HP Operations Orchestration

for the Windows and Linux operating systems

Software Version: OO 10.x

Web Services Wizard Guide

Document Release Date: November 2013

Software Release Date: November 2013

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 10.00 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

This section includes the following topics:

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 is released and the integration content does not support the new version, you can use the Web Services Wizard to create new OO flows.

Example

You have an application named **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 , the application's APIs from the Web service's WSDL 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 for each method, 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 using 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 from the Base Content Pack. The flows are created in the project location folder specified by the user. 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.

Download OO Releases and Documents on HP Live Network

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

To download OO releases and documents, go to:

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 sign-in page appears.
3. Enter your HP Password User ID and Passwordand click Sign-in.
4. Click **HP Operations Orchestration 10.0**.
5. Search for Base Content Pack.

2 Wizard Processing Details

This section includes the following topics:

How the Web Services Wizard Uses SoapUI

SoapUI is an open-source Web service testing tool. It provides Web service inspection, invoking, development, and simulation. The Web Services Wizard uses SoapUI to parse the WSDL and create a template SOAP request (Base Content Pack 2013 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 named **Request 1** for every operation in the WSDL. This is the template that the Web Services Wizard receives from SoapUI, and uses to populate the **xmlTemplate** input.

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

For HP 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 under **/Studio/tools/conf/**.

You only need to enter the configuration information once (the first time you run the wizard against a WSDL outside the firewall). It is then read from the **wsw,properties** file and prepopulated in the wizard GUI. You can change it in the file or in the wizard GUI and the values are saved for the next time you run the wizard.

For OO 10.00, specifying a Web proxy in the **wsw.properties** file is no longer supported. You can change it only from the wizard GUI.

After retrieving the templates for the request and the response, the WSDL is discarded. No further information is obtained from the WSDL, and all subsequent operations in both the Web Services Wizard and the **Invoke Method 2** operation are 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 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.

Locate Inputs and Create 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, as an element (such as **Name**)may 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**. As arrays may be nested, there may be templates whose values contain more than one zero (0).

The next task is to define a meaningful set of input names to be created. This is done 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** contains:

```
Test|Name=Name
Test|Address=Address
```

The creation of the **inputMap** is a little complex. Use the following tips when determining a name for each path name:

- Use the last part 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, add a prefix for additional levels (with a period separator) onto the user-friendly name to make the name 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

as 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 (*). The position of the wildcard in the user-friendly name is moved to the end of the next element. In the above example, for **Test|0|Name** and **Test|0|Address**, the following input map would be created:

Test|*|Name=^Name*$

Test|*|Address=^Address*$

Note:

  - The purpose of moving 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 part 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** results in:

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

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

regardless ofthe 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), s/he needs 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** are created as inputs to the step and the flow.

The Web Services Wizard accepts JSON-formatted arrays for the array types 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|*|Affecte
dCI=^AffectedCI*$
```

- The associated Web Services Wizard created AffectedCI0 input field JSON array formatted value should be similar to:

