WEBSYDIAN™

WebsydianExpress Web Service Tutorial

Providing a web service in WebsydianExpress

Revision date 14 August 2009

Table of Contents

Overview

This document shows how you can create and deploy a simple web service using TransacXML and WebsydianExpress.

The tutorial will go through the following steps:

- Define the structure of the request and the response documents in Plex.
- Create a function that can handle the received request document and create the response document.
- Add the service to the WebsydianExpress service structure
- Create a WSDL file for the service

Compared to TransacXML without WebsydianExpress

The tutorial shows the solution to the same task as the TransacXML tutorial “Providing a web service”.

If you compare the two tutorials, you will see that the main difference is in the deployment of the service.

Development Language Variants

This tutorial describes the solution for the Windows version of WebsydianExpress.

If you want to use the Win/iSeries version, the only thing you have to do is to change the variant of the WSYAPI version to AS/400 RPG Server.

If you want to use the Java version of WebsydianExpress, you will have to change the variant of WSYAPI to Java JDBC, the variant of WSYBASE to DWA Java, and the variants of WSYDOM and SDSTRING to Java. In addition to these changes, there are some differences when it comes to setting up the “generate and build” settings – and deploying the generated objects.

You can’t use the WebsydianExpress for iSeries variant for handling XML documents using TransacXML.

Preparations

Creating a local model

Before you are able to run this tutorial, you have to create a Plex local model that has the WebsydianExpress XML pattern library (WSYAPIWS) 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 WSYAPIWS 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*

Model	Variant	Version	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*
VALIDATE	Base	*Latest Version*	*Latest Level*
WINAPI	Base	*Latest Version*	*Latest Level*
WSYAPI	ODBC Server	v.3.0	v.3.0
WSYAPIWS	Base	v.3.0	v.3.0
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

Installing WebsydianExpress

If you do not have an existing WebsydianExpress installation, you will have to download WebsydianExpress from **www.websydian.com** and install it.

In the download application in the Support section, select WebsydianExpress v3.0 for Windows for the Plex version you want to work with, download the install application and run it.

You can find more information about installing WebsydianExpress in the WebsydianExpress documentation.

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
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 makes it possible 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 an XMLServiceHandler

The XMLServiceHandler is the function that reads the content of the request document, performs the necessary business logic, and creates the response document.

The XMLServiceHandler does **not** handle the SOAP-wrapping of the message. WebsydianExpress handles this based on the definitions made in the service structure (see below).

Enter the following triples to define the XMLServiceHandler:

Source object	Verb	Target object
GreetServiceHandler	is a FNC	WSYAPIWS/ XMLAbstract.XMLServiceHandler
GreetServiceHandler	local FLD	Response.Fields.greeting
GreetServiceHandler	implement SYS	Yes
GreetServiceHandler	file name NME	MyGreet
GreetServiceHandler	impl name NME	MyGreet

Add code to the action diagram

Create a message that will be used to format the greeting string:

Source object	Verb	Target object

Source object	Verb	Target object
GreetServiceHandler	message MSG	Greeting
GreetServiceHandler.Greeting	parameter FLD	Request.Fields.firstName
GreetServiceHandler.Greeting	parameter FLD	Request.Fields.lastName

Specify:

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

For the literal value of: GreetServiceHandler.Greeting

Add the following code to the Handle Request Post Point of the GreetServiceHandler:

```
Call Request.GetFirstOccurrence
   Map with:
   Variable Input:
      InputDocument<XMLAPIFields.WSObjectStoreReference>
      InputDocument< XMLAPIFields.WSObjectDocument>
   Variable Parent:
      <ParentElement.NULL>

If Environment<*Returned Status> != <*Returned Status.*Successful>
      Set  Environment<*Returning status> = <*Returning
      status.*Error>
      Go Sub Terminate

Call  Request.SingleFetch
   Map with:
      InputDocument<XMLAPIFields.WSObjectStoreReference>
      Request.GetFirstOccurrence/Output<ObjectNode>

If Environment<*Returned Status> != <*Returned Status.*Successful>
      Set  Environment<*Returning status> = <*Returning
      status.*Error>
      Go Sub Terminate

Format Message  Message: GreetServiceHandler.Greeting,
Local<Response.Fields.greeting>
   Map with:
   Request.SingleFetch/Data<Request.Fields.firstName>
   Request.SingleFetch/Data<Request.Fields.lastName>

Call Response.InsertRow
   Map with:
   Variable Input:
      OutputDocument<XMLAPIFields.WSObjectStoreReference>
      OutputDocument<XMLAPIFields.WSObjectDocument>
      OutputDocument<XMLAPIFields.WSObjectDocument>
   Variable Data:
      Local<Response.Fields.greeting>
```

```
If Environment<*Returned Status> != <*Returned Status.*Successful>
     Set  Environment<*Returning status> = <*Returning
     status.*Error>
     Go Sub Terminate
```

