
        <!--1 or more repetitions-->
        <Address>?</Address>
    </Test>
</soapenv:Body>
</soapenv:Envelope>
```

yields just the single output `/Test`.

If JSON arrays are being used, an additional output named `jsonStripped` will be populated with the SOAP response in a JSON-formatted string.

The wizard then creates step outputs for each output that was located in the template, assigning an XPath filter to each one (whose value was determined above). At this point the wizard has completed its main lifting. The remainder of the process resumes when the flow is run, calling the **Invoke Method 2** operation.

Populating InvokeMethod2 Default Values for All Operations

The Web Services Wizard allows setting **InvokeMethod2** inputs so that each operation created from the WSDL would have the inputs set by default. For example, the **timeout** input might be the same for all Web service operations and setting the value once in the wizard will, in turn, set the timeout input value for all operation(s) selected on the selection page. The setting of the default values in the Web Services Wizard is completely optional.

The Web Services Wizard will not perform any validation of the default inputs entered. This validation will still take place during the actual running of the flow(s). The Web Services Wizard will however, limit you to only specifying default values for the authentication type selected. For example, if the **http** authentication type is selected, the wizard only allows you to enter the default inputs for HTTP authentication and skips over the WS-Security page when you click the **Next** button.

3 The Invoke Method 2 Operation

This section includes the following topics:

- [Overview of the Invoke Method 2 Operation](#)
- [Building a SOAP Request](#)
- [Complete Set of Inputs](#)

Overview of the Invoke Method 2 Operation

The **Invoke Method 2** operation is called when the flow is run. Its basic tasks are to:

- Build a SOAP request based on the **xmlTemplate**, the **inputMap**, and the inputs supplied to the operation (see the next section).
- Perform any security functions as indicated by input values, such as signing the outbound request, encrypting it, and setting up SSL for https.
- Perform an XSLT transformation on the SOAP reply to populate the **documentStripped** and/or **jsonStripped** output, which strips the namespace prefixes from all of the output fields. For example the reply:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <xyz:Test>
      <xmlns:Name>Real Name</xmlns:Name>
      <abc:Address>An address</abc:Address>
    </xyz:Test>
  </soapenv:Body>
</soapenv:Envelope>
```

would become:

```
<Envelope>
  <Body>
    <Test>
      <Name>Real Name</Name>
      <Address>An address</Address>
    </Test>
  </Body>
</Envelope>
```

or in a JSON formatted string:

```
{"Body": {"Test": {"Name": "Real Name1", "Address": "An address1"}}
```

This conversion is necessary because the operation outputs use XPath filters or JSON to extract their values, and XPath expressions do not work well with XML that contains namespaces.

Building a SOAP Request

Building a SOAP request involves the following steps:

- Input resolution

This step uses the **inputMap** together with the operation inputs, to determine what values should be substituted.

For example, if the **inputMap** contains **Test | Name=Name** and there is an input called **Name** with the value **George Washington**, this step combines them to determine that the element in the request corresponding to **Test | Name=Name** should have the value **George Washington**. This step also handles wildcards in array references (new in OO 9.0). For example, an **inputMap** containing **Test | * | Name=Name*** and inputs **Name0** and **Name1** should have their values correspond to the elements in the SOAP request corresponding to **Test | 0 | Name** and **Test | 1 | Name**.

- Filling values

This step parses through the SOAP template looking for tokens. Each time it finds one, it attempts to find a value resolved from the previous step, and substitutes it if found. If no input is found with the specified name, the token is removed.

If the processing encounters the beginning of an array (indicated by the special comments in the template (“x or more repetitions” or “m to n repetitions”), then the resolved inputs for that array are sorted numerically (so that 10 appears after 9 rather than between 1 and 2), and then substituted into the SOAP request. Note that any missing gaps in the input names are ignored; for example, if the inputs are **Name0** and **Name2** (and **Name1** is missing), then only two values are substituted in the template (the values for **Name0** and **Name2**): no empty entries are created for missing values.

Complete Set of Inputs

Input	Description
contentType	Sets the http Content-Type header to the given value. Defaults to text/xml .
ICONCLUDE_WSW_VERSION	Must be the constant 2 .
header_*	Any input that begins with header_ is processed by the Http Client Post Raw operation, which then creates an HTTP header out of it. For example, if the input named header_Accept-Encoding contains the value gzip , the request will be altered to add the HTTP header Accept-Encoding: gzip .
inputMap	Described in Locating Inputs and Creating the inputMap .
password	The password sent to the Web service.
proxy	The name of the proxy host, if necessary, that is used to make the Web service request across a firewall.
proxyUsername	The proxy username, if necessary, used when making Web service requests