```
["CIvalue1","CIvalue2","CIvalue3"]
```

Locate Outputs and Create 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 correctly 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. This causes the entire portion of the XML document to be returned in a single output, and it is the flow developer's responsibility to use other operations (like XML or JSON ones) to extract the relevant items. 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** is 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.

Populate InvokeMethod2 Default Values for All Operations

The Web Services Wizard allows setting **InvokeMethod2** inputs so that each operation created from the WSDL has the inputs set by default. For example, the **timeout** input can 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. Setting the default values in the Web Services Wizard is optional.

The Web Services Wizard does not validate the default inputs entered. This validation takes place during the flow run.. The Web Services Wizard allows you to specify default values only for the authentication type selected. For example, if the **http** authentication type is selected, the wizard allows you to enter the default inputs for HTTP authentication onlyand skips 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

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 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, that 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 as the operation outputs use XPath filters or JSON to extract values, and XPath expressions do not work well with XML that contains namespaces.

Build a SOAP Request

Building a SOAP request includes the following steps:

- Input resolution

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

For example, if the **inputMap** contains **Test|Name=Name** and there is an input named **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. For example, an **inputMap** containing **Test|*|Name=Name*** and inputs **Name0** and **Name1** should have values corresponding to the elements in the SOAP request corresponding to **Test|0|Name** and **Test|1|Name**.

- Completing values

This step parses through the SOAP template looking for tokens. When 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"), 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 that is used to make the Web service request across a firewall. (Optional)
proxyUsername	The proxy username, if necessary, used when making Web service requests across a firewall. (Optional)

proxyPassword	The proxy password used when making Web service requests across a firewall. (Optional)
proxyPort	The port on the proxy host used to make the Web service request across a firewall. (Optional)
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 is useful for troubleshooting purposes.
timeout	The timeout in ms for the HTTP connection. Note that there may be other timeouts that affect the connection, such as the timeout between Central and the RAS.
trimComments	Removes all comments from the outbound SOAP request. *(Hidden input)*
trimNullOptionalTypes	By default (**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. *(Hidden input)*
trimNullComplexTypes	By default (**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. *(Hidden input)*
trustAllRoots	When set to **true**, when HTTPS connections are made, it ignores the signing authority of the certificate (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, extracted from the WSDL. This generally has variable references to the host and port so that this value does not need to be changed to send a request to a host or a port different from the one hosting the WSDL.