A short explanation of the code

The GetFirstOccurrence function finds the top element in the request document (The Request node).

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

At this point, you would enter your application-specific functionality. In this tutorial, you just create a response using the values from the request document.

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

Error handling for SOAP services

In the example, the *Returning status is just set to *Error to indicate that the handling in the service handler has encountered an error-.

If this service handler is called as part of a SOAP service, WebsydianExpress will create a generic soap fault message when the service handler returns an error.

If you want to return a more specific error message, you can specify a value for the faultstring of the soap fault by calling the XMLAPI._Parameters.SetSoapFault API (Placed in WSYAPIWS).

Generate and build

Generate and build settings

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, wsexpress.lib

These four files are found in the Development folder of your Websydian installation (use the ones for your Plex version).

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

Generate application

Generate and build:

All functions scoped by the Request and Response entities.

MyXMLServicehandler

Deploy the service

Move objects to application folder

Move the generated objects to the folder “Application Service/PlaceObjectsHere” that is scoped by the WebsydianExpress installation.

Publish service

To make the service available, you must add it to the service structure of a site in the WebsydianExpress installation.

In the following, the service is defined in the basicsite. Note that adding a service **only** makes it available in the site where you added it.

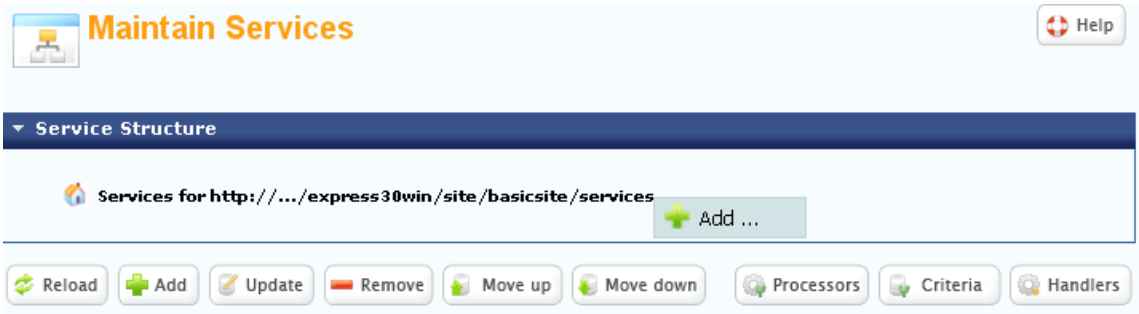
Open the administration interface of the basicsite of WebsydianExpress and login using an administrator profile.

Click on the menu item Web services→Web services.

