Web Service Tutorial

Providing a web service

Revision date 14 August 2009

Table of Contents

Overview

This tutorial will show you how to develop and deploy a simple SOAP based web service.

The tutorial will go through the following tasks:

1. Define the structure of the request and the response documents in Plex.
2. Create a function that can receive the http request from the caller, extract the request document from the http request, call the function that will handle the request and return the response to the caller.
3. Create a function that can handle the received request document and create the response document.
4. Deploy the generated service.
5. Create a WSDL file, which makes it possible for external systems to call the web service.

Preparations

Creating a local model

Before you are able to run this tutorial, you have to create a local Plex model that has the Websydian web service pattern library (WSYSOAP) as a library model.

If you want to use an existing model, you can of course do so. Just make sure to check the configuration of the model (see below).

> Create a Plex group model that has WSYSOAP as a library model.
>
> Create a Plex local model based on the group model.
>
> Specify the following configuration:

Model	Variant	Version	Level
ACTIVE	Base	*Latest Version*	*Latest Level*
DATE	Windows Client	*Latest Version*	*Latest Level*
FIELDS	Base	*Latest Version*	*Latest Level*
FUNDATI	Base	*Latest Version*	*Latest Level*
JAVAAPI	Base	*Latest Version*	*Latest Level*
OBJECTS	Base	*Latest Version*	*Latest Level*
SDSTRING	Windows	v.6.1	v.6.1
STORAGE	Base	*Latest Version*	*Latest Level*
UIBASIC	Base	*Latest Version*	*Latest Level*
UISTYLE	Base	*Latest Version*	*Latest Level*

Model	Variant	Version	Level
VALIDATE	Base	*Latest Version*	*Latest Level*
WINAPI	Base	*Latest Version*	*Latest Level*
WSYBASE	DWA – Windows	v.6.1	v.6.1
WSYDOM	Windows	v.6.1	v.6.1
WSYHTTP	Windows	v.6.1	v.6.1
WSYSOAP	Base	v.6.1	v.6.1
WSYXML	Base	v.6.1	v.6.1

Creating document definitions

When you provide a web service, you will normally be in charge of determining the structure of the request and response document.

This means that you can't expect to have a schema or a WSDL file that can be imported to define the request and response documents in Plex.

Because of this, you have to create the definitions of the documents manually in Plex.

Create the following triples to define the request and response documents:

Source object	Verb	Target object
Request	is a ENT	WSYXML/ XMLElementWithServiceFunctions
	is a ENT	WSYXML/NamespaceAware
Request.Fields	field FLD	firstName
	field FLD	lastName
Request.Fields. firstName	is a FLD	WSYXML/ElementField
Request.Fields. lastName	is a FLD	WSYXML/ElementField
Request	has FLD	Request.Fields.lastName
	has FLD	Request.Fields.firstName
Response	is a ENT	WSYXML/ XMLElementWithServiceFunctions
	is a ENT	WSYXML/NamespaceAware

Source object	Verb	Target object
Response.Fields	field FLD	greeting
Response.Fields.greeting	is a FLD	WSYXML/ElementField
Response	has FLD	Response.Fields.greeting

The Request and Response documents both scope two source code objects: Namespace and Prefix.

The Namespace source code determines namespace of the top element of the documents and of the scoped element fields.

The Prefix source code determines the prefix that is used as an alias for the namespace. Specify the following literal values:

For Request.Namespace and Response.Namespace:

http://www.websydian.com/services/greet

For Request.Prefix and Response.Prefix:

p1

These definitions enables you to create and read the request and response documents.

The definitions for the Request document define a document like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<p1:Request xmlns:p1="http://www.websydian.com/services/greet">
  <p1:firstName>String</p1:firstName>
  <p1:lastName>String</p1:lastName>
</p1:Request>
```

The definitions for the Response document define a document like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<p1:Response xmlns:p1="http://www.websydian.com/services/greet">
  <p1:greeting>String</p1:greeting>
</p1:Response>
```

It is highly recommended that you always specify a namespace for the top element of the request and the response document. The SOAP standard does not demand this, but many tools can't call services where the documents have no namespace.