	across a firewall.
proxyPassword	The proxy password, if necessary, used when making Web service requests across a firewall.
proxyPort	The port on the proxy host, if necessary, that is used to make the Web service request across a firewall.
returnXMLRequest	If this input is set to true , a new output named rawXMLRequest is returned by the operation which contains the text of the SOAP request that was sent. This can be very valuable in troubleshooting.
timeout	The timeout in ms for the HTTP connection. Note that other timeouts may also come into play, such as the timeout between Central and the RAS.
trimComments	Removes all comments from the outbound SOAP request. <i>(Hidden input)</i>
trimNullOptionalTypes	If this input is set to true , for every element in xmlTemplate that is marked as Optional and for which no token has been substituted with a value, the element is removed from the outbound SOAP request. <i>(Hidden input)</i>
trimNullComplexTypes	If this input is set to true , for every element in xmlTemplate that has sub-elements (including arrays) and for which no token has been substituted with a value, the entire element (and all of its embedded elements) is removed from the outbound SOAP request. <i>(Hidden input)</i>
trustAllRoots	If this input is set to true , when HTTPS connections are made, it ignores the signing authority of the certificate (effectively permitting self-signed certificates) and ignores discrepancies between the hostname on the certificate and the actual server name that is hosting the Web service.
url	The URL of the Web service, which is extracted from the WSDL. This normally has variable references to the host and port so that this value need not be changed in order to send a request to a host or port that is different from the

	one hosting the WSDL.
useCookies	Determines whether the HTTP client will use cookies (that is, store them during the connection and send them back for subsequent HTTP requests to the same server).
usesJSON	Use JSON arrays for all the inputs of array type.
username	The username sent to the Web service.
xmlTemplate	Described in <i>How the Web Services Wizard Uses soapUI</i> .
WSSecurityEncryptRequest	A Boolean (defaults to false) indicating whether to encrypt the SOAP request.
WSSecurityKeystore	When encrypting or digitally signing the SOAP request, this indicates the keystore containing the certificate.
WSSecurityKeystorePassword	When encrypting or digitally signing the SOAP request, this indicates the password to the keystore.
WSSecurityKeystoreType	When encrypting or digitally signing the SOAP request, this indicates the keystore type.
WSSecuritySignRequest	A Boolean (defaults to false) indicating whether to digitally sign the SOAP request with an X509 signature.
WSSecurityTimestampRequest	A Boolean (defaults to false) indicating whether to securely timestamp the SOAP request.
wswAuthenticationType	Expects one of the following values: http , ws-security text , ws-security digest , and none . http is used for normal HTTP authentication, where the user and password are sent as HTTP headers. The two ws-security* options use SOAP WS-Security protocols.
*	All other headers are passed intact to the Http Client Post Raw operation, which may interpret them.

4 Using the Web Services Wizard Installer

This section includes the following topics:

- [About the Web Services Wizard Installer](#)
- [System Requirements](#)
- [Running the Web Services Wizard](#)
- [Web Services Wizard Dependencies](#)
- [Configuring Logging Settings](#)
- [Uninstalling the Web Services Wizard](#)

About the Web Services Wizard Installer

The Web Services Wizard Installer installs the Web Services Wizard (**wswizard.exe**), which allows you to create OO flows based on the operations available in the WSDL that you specify when you run the wizard.

System Requirements

Following are the minimum software requirements for systems running Web Services Wizard for HP Operations Orchestration:

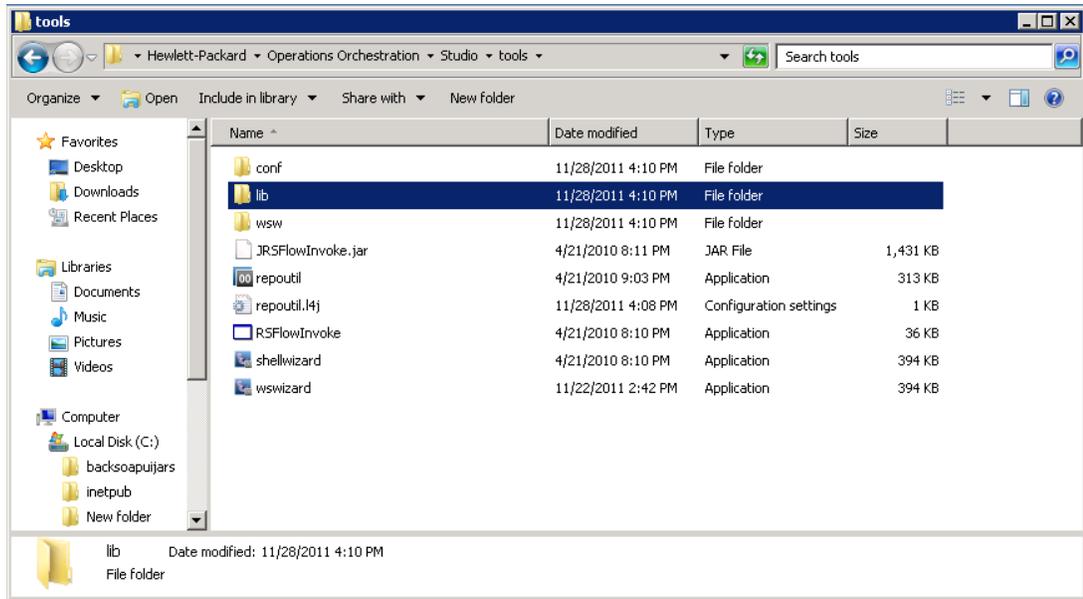
- HP Operations Orchestration Studio 9.00 (x32 or x64).
- HP Operations Orchestration Content Pack 7 or newer
- You must not have any version of the Web Services Wizard installed on your computer. Make sure the wswizard.exe that is installed with Studio and is located under Studio/tools is not running.