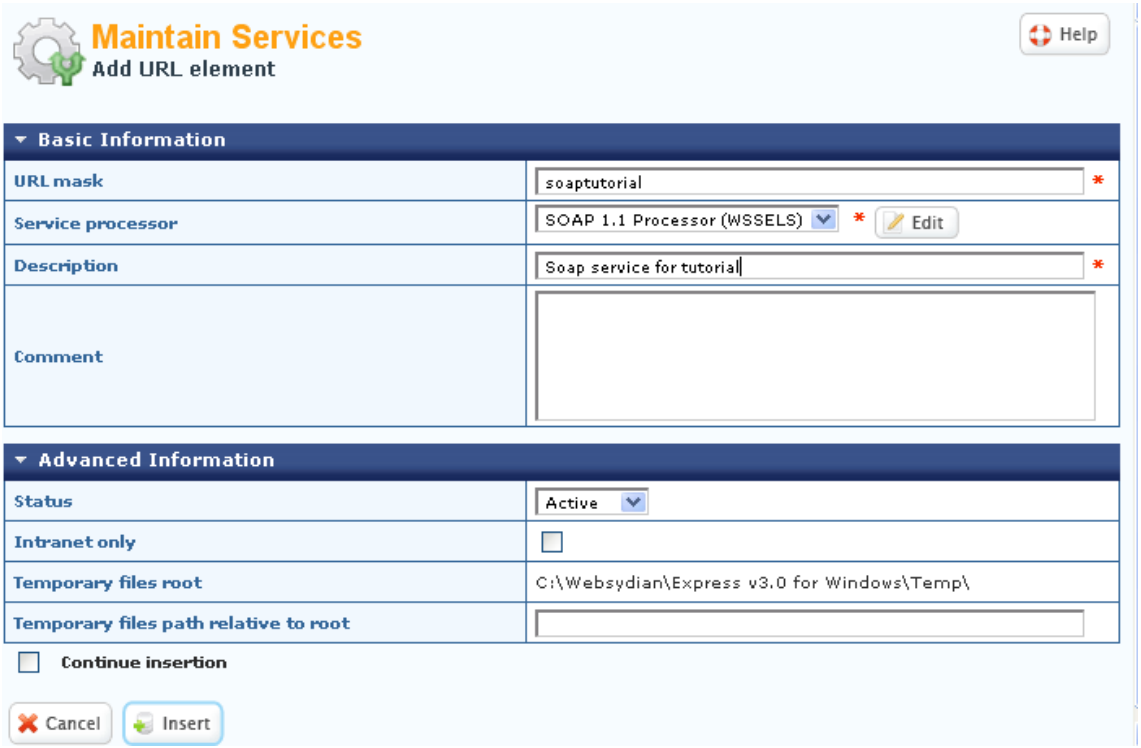
Adding a URL element

The first part to add to the service structure is a URL-element. This specifies two things:

1. A URL extension of the service-URL for the site.
2. The service processor that will pre- and post- process the request and response documents. In this case, this processor will handle the unwrapping of the SOAP request and the wrapping of the SOAP response.



Right-click on the root of the structure and select Add



Enter/select:

URL mask: soaptutorial

Service processor: SOAP 1.1 Processor (WSSELS)

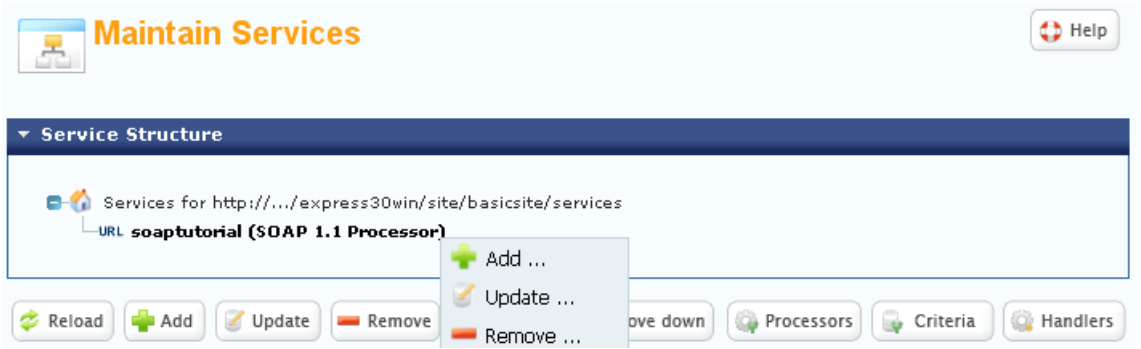
Description: Soap service for tutorial

Press Insert

Adding a criterion

Each URL-element can provide several services. This means that WebsydianExpress need to know how to determine which service to call. You define this by adding a criterion to the URL-element.

As this is an example of how to provide a SOAP-based service, the SOAPAction criterion is the relevant criterion to use.



Right-click on the created URL element and select Add.

Select SOAPAction as criterion.