Before you can use the functions scoped by the two entities, you need to make all the scoped objects real.

Make all objects scoped by the Request and Response entities real.

Create a web service entity

To be able to provide a SOAP-based web service, you need a number of service functions and definitions for the SOAP-structure that is wrapped around the request and response documents.

You create these by creating an entity that inherits from the abstract HttpSoap entity found in the WSYSOAP model.

This creates all the definitions that are necessary for creating both the subscriber and the provider role. As this tutorial only handles the provider part, the functions used to call a web service are set to implement no.

Create the triples:

Source object	Verb	Target object
Publisher	is a ENT	WSYSOAP/HttpSoap
Publisher.Services.SoapGenerator	implement SYS	No
Publisher.Services.CallSoapGenerator	implement SYS	No

Create XMLHandler

The function that will handle the request document and write the response document must inherit from an abstract XMLHandler.

Create the triples:

Source object	Verb	Target object
Publisher.Services	includes FNC	XMLHandlers
Publisher.Services.XMLHandlers	includes FNC	GreetService
Publisher.Services.XMLHandlers.GreetService	is a FNC	Publisher.Abstract.XMLHandler
	is a FNC	WSYDOM/DomServerExternal
	implement SYS	Yes
	local FLD	Response.Fields.greeting

Make the publisher entity and all its scoped objects real.

The function servicing the http requests (the SoapProcessor) can call a number of different XMLHandlers, based on the http header SOAPAction that is sent as part of the http request.

The XMLHandler scopes a source code object named SOAPAction.

When you specify a value in this source code, it instructs the SoapProcessor to call the XMLHandler when it receives an http-request with this value for the SOAPAction header.

Specify:

http://www.websydian.com/services/greet

In the source code:

Publisher.Services.XMLHandlers.GreetService.SOAPAction

Add code to action diagram

Create a message that can format the greeting string:

Source object	Verb	Target object
Publisher.Services.XMLHandlers.GreetService	message MSG	response
Publisher.Services.XMLHandlers.GreetService.response	parameter FLD	Request.Fields.firstName
	parameter FLD	Request.Fields.lastName

Specify:

Hello, &(1:) &(2:), welcome.

For the literal value of: Publisher.Services.XMLHandlers.GreetService.response

Open the action diagram for the function: Publisher.Services.XMLHandlers.GreetService

Add the following code (subroutine Handle Xml):

Post point XmlHandler functionality

```
Request.GetFirstOccurrence
   Map with:
   Variable Input:
      InputDocument<BodyInObjectStoreReference>
      InputDocument<BodyInDocument>
   Variable Parent:
      <ParentElement.NULL>

If Environment<*Returned status> == <Returned status.*Successful>
     Request.SingleFetch
          Map with:
          Variable Input:
               InputDocument<BodyInObjectStoreReference>
               Request.GetFirstOccurrence/Output<ObjectNode>

     If Environment<*Returned status> != <Returned
     status.*Successful>
          Set Response<FaultCode> = <FaultCode.Server>
          Set Response<FaultString> = <FaultString.Server>
     else
          Format message Message:
          Publisher.Services.XMLHandlers.response.GreetService,
          Local<Response.Fields.greeting>
               Map with:
               Request.SingleFetch/Data<firstName>
               Request.SingleFetch/Data<lastName>

          Response.InsertRow
               Map with:
```

```
            Variable Input:
                 OutputDocument<BodyOutObjectStoreReference>
                 OutputDocument<BodyOutDocument>
                 OutputDocument<BodyOutDocument>
            Data:
                 Local<Response.Fields.greeting>

If Environment<*Returned status> != <Returned
status.*Successful>
     Set Response<FaultCode> = <FaultCode.Server>
     Set Response<FaultString> = <FaultString.Server>
```

A short explanation of the code:

The GetFirstOccurrence function finds the top node in the request document.