useCookies	Determines whether the HTTP client will use cookies (store them during the connection and send them back for subsequent HTTP requests to the same server).
usesJSON	Use JSON arrays for all inputs of array type.
username	The username sent to the Web service.
xmlTemplate	Described in *How the Web Services Wizard Uses SoapUI*.
WSSecurityEncryptRequest	A boolean value (default **false**) indicating whether or not 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 value (default **false**) indicating whether or not to digitally sign the SOAP request with an X509 signature.
WSSecurityTimestampRequest	A boolean value (default **false**) indicating whether or not to securely timestamp the SOAP request.
wswAuthenticationType	Can be assigned with 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 can interpret them.

4 Using the Web Services Wizard

This section includes the following topics:

System Requirements

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

- For running the wizards, the environment must have Java SE Runtime Environment 7 (also known as JRE) installed.

Install the Web Services Wizard

The Web Services wizard is automatically installed if HP OO Studio is chosen from the Operations Orchestration installer

Web Services Wizard Code Dependencies

The wizard does not have any dependencies. All third parties are encapsulated into the executable files.

Configure Logging Settings

The configure logging settings are no longer supported in the10x wizard.

WS Wizard Enhancements from 9x

- The wizard now appears in the taskbar and can be closed, minimized or brought to the front.
- The UI is correctly divided. The scroll bar is now not necessary.
- Operation selection has a search functionality which allows you to quickly find an operation by typing letters of the operation name.