Running the Web Services Wizard Installer

The Web Services Wizard Installer is an executable file that can be downloaded from the HP Live Network page. Follow the instructions from [Downloading OO Releases and Documents on HP Live Network](#) to locate the installer under HP Operations Orchestration Content Pack 7. Follow the instructions from the installation wizard. After the Web Services Wizard is installed, a new folder is created in the OO Home folder in **Studio/tools/**. The new items are:

- wswizard.exe (the Web Services Wizard for HP Operations Orchestration application executable)
- wsw/lib/ folder (contains the wswizard.exe dependencies)
- under conf/ folder: wsw.properties and wsw.log4j.properties (contain logging configuration and user interface messages files)

A new folder is also created under OO Home folder: wsw/uninst, which contains the uninstall application executable and other files.



The Web Services Wizard is available from the Start Menu folder if you navigate to Hewlett-Packard/Operations Orchestration/Wizards where you can find shortcuts for running the application or uninstalling it.

Web Services Wizard Code Dependencies

When you run the Web Services Wizard, it starts a new javaw process and searches for library dependencies in the order defined in the classpath:

```
wsw\lib\*;
```

```
lib\*;
```

```
..\..\Studio\tools\lib\*;
```

```
..\..\Studio\tools\thirdparty\*;
```

```
..\..\Studio\tools\conf (The Wizard SDK searches this conf folder for certain files.)
```

Configure Logging Settings

Once the installation succeeds, you will find the new files wsw.properties and wsw.log4j.properties in the OO home directory, in the Studio/tools/conf/ folder. These files allow you to configure basic logging settings for the Web Services Wizard. When you run the Web Services Wizard, logging information is written to the file WebServiceWizard.txt, which can be found in the OO Home folder in Studio/tools/.

Uninstalling the Web Services Wizard

Before uninstalling the Web Services Wizard, make sure you back up your installation and repository. For information on backing up HP Central and Studio, see the *OO Administrator's Guide*.

Note: Uninstalling the Web Services Wizard deletes all of the resources, files, and folders created when the Web Services Wizard was installed. However, uninstalling does not delete the Studio/tools/ subfolder in the OO home directory or the log file if one was created there, or any repository that was created by the wizard in this folder (this happens when you run the wizard without providing an absolute repository path).

It is recommended that the Web Services Wizard be uninstalled before Studio. If Studio is uninstalled first, the wizard can be uninstalled afterwards, since the unistall executable is located outside Studio folder, under OO home directory in /wsw/uninst.

To uninstall Web Services Wizard

- 1 Make sure that the Web Services Wizard is shut down.
- 2 Open the Control Panel, and then click **Add/Remove Programs**.
- 3 Scroll down to and highlight **HP Operations Orchestration WS Wizard <version_number>**, and then click **Remove**.
- 4 When you are prompted to confirm whether you want to remove the Web Services Wizard and its components, click **Yes**.

The **Uninstall Status** box appears, in which progress of the removal is tracked on a progress bar. When the Web Services Wizard is completely removed, a message box informs you.

If you have any relevant folders open, the message box may tell you that some components could not be removed. This is not significant.

- 5 Click **OK**.

5 Using the Web Services Wizard to Create Web Services Flows

This section includes the following topics:

- [Using the Web Services Wizard to Create OO Flows from Selected WSDL Operations](#)
- [After Running the Web Services Wizard](#)

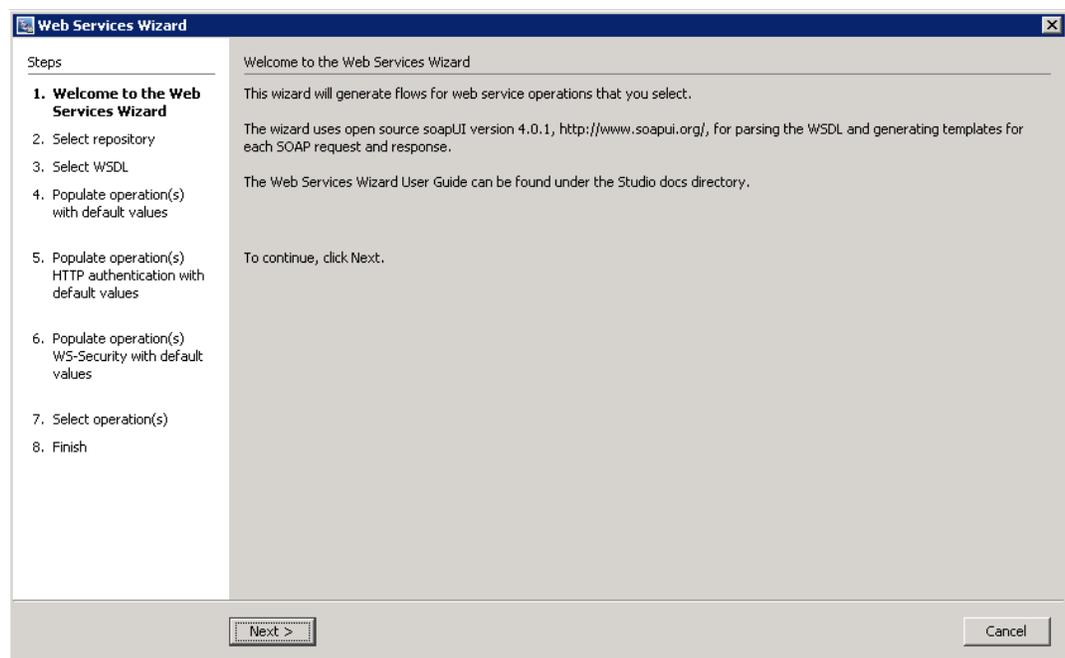
Using the Web Services Wizard to Create OO Flows from Selected WSDL Operations