The SingleFetch uses this position to read the information contained in this element.

At this point, you would enter your application specific functionality. To keep the tutorial simple, the only processing is that the formatting of the received data into the answer string for the response document.

The InsertRow call inserts the response data into the Response element in the response document.

If the document can't be read (e.g. if the structure is not as expected), a FaultCode and a FaultString is returned. The calling function uses these values to send an error message to the caller of the web service.

Create SoapProcessor function

The SoapProcessor is the function that receives the http request and retrieves the request document from the soap envelope that contains the request document itself.

The SoapProcessor uses "comprises" triples to determine which XMLHandlers it can call to handle the requests.

The SoapProcessor determines which XmlHandler function to call for a particular request by comparing the http header SOAPAction with the content of the SOAPAction source code scoped by the XmlHandlers.

Create the following triple:

Publisher.Services.SoapProcessor	comprises FNC	Publisher.Services.XMLHandlers. GreetService

Create definitions for the CreateWSDL function

In almost all cases, you must generate a WSDL file for the web service to make it possible for other applications to call it. To be able to create the WSDL, a few more definitions are necessary.

Create the triples:

Source object	Verb	Target object
Publisher.Services.XMLHandlers.	comprises FNC	Request.CreateSchema

Source object	Verb	Target object
GreetService		
Publisher.Services.XMLHandlers. GreetService	comprises FNC	Response.CreateSchema
Publisher.Services.SoapProcessor. CreateWSDL	implement SYS	Yes
Publisher.Services.SoapProcessor. CreateWSDL.HandleXmlHandlers	implement SYS	Yes

Create an ApplicationServiceListener

To make it possible for the web server component to communicate with the SoapProcessor, you must create a function that inherits from the abstract WSYSOAP/ApplicationServiceListener.

Create the triples:

Source object	Verb	Target object
TutorialListener	is a FNC	WSYBASE/ApplicationServiceListener
	implement SYS	Yes
	impl name NME	LISTTUT
	file name NME	LISTTUT

Open the action diagram of the TutorialListener function and add the following code:

Post Point: Call to EventDispatcher

```
Call Publisher.Services.SoapProcessor
```

Generate and build

Library Objects

Open the Generate and build settings – System definitions for the local PC.

Select “32 bit C++ build”.

Check the “Use pre-built libraries” checkbox.

Specify the following libraries:

Websyd.lib, WsydXml11.lib, WsydXml11_dll.lib

You find these three files in the Development folder of your Websydian installation (use the ones for your Plex version).

If you are using a **Plex 5.5 SP1** or earlier:

wininet.lib, ws2_32.lib.

These two lib files are part of the Visual Studio installation. Don't specify a path; the compiler knows where to find them.

For Header Directories, specify the "include" folder found under the Development folder of your Websydian installation.

Drag the following subject areas from the object browser to the Generate and Build window. Generate and build all the objects in the subject areas in one build.

WSYDOM/DOMObjectsToGenerateAndBuild

SDSTRING/SDStringObjectsToGenerateAndBuild

WSYHTTP/HTTPClientObjects

WSYSOAP/SOAPObjectsToGenerateAndBuild

WSYSOAP/WSDLObjectsToGenerateAndBuild

WSYXML/SchemaObjectsToGenerateAndBuild

WSYBASE/ DWA_Win_ObjectsToGenAndBuild

Application

Generate and build:

All objects scoped by the Publisher entity.

All objects scoped by the Request entity.

All objects scoped by the Response entity.

TutorialListener.

Create an exe file for TutorialListener.

Add the following to the TutorialListener.ini file:

[TransacXML]
MAX_CONTENT_LENGTH=100000
TEMPORARY_FILES=c:\temp\SoapProcessor

DELETE_TEMPORARY_FILES=N

Create an exe file for Publisher.Services.SoapProcessor.CreateWSDL

The MAX_CONTENT_LENGTH setting specifies the largest request document you will accept.