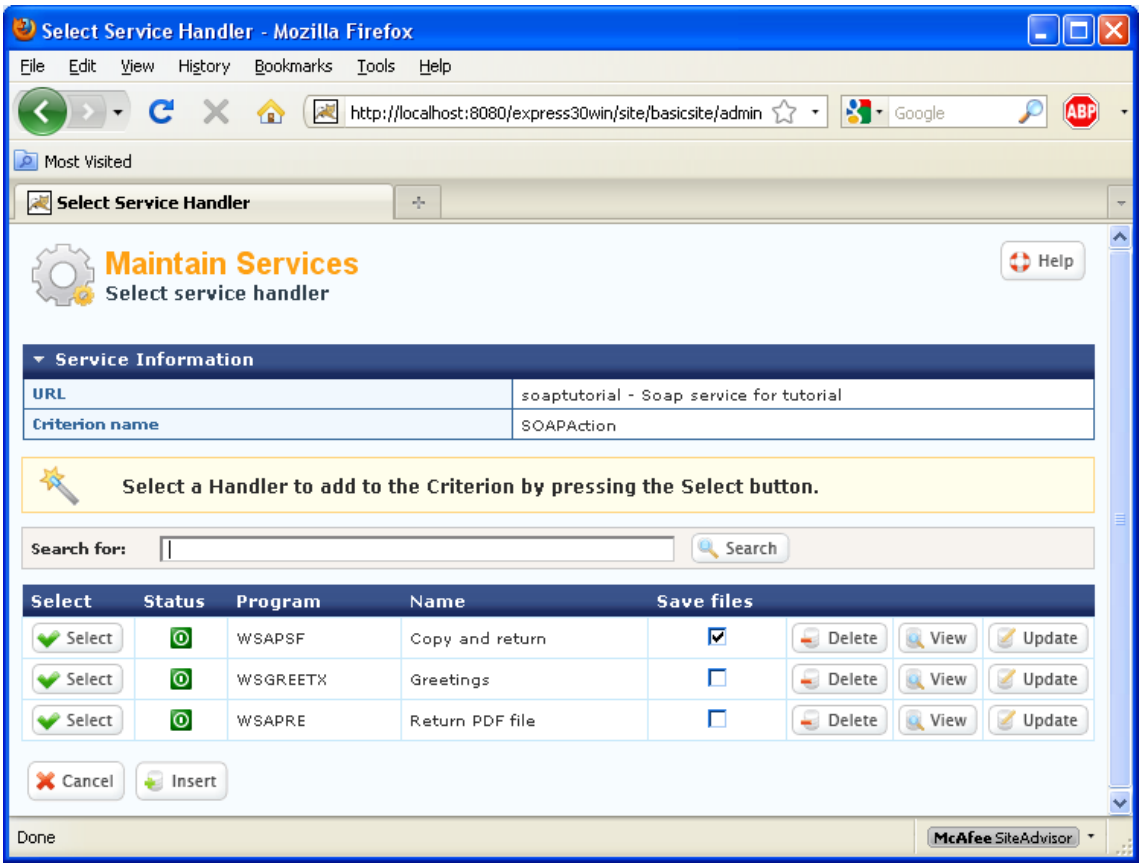
Press Insert.

Adding the service handler

The last thing to add is the service handler itself. The service handler specifies which program WebsydianExpress should call.

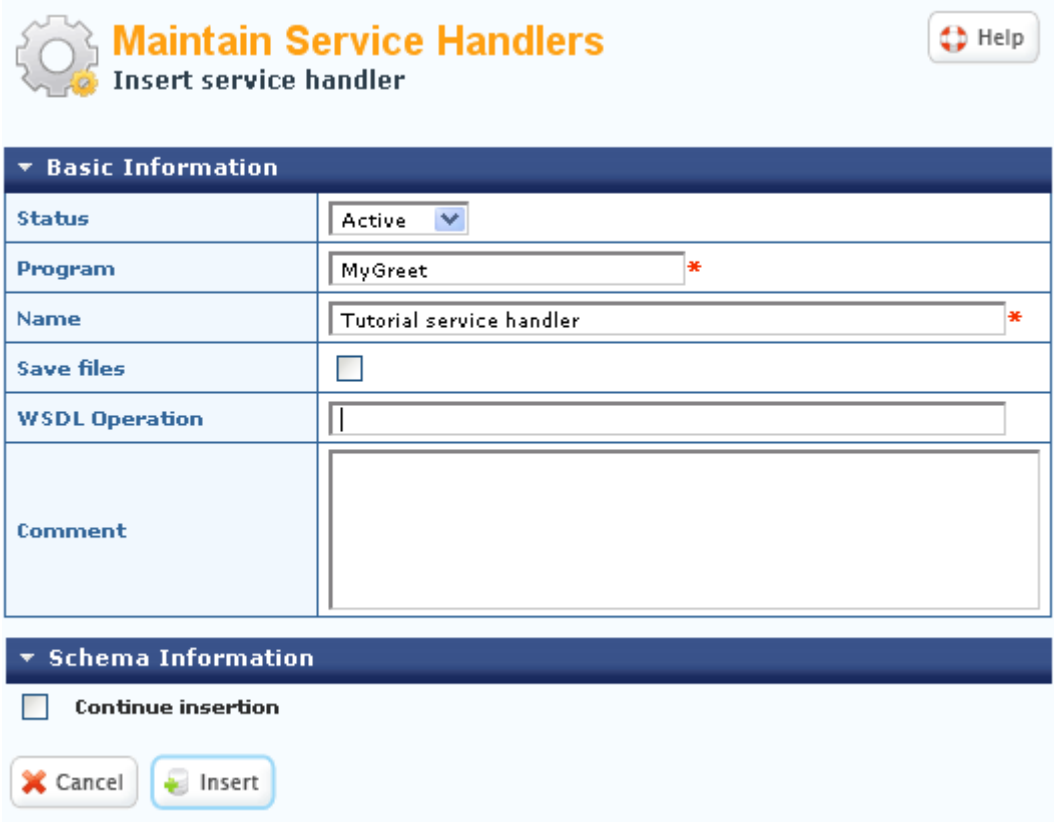
The service handler is added to a criterion. When you add the service handler, you specify the value the criterion must have for the service handler to be selected.