The Web Services Wizard creates OO flows based on the operations available in the WSDL that you specify when you run the wizard. This tool is available in the OO home directory, under the Studio/tools/ folder or by clicking the WS Wizard for HP Operations Orchestration icon on the desktop. The Web Services Wizard is a window that leads the user through the tasks one step at a time and simplifies the process of flow creation. It is a simple and intuitive dialog, that does not include minimize/maximize buttons, cannot be placed in the taskbar and does not offer complex dialog options.

To use the Web Services Wizard to create an OO flow from a WSDL

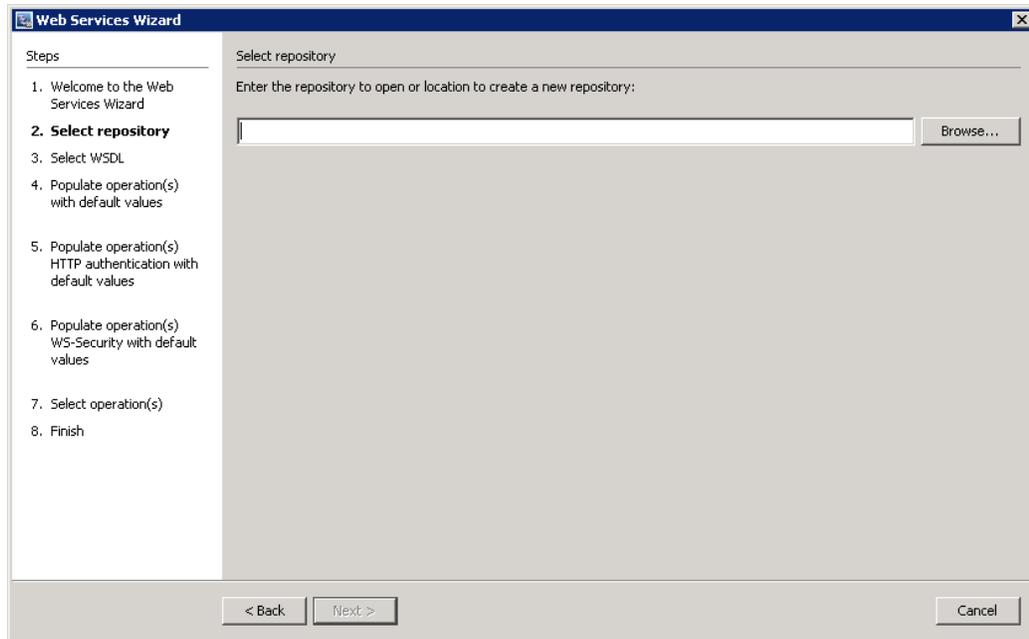
- 1 Start the Web Services Wizard.

It displays a **Welcome** page.

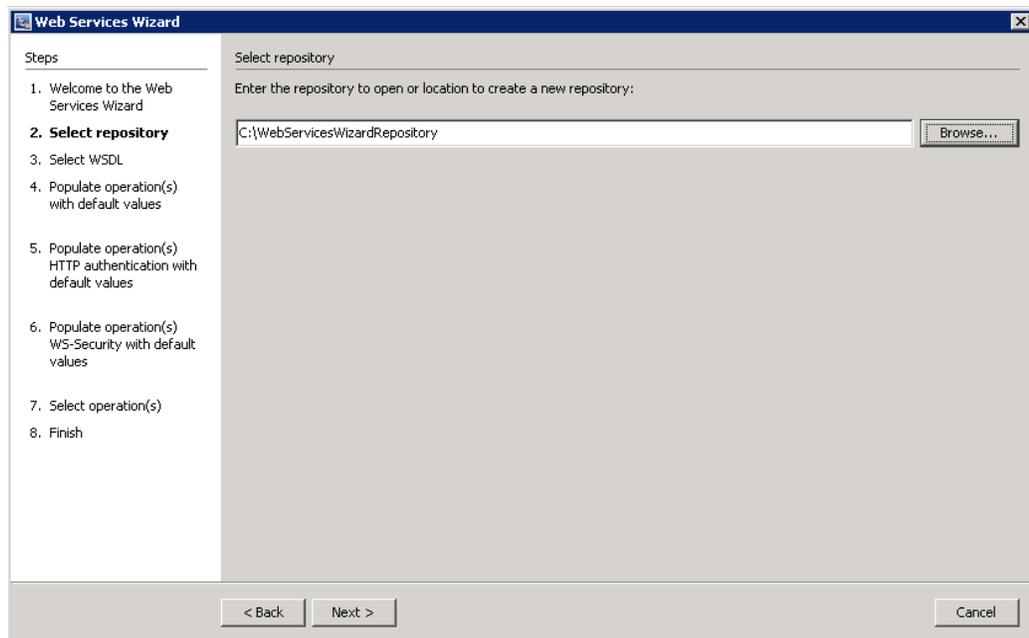


- 2 Click **Next** to continue.