- The wizard includes functionality to override an existing flow/s.

5 Using the Web Services Wizard to Create Web Services Flows

This section includes the following topics:

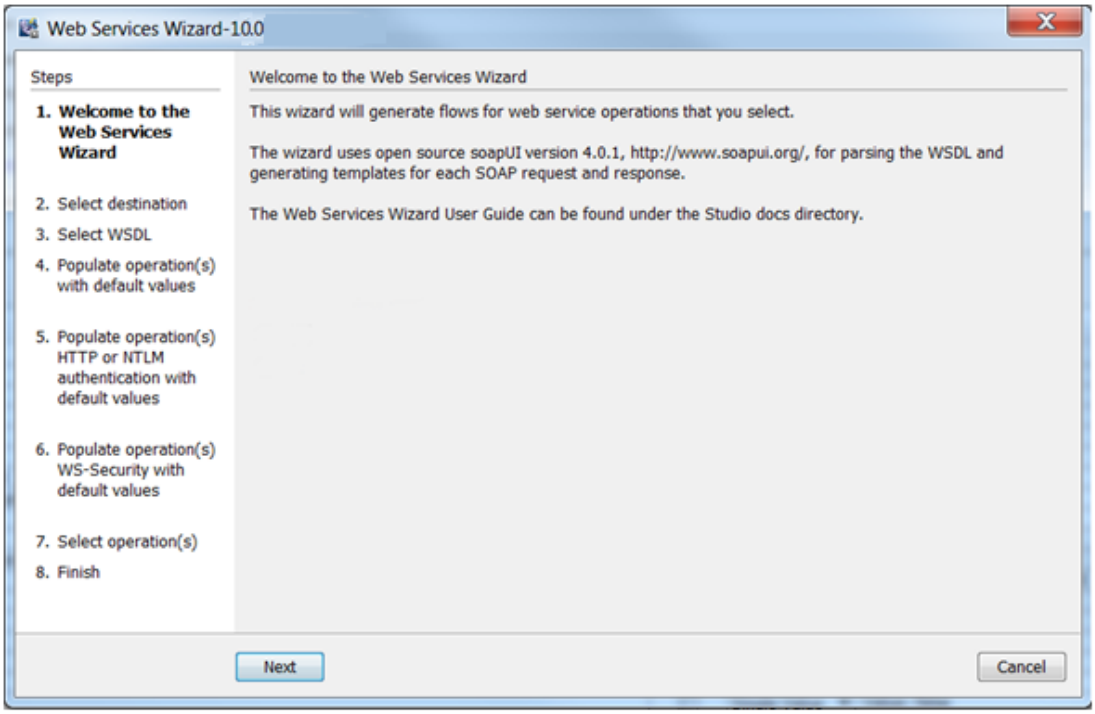
Use the Web Services Wizard to Create OO Flows from Selected WSDL Operations

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 by launching the wizard executable file. The Web Services Wizard is a is a simple and intuitive tool that leads the user through the tasks and simplifies the process of flow creation.

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

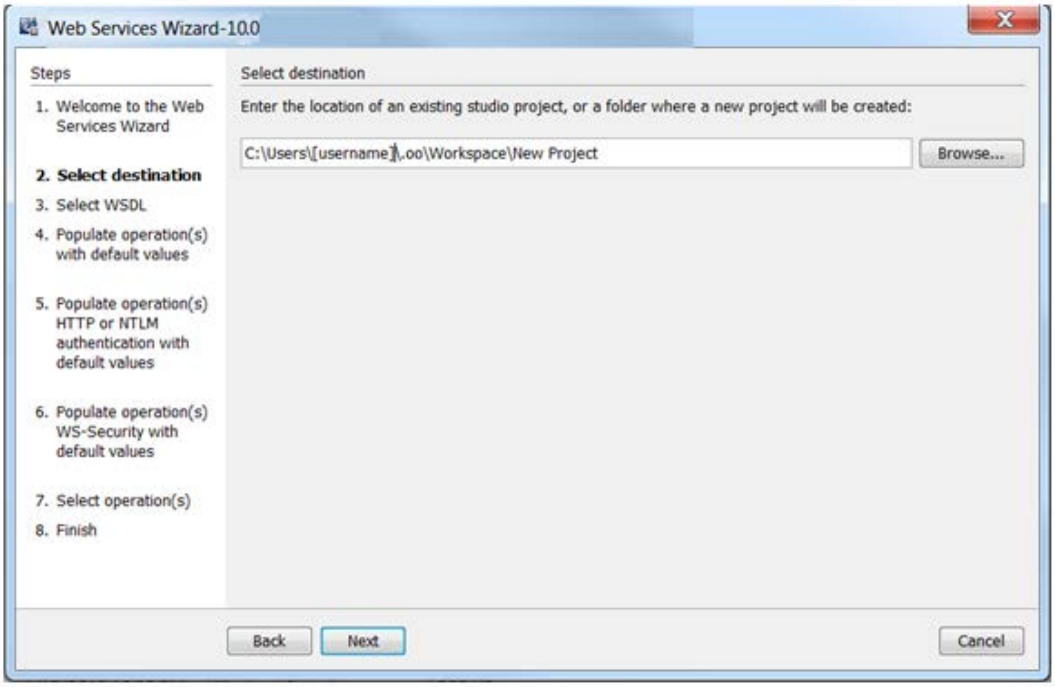
1 Start the Web Services Wizard.

The **Welcome** page opens



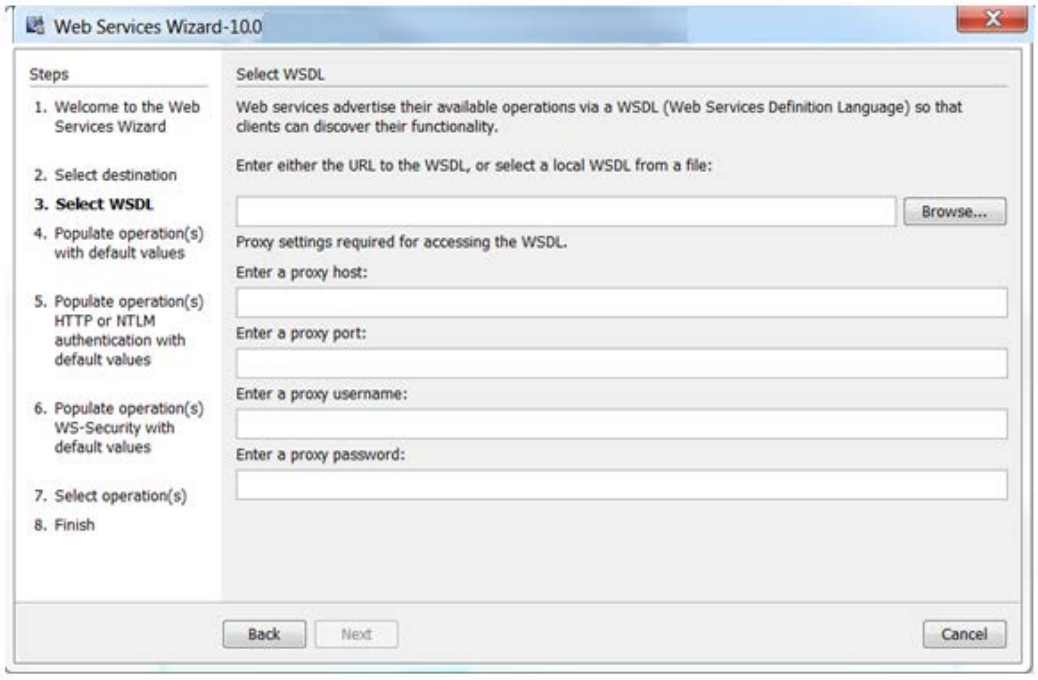
2 Click **Next** to continue.

The **Select Destination** page opens.



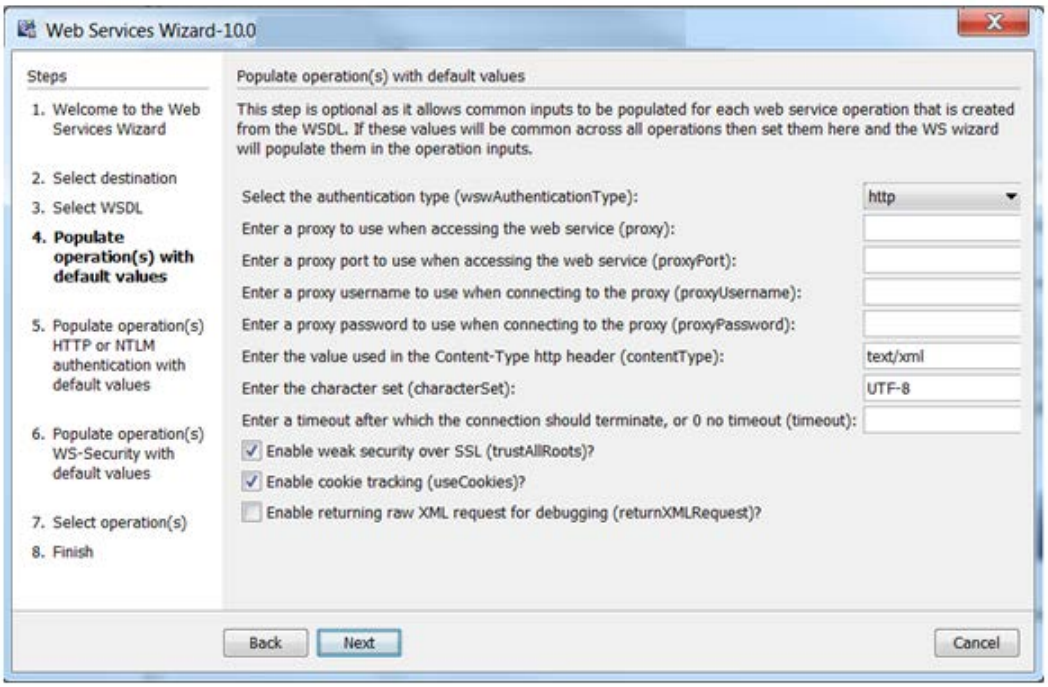
- In the **Enter the location** field, enter the required project path or click **Browse** to locate the project location, and then click **Next.**
- The wizard generates a 10.x studio project but not a content pack or a repository. The project has a default location: C:\Users\[username]\.oo\Workspace\New Project.

3 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. In this page, you can set default values for the flows that the Web Services Wizard generates.

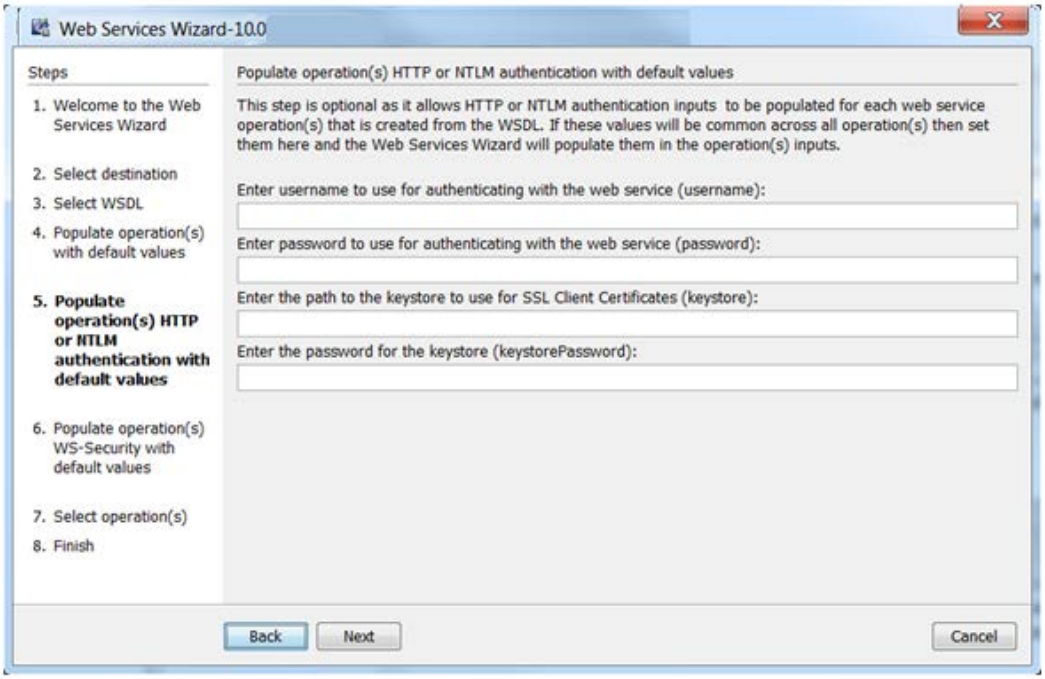
4 **Optional Step:** Enter values for any common inputs (these are the default inputs of Invoke Method 2 operation, so every flow created by the wizard will contain them). If the inputs are common for all flows created, they can be entered on this page. The default values are populated on the page.



Note: If you set the values here, each operation will be assigned with the preset values . To change the value, you need to modify each flow in Studio or rerun the Web Services Wizard select the **Overwrite the flow if already exists checkbox** in step 7.