Right-click on the SOAPAction criterion and select Add.



On this page, you can select existing service handlers. In this case, the service handler does not yet exist so you must create it first.

Press Insert to go to the service handler create page.

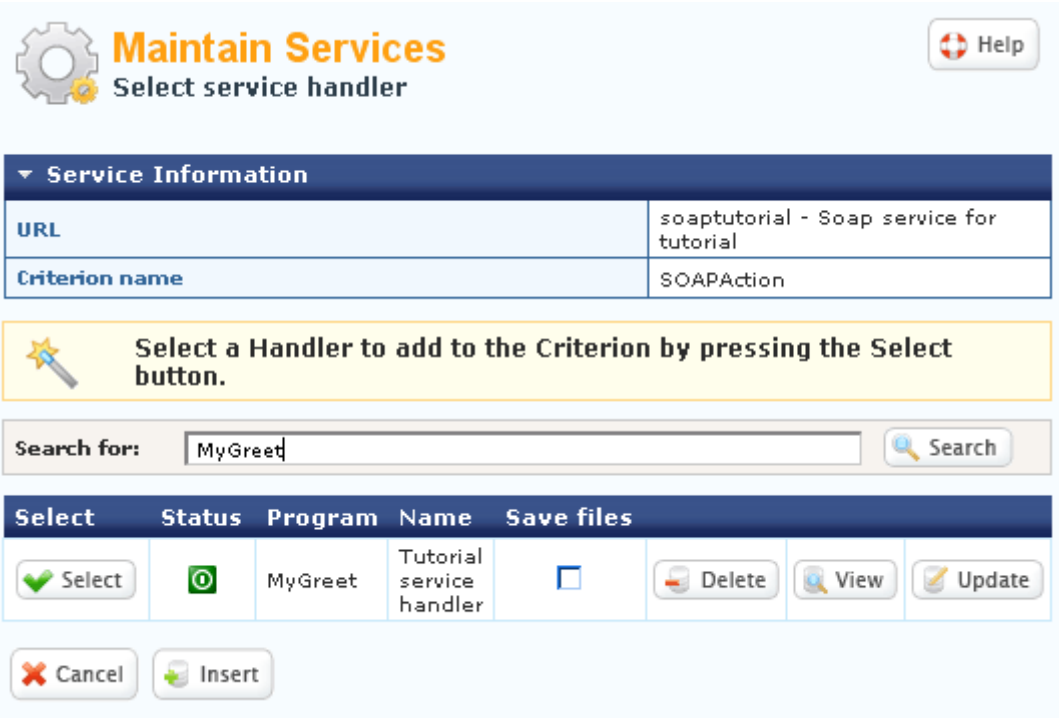


Enter

Program: MyGreet (This is the implementation name of the service handler)