The **Select Repository** page opens.

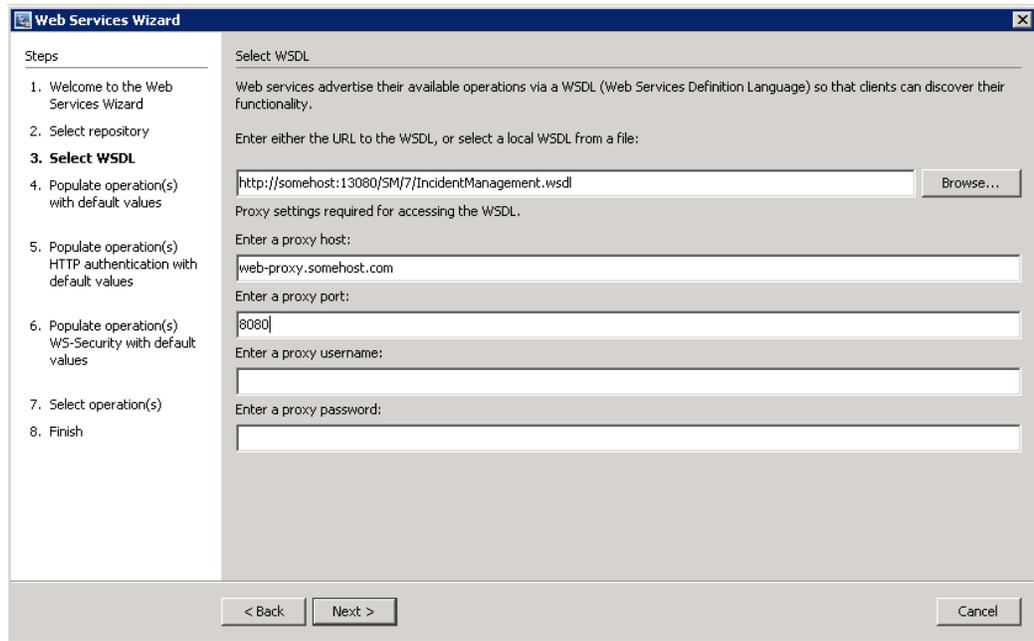


- 3 Enter or select a repository for the flows you want to create, and then click **Next**.
The Web Services Wizard validates the repository.



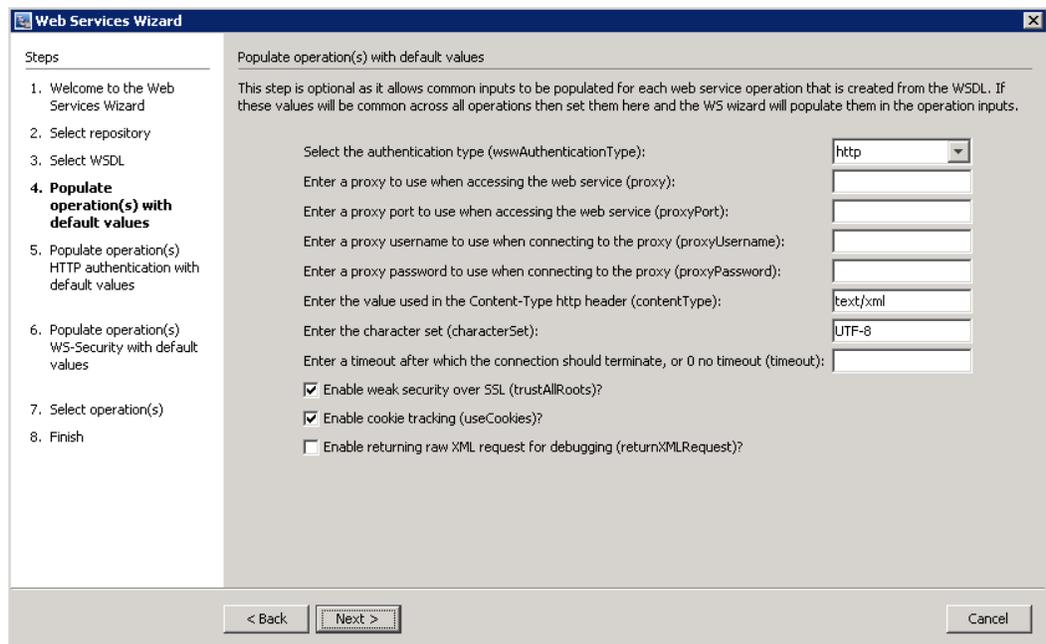
Note: The repository should not be locked (that is, it should not be opened in OO Studio).
If the specified repository is valid, the **Select WSDL** page opens.

- 4 Enter the URL to the WSDL, or select a local WSDL from a file system.