The TEMPORARY_FILES setting must specify an existing folder where the SoapProcessor can write temporary files.

DELETE_TEMPORARY_FILES=N specifies that the SoapProcessor should not delete the temporary files when the request has been processed. This is beneficial when developing and testing as the content of the temporary files often are useful for finding errors.

Deploy the service

You are now ready to do the final deployment of your Web Server in order to do so you will need to setup the Websydian DWA environment. You do this by completing the following steps.

Step 1: Ensure that required software is installed

Make sure that you have the SUN Java version 6 installed on your computer.

Then if you do not have Apache Tomcat version 6 installed please go to the Apache Tomcat download site (**http://tomcat.apache.org/download-60.cgi**) to obtain the installation program and install it keeping the default settings as proposed by the installation program.

After installing Tomcat, go to the Control Panel, select Administrative Tools and then Services. Locate the service named Apache Tomcat right click it and select Start to start the service.

Check that Tomcat is running by opening your browser and enter the URL **http://localhost:8080**. At this point, you should see the Tomcat Welcome screen.

Step 2: Download and extract the Websydian test environment

Download the file
http://www.websydian.com/wsyweb20/dwn/WindowsDWA_TestBench.zip

Then extract the content of the file to your Release directory located in the GEN directory where your local model is located.

Step 3: Setup the Websydian test environment

1. Open the file named 'DeploySettings.cmd' located in the 'TestBench' directory with notepad.
2. Change the value of the 'LISTENER_IMPL' name to LISTTUT (the implementation name of your implemented *ApplicationServiceListener)*
3. Verify that the value of the 'TOMCAT_PATH' points to your Apache Tomcat installation directory.

Step 4: Start the Websydian test environment

In order to start the Websydian test environment do the following:

1. Open the directory ‘TestBench’ located in your GEN\Release directory
2. Run the command ‘Deploy.cmd’ to deploy your application into the test environment (**Disregard the message “File not found - *.htm”**)
3. Run the command ‘Start.cmd‘ to start the test environment

Use the command ‘Stop.cmd’ to stop the test environment in a controlled manner.

If you make any changes to the application (such as creating new functions, changing existing ones or changing the content of the INI-file) you must:

1. Run the ‘Stop.cmd’ to stop the application
2. Run the ‘Deploy.cmd’ to deploy the changes
3. Run the ‘Start.cmd’ to restart the application.

Test the function

One of the common issues when you are developing services is to test the services you have created.

To test a service, you need to be able to make an http-request containing a valid SOAP-document to the service. You can make such a request in several different ways:

1. Generate the WSDL for the service (See below). Feed this WSDL to an external tool (e.g. XMLSpy or soapUI) and let the tool generate a request.
2. Create a test program that calls a SoapGenerator that calls the service (See the tutorial “Calling a web service”).
3. Use an http test client tool that is available at the Websydian download application.

The tutorial will show how to use the test client, as the first option is dependent on access to an external tool, while the second option does lead to some extra work.

You can download the tool here:

http://www.websydian.com/wsyweb20/dwn/ServiceRequestTestTool.zip

You can find the documentation for the tool here:

http://www.websydian.com/websydiandoc/v61/source/WebsydianExpressv3.0/Tutorials/testclient.htm