Name: Tutorial service handler

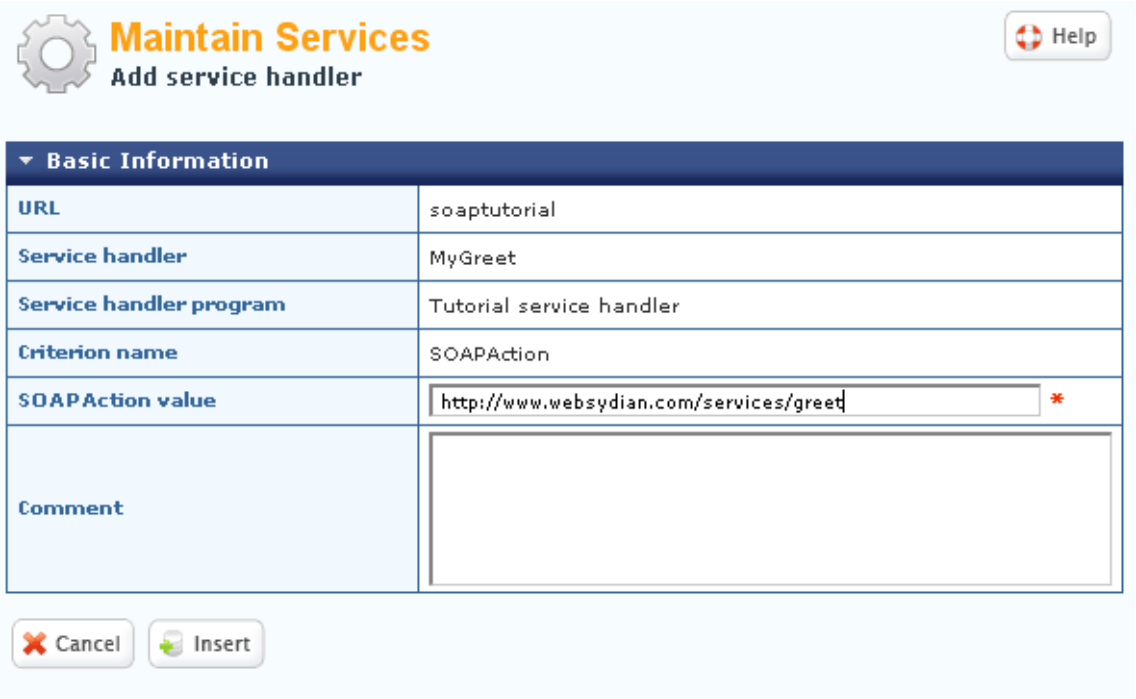
Press Insert.



This reloads the service handler grid - with the new handler available.

Press Select for the service handler.

This opens a pop-up window. Here you specify the SOAPAction for the service handler.

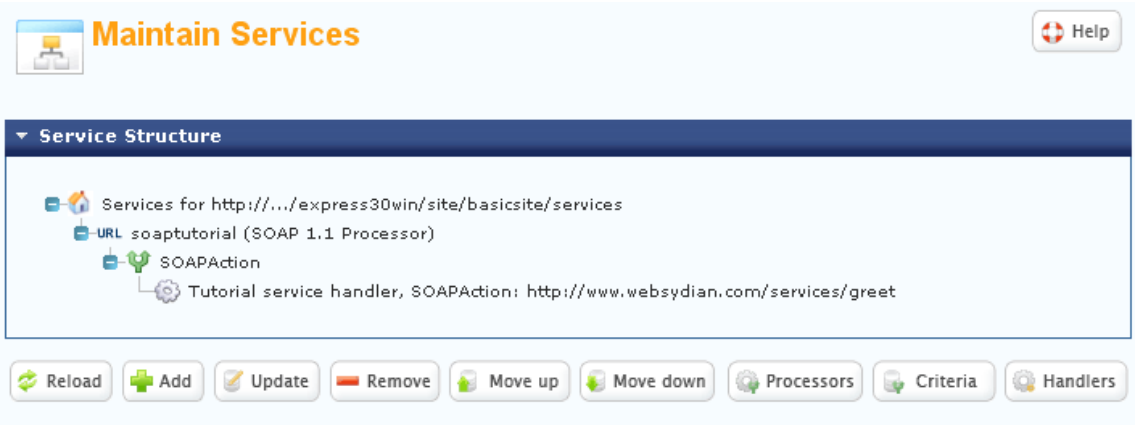


Enter:

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

Press Insert.