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.

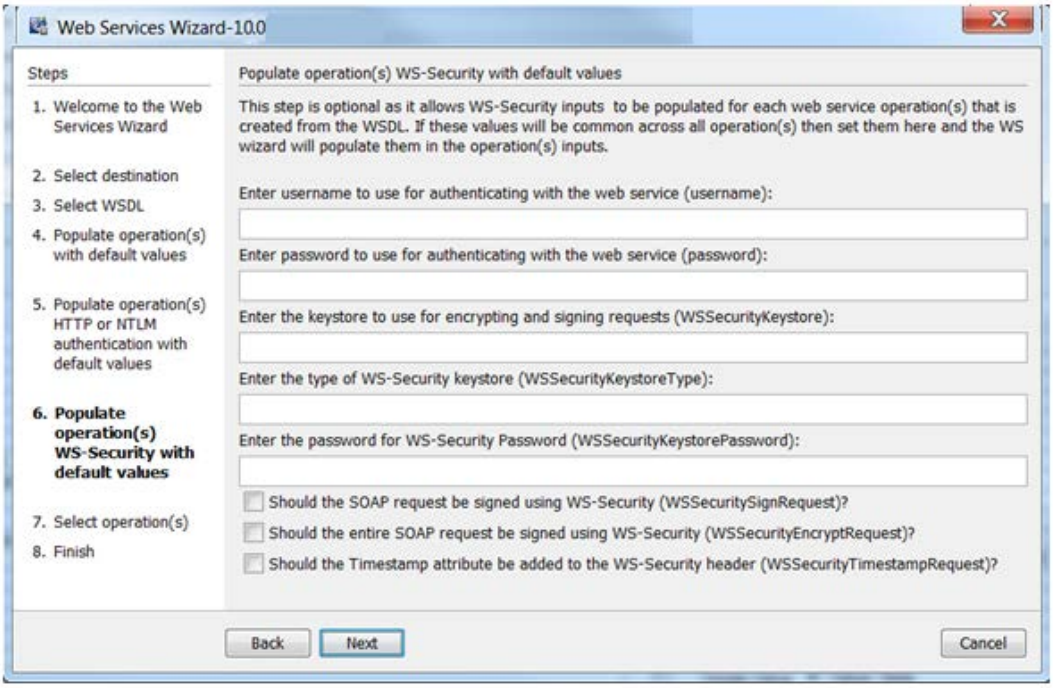
5 **Optional Step:** Enter values for the common HTTP authentication inputs. If the inputs are common for all flows created, they can be entered on this page.



Note: If you set the values here, each operation will be assigned with the preset values . To change the value, you need to modify each flow in Studio or rerun the Web Services Wizard select the **Overwrite the flow if already exists checkbox** in step 7.

Click **Next** to continue.

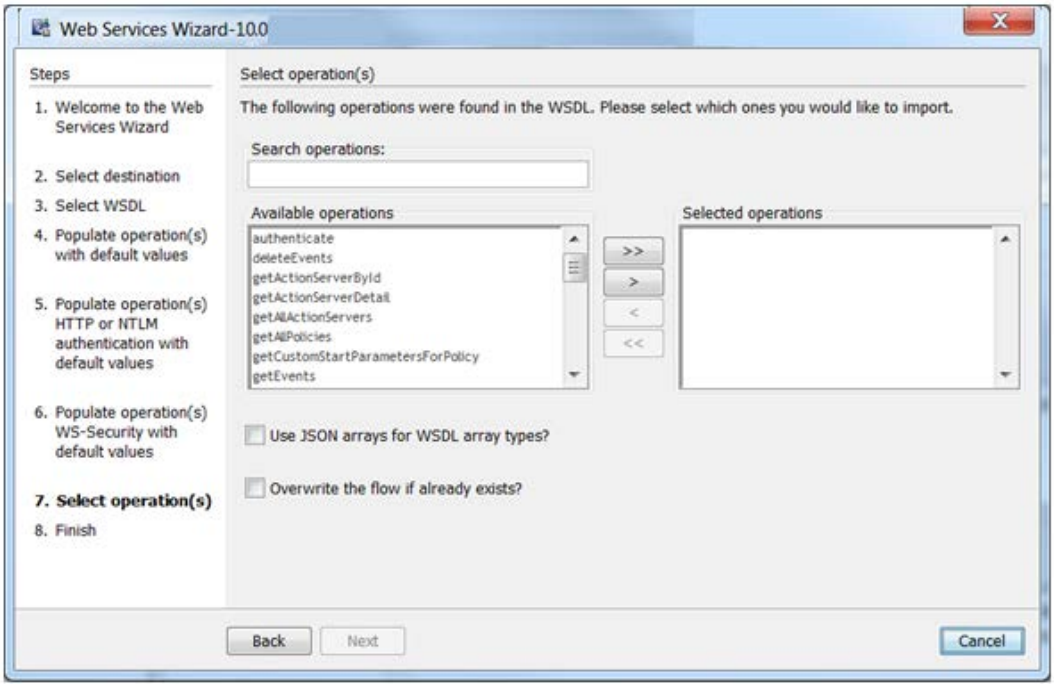
6 **Optional Step:** Enter values for the common WS-Security inputs. If the inputs are common for all flows created, they can be entered on this page.



Note: If you set the values here, each operation will be assigned with the preset values . To change the value, you need to modify each flow in Studio or rerun the Web Services Wizard and select the **Overwrite the flow if already exists checkbox** in step 7.

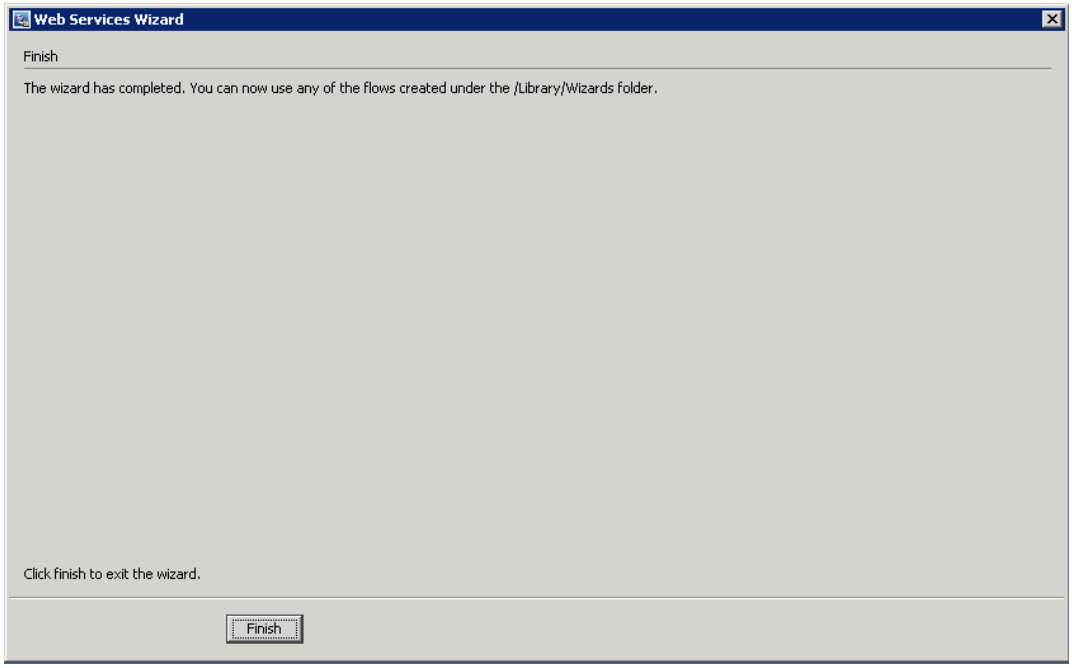
Click **Next** to continue.

7 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 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 **Next** to continue.



You can move the operations from one column to the other. Use the search textbox if the list is long, and you cannot find the required cmdlet. The wizard searches the list for the operations with names containing the search text. In addition, the wizard updates the list as you type.

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


Web Services Wizard
Finish
The wizard has completed. You can now use any of the flows created under the /Library/Wizards folder.
Click finish to exit the wizard.
Finish

After Running the Web Services Wizard

If the Web Services Wizard ran successfully, you will have a new set of flows ready for use. However, may have to make some adjustments before the operations can be used, due to the following issues:

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

To diagnose and correct these situations, read this section, along with the Troubleshooting section.

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 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. Sometimes some of the inputs that the Web service definitions indicate as required are not actually required, and mirroring these settings in the flow would force the flow user to enter unused values when running it. So, the Web Services Wizard sets all 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 relevant Web service.
- 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 to be used to reach the Web service.

6 Troubleshooting

This section includes the following topics:

General Troubleshooting Principles

- If you experience difficulties running the Web Services Wizard, first check any changes you make on one input before trying them on all 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) as it is difficult to access them through the wizard. These 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 after 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 results 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 message "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 as 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 may return the following exception:

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

  The original WSDL file, that was 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 specific 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**) in place of **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>
```