If proxy information is required to access the WSDL URL, enter it here. If loading the WSDL succeeds, the **Populate operation(s) with default values** page opens. That allows you to set default values for the flows that the Web Services Wizard generates.

- 5 **Optional Step:** Enter values for any common inputs (these are the default inputs of Invoke Method 2 operation so every flow created by the wizard will contain them). If the inputs will be common for all flows created, they can be entered on this page. The default values have been populated on the page.

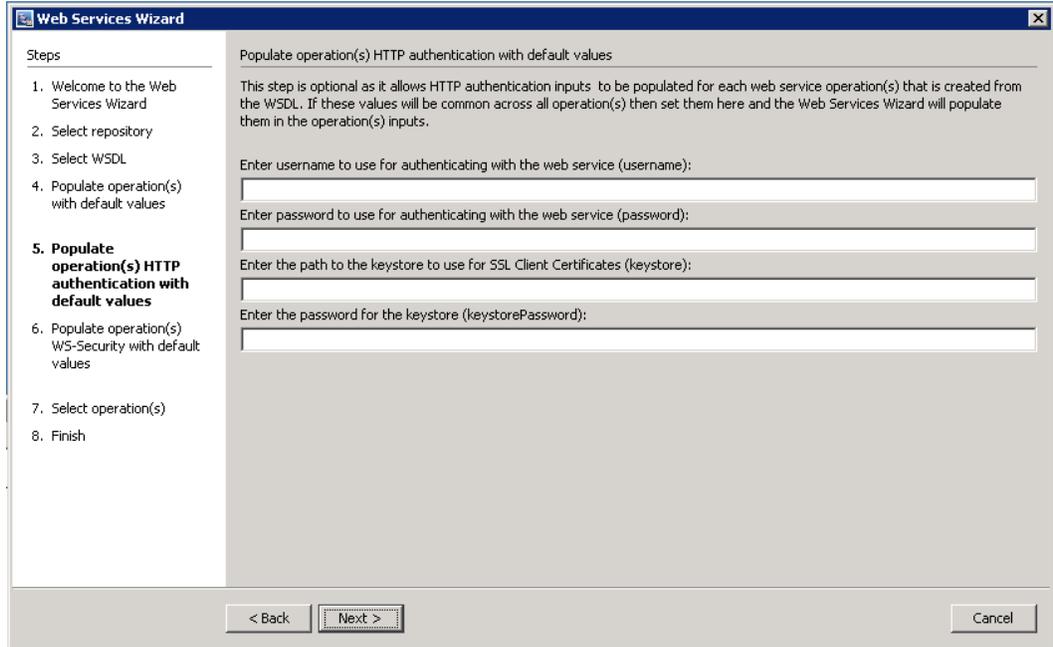


Note: If you set the values here, each operation will have the values preset and the only way to change them is to modify each flow in Studio or rerun the Web Services Wizard and generate the flows in a different repository.

Click **Next** to continue to either the **Populate operation(s) HTTP authentication with default values page** or **Populate operation(s) WS-Security with default values page** or **Select operation(s) page** depending on the authentication type

selected. For example, if you select an authentication type of “ws-security text”, the next page will be the optional step of populating the WS-Security default input values.

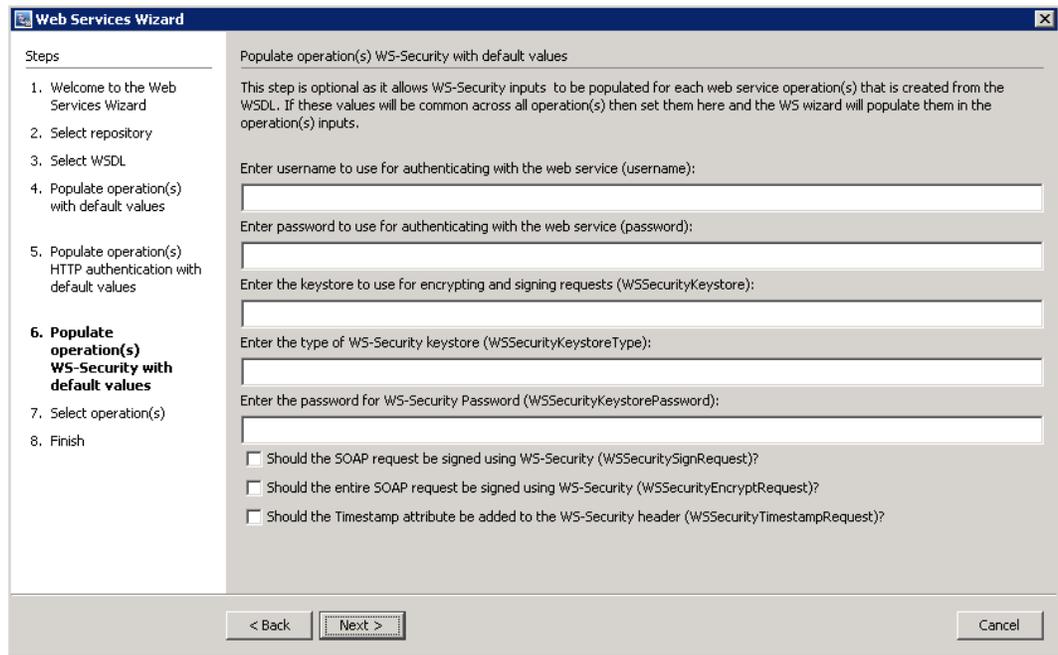
- 6 **Optional Step:** Enter values for the common HTTP authentication inputs. If the inputs will be common for all flows created, they can be entered on this page.