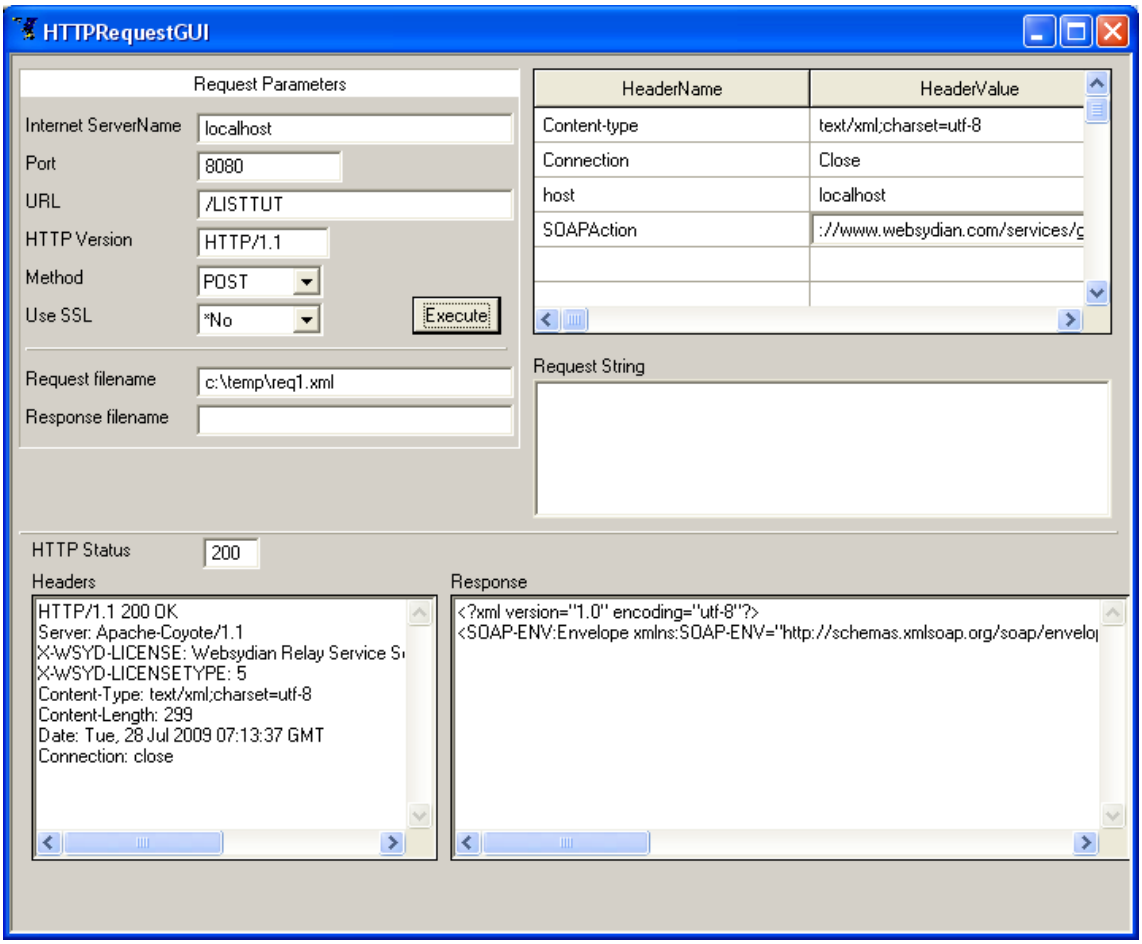
Create test document

Use notepad to create a text file named TestRequest.xml - with this content:

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
       <SOAP-ENV:Body>
            <ws:Request xmlns:ws="http://www.websydian.com/services/greet">
                <ws:firstName>John</ws:firstName>
                <ws:lastName>Doe</ws:lastName>
            </ws:Request>
       </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Use the test client to call the service

Run the AAvf.exe to start the tool.



Enter/select:

Internet Server Name: localhost (The server your Tomcat is installed on)

Port: 8080 (The port your Tomcat is using)

URL: /LISTTUT

Http Version HTTP/1.1

Method: POST

Use SSL: No

Request filename: The name of the file containing the XML document

In the header table (top right corner):

Change the value for the Content-type header to “text/xml;charset=utf-8”

Enter a new header:

HeaderName: SOAPAction

HeaderValue: http://www.websydian.com/services/greet

Press Execute to test the application.

You can see the result of the test in the bottom part of the panel.