- In some cases, the InvokeMethod2 operation fails to run when used in a flow in Studio. This occurs because the SOAP envelope isincorrect. A single flow UUID is passed within `<flowUuids></flowUuids>` instead of an array. This occurs for all soap requests that have an element `wsdl:arrayType` that does not have `nillable="true"`.

  In the SOAP UI, the generated request contains an empty `flowUuids` array by default. You should edit it manually and insert any relevant UUIDs of interest. For example, the xmlTemplate input of the Invoke Method 2 step in the generated Get Flow Details can be modified manually as shown below:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsc="http://wscentralservice.services.dharma.iconclude.com"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <soapenv:Header/>
   <soapenv:Body>
      <wsc:getFlowDetails
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <flowUuids xsi:type="xsd:string">?</flowUuids>
    </wsc:getFlowDetails>
   </soapenv:Body>
</soapenv:Envelope>
```

OO WebService Tool with Proxy

If you cannot launch the OO WebService wizard to create ops from a wsdl file (local or remote), create an xml file named soapui-settings.xml in the following format and save it in your home directory:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <con:soapui-settings xmlns:con="http://eviware.com/soapui/config">
  <con:setting id="WsdlSettings@cache-wsdls">true</con:setting>
  <con:setting id="WsdlSettings@pretty-print-response-xml">true</con:setting>
  <con:setting id="HttpSettings@include_request_in_time_taken">true</con:setting>
  <con:setting id="HttpSettings@include_response_in_time_taken">true</con:setting>
  <con:setting id="WsdlSettings@name-with-binding">true</con:setting>
  <con:setting id="HttpSettings@max_connections_per_host">500</con:setting>
  <con:setting id="HttpSettings@max_total_connections">2000</con:setting>
  <con:setting id="ProxySettings@host">111.222.111.322</con:setting>
  <con:setting id="ProxySettings@port">3128</con:setting>
  <con:setting id="ProxySettings@enableProxy">true</con:setting>
 </con:soapui-settings>
```

If this fails due to proxy issues, this file will ask for your credentials for the proxy (hard-coded in the file). Modify the proxy host and port in the file, place it in your home directory, and enter the username and password, when prompted. This procedure will allow the wsdl file to be loaded correctly.