Note: If you set the values here, each operation will have the values preset and the only way to change them is to modify each flow in Studio or rerun the Web Services Wizard and generate the flows in a different repository.

Click **Next** to continue.

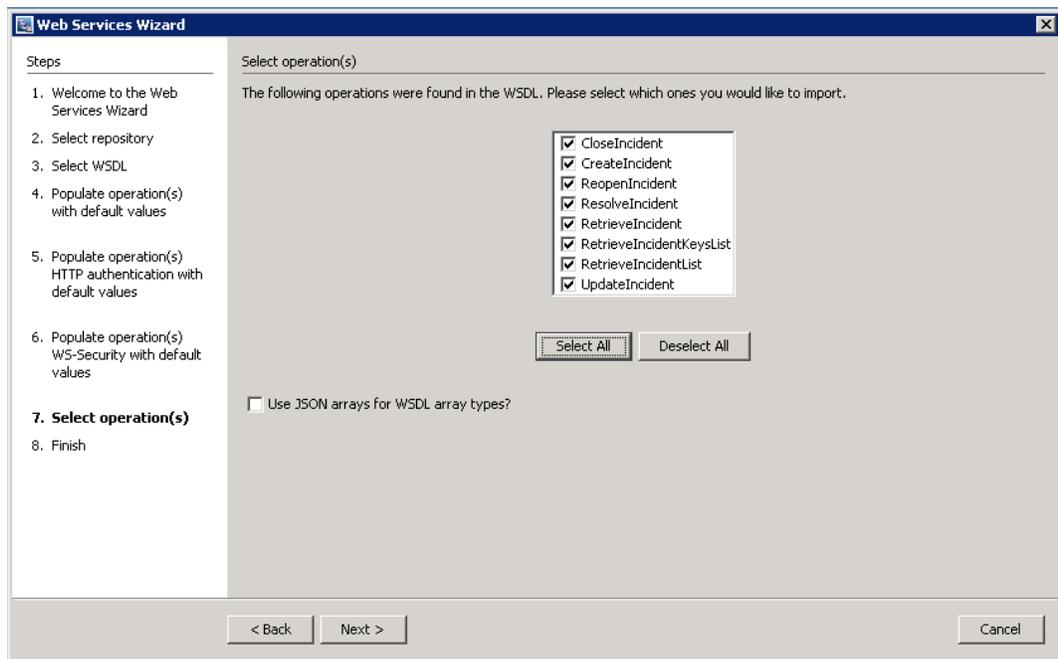
- 7 **Optional Step:** Enter values for the common WS-Security inputs. If the inputs will be common for all flows created, they can be entered on this page.



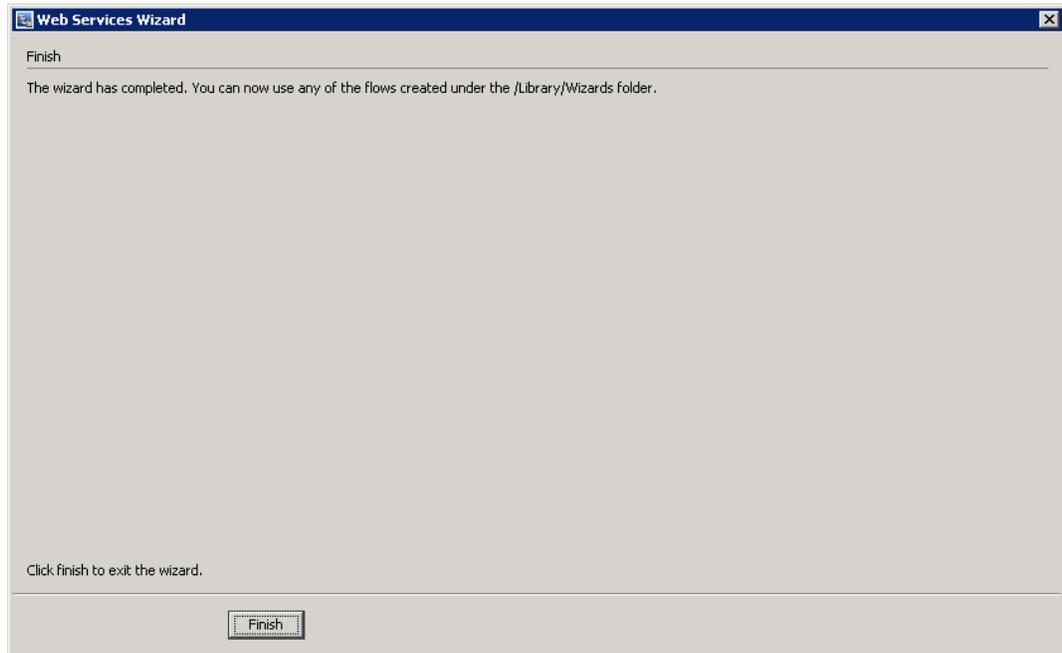
Note: If you set the values here, each operation will have the values preset and the only way to change them is to modify each flow in Studio or rerun the Web Services Wizard and generate the flows in a different repository.