The HTTP Status specifies whether the service returns an error (200 is successful).

The “Response” pane shows the XML document returned by the service, which should have the following content (it will not be shown formatted in the tool):

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
       <p1:Reponse xmlns:p1="http://www.websydian.com/services/greet">
          <p1:greeting>Hello, John Doe, Welcome.</p1:greeting>
       </p1:Reponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Problem solving

1. Check folder for temporary files.

See LISTTUT.ini for the location of the folder.

If the folder does not exist, create it and try again.

2. Check whether a Request document has been created.

The Request documents have names that start with “SoapIn”.

If there are no such documents in the temporary files folder, it indicates that the request has never reached the service – or that the request did not contain a document.

3. Check that the Service has been deployed on the expected URL

You can check whether the service is available on the URL using your browser.

If you are using Internet Explorer, you need to first Select Tools→Internet Options→Advanced and **un**check the “Show friendly HTTP error messages” option.

Enter the URL in the browser. This shows an XML-document stating that the request is invalid in the browser.

This should also write two temporary files to the temporary folder

4. Check whether a Response document has been created

If you find a request document – but no response document, the most common reason is that an error in the XMLHandler has meant that the InsertRow function has not been called – often that the XMLHandler has ended in a state where the program can’t terminate.

Debug the XMLHandler to check that it behaves as expected.

Note that you can use the Plex action diagram debugger for this.

5. If the returned XML document is a fault document

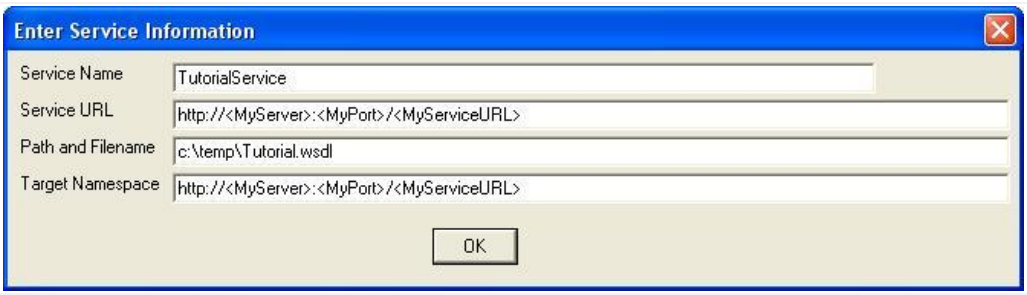
If the service returns a SOAP-fault document, it can state different things about the error.

The most common case is that the maximum content-length has been exceeded. This is normally an indication that you have not entered MAX_CONTENT_LENGTH setting as specified above.

Generate the WSDL

The WSDL file is not necessary for this tutorial. However, as a WSDL file is usually necessary if other applications are going to call the service, this section of the tutorial shows how you can create a WSDL file for the service.

Open the “Generate and Build” window and run the CreateWSDL function



Enter:

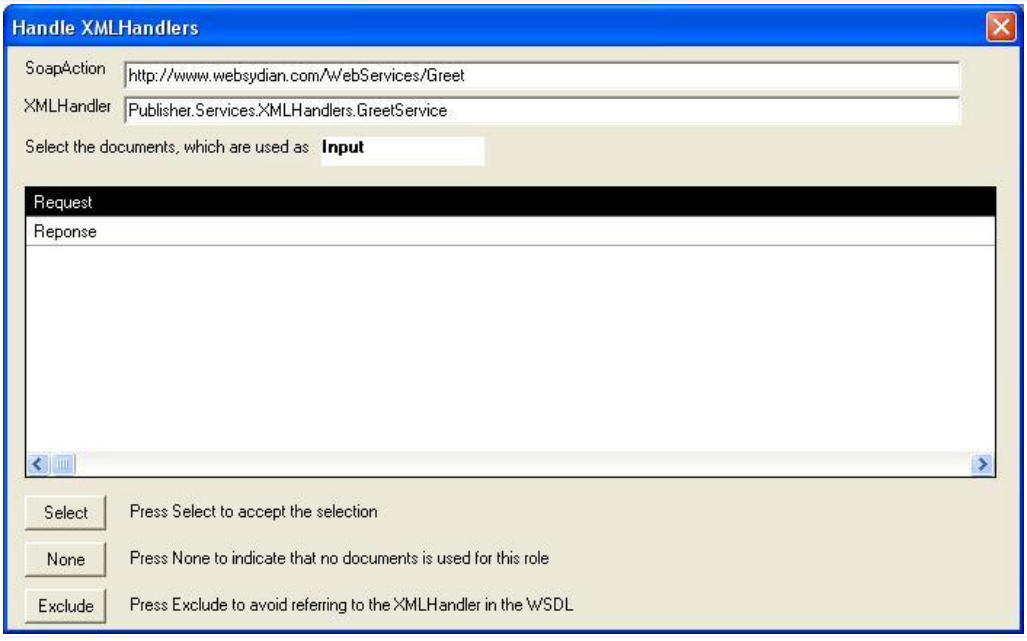
TutorialService

http://localhost:8080/LISTTUT

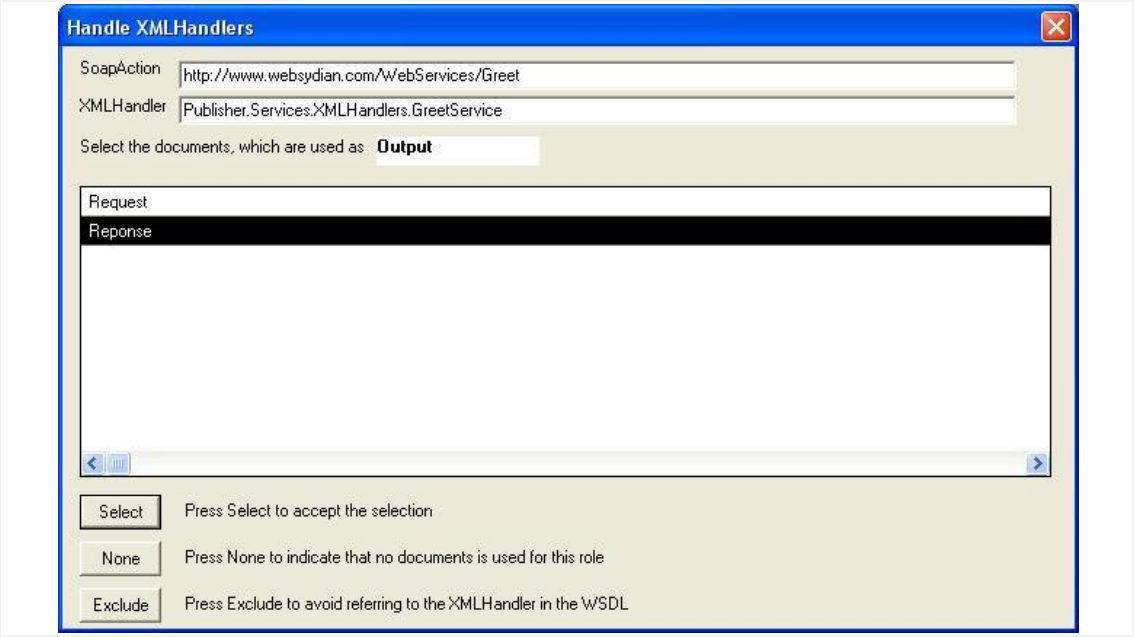
Where to store the WSDL file

Use the default for the target namespace

Press OK



Select the Request document



Select the Response document

This generates two files: The WSDL file and a schema file that contains the definitions for the request and response documents.

If you have access to XMLSpy, soapUI or another tool that can generate an http request based on a WSDL file, you should try using the generated WSDL file for this purpose.