Click the **Next** button to continue.

- 8 Select the operation(s) for which you are interested in creating flows. The available operations are displayed in the list. If you want to use JSON formatted arrays for all array type inputs in all the generated flows, check the **Use JSON arrays for WSDL array** types box. If you do not check the box, you can still use JSON formatted arrays, but you will have to manually set the input usesJSON in Invoke Method 2 to true for all the flows that you want to accept JSON data. Click the **Next** button to continue.



After the flows are successfully created and saved in the repository, the Web Services Wizard finishes.



After Running the Web Services Wizard

If the Web Services Wizard ran successfully, you will have a new set of flows that are ready for you to use. However, there are a number of reasons in why you may have to make some adjustments before the operations are usable:

- The source WSDL may have problems or may have changed.
- There may be undocumented headers.

Read this section, together with the Troubleshooting section, to diagnose and correct these situations.

Notes:

In addition to inputs, the parsing obtains results that can be captured as operation outputs (which are expressed as results in steps). Any arrays in the XML are extracted only as a single XML result from which the flow author can extract narrow subsets.

- In the flows that the Web Services Wizard generates, the flow inputs that correspond to Web services inputs are optional. It is frequently the case that some of the inputs that Web service definitions indicate as required are not actually required, and mirroring those settings in the flow would force the flow user to enter unused values when running it. So, the Web Services Wizard sets all the inputs as optional. When the Web service does indicate that a field is optional, it precedes the field with the comment "`<!-- Optional:>`" or "`<!zero or more repetitions-->`". For information on which inputs should be required, see the documentation for the Web service for which you are creating operations.
- If the Web service, whose WSDL you are accessing, resides on the other side of a firewall from your Studio machine, you must specify an HTTP proxy with which to reach the Web service.

6 Troubleshooting

This section includes the following topics:

- [General Troubleshooting Principles](#)
- [Troubleshooting Steps](#)

General Troubleshooting Principles

If you experience difficulties running the Web Services Wizard, first confirm that any changes you make work for one input before trying them will all the inputs.

If you experience difficulties running the Web Services Wizard against a WSDL with a URL that starts with **https**, try opening the WSDL in a browser and saving it to the local file system. Make sure to copy all dependencies (such as xsd files) since accessing them through the wizard will be equally difficult. Such files can be found under `<xs:schema><xs:import>` tags. Then run the Web Services Wizard against the WSDL file instead.

If an unexpected error message is returned from running the OO flows that the Web Services Wizard created, try adding and setting the **trimNullOptionalTypes** and/or **trimNullComplexTypes** to **false** in the **Invoke Method 2** operation of your flow. This will result in the outbound SOAP request looking more like the request sent by soapUI when inputs have null values.

Troubleshooting Steps

If the Web Services Wizard fails to load the operations for selection and returns with a null pointer exception:

- Try removing any white space around the comments section of the WSDL.

This is a known issue with the soapUI utility that the Web Services Wizard uses.

- The Web Services Wizard passes on the soapUI's "Null-pointer Exception" message followed by the rather obvious "Failed to load WSDL" if you attempt to load an invalid WSDL.

This is a known issue with the soapUI utility that the Web Services Wizard uses.

- Validate that the XML request is what you expected.

This can be done by setting the **returnXMLRequest** input value to **true** in the **Invoke Method 2** operation in your newly created flow. This will add an output result of the actual XML request that was sent.

- Try the request in soapUI to verify that the Web service is working correctly.

Install soapUI (<http://www.soapui.org/>), create a project from the WSDL and a request object for the operation in question. Then replace its content with the XML request from the output above.

- Some WSDLs have been written in a way that causes the Web Services Wizard to fail to recognize some array types. When one of these OO flows runs, it might return the following exception:

```
<faultcode><soapenv:Server.userException</faultcode><faultstring>org.xml.sax.SAXException: Found character data inside an array element while deserializing</faultstring></pre>
```

The original WSDL file, which failed to be correctly processed by the Web Services Wizard, used the **ArrayOf_xsd_String** implementation:

```
<wsdl:message name="createSelectionListRequest">
...
<wsdl:part name="values" type="impl:ArrayOf_xsd_String"/>
```

```
...  
</wsdl:message>
```

The modified WSDL file, which is correctly processed by the Web Services Wizard, redefines the type **ArrayOf_xsd_String** to **WSListValues** (this is a particular case for the **createSelectionList** operation from the example). Using the **WSListValues** type definition you can also define your own array of string types (for example, **ArrayOfStrings**) instead of using **ArrayOf_xsd_String**.

```
<wsdl:types>  
    ...  
<complexType name="WSListValues">  
<sequence>  
<!--Zero or more repetitions!-->  
<element maxOccurs="unbounded" minOccurs="0" name="value" type="xsd:string"  
</sequence>  
</complexType>  
    ...  
</wsdl:types>  
  
<wsdl:message name="createSelectionListRequest">  
    ...  
<wsdl:part name="values" type="tns1:WSListValues"/>  
    ...  
</wsdl:message>
```