This results in the following structure:



At this point, the service is available.

This structure means that:

1. Requests sent to the URL http://.../express30win/site/basicsite/services/soaptutorial will be handled as soap requests.

2. The SOAPAction http-header will be used to determine which service handler to call.

3. The Tutorial service handler will be called when the SOAPAction header is http://www.websydian.com/services/greet

Testing the service

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:

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

You can find the documentation for the tool here:

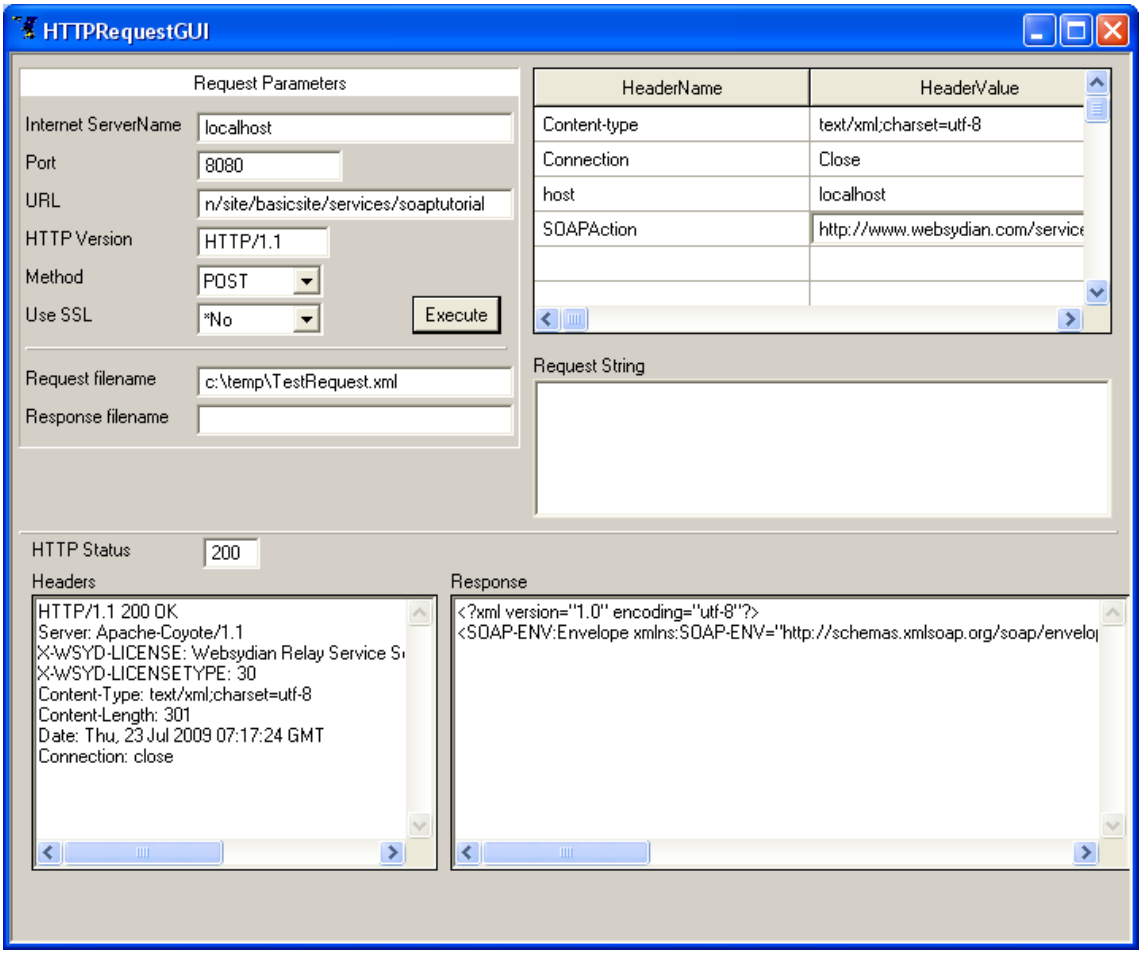
http://www.websydian.com/websydiandoc/v61/source/WebsydianExpressv3.0/Tutorials/testcient.htm

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>
```

Run the test client



Enter the following values:

Internet ServerName: localhost

Port: 8080 (The port your Tomcat is using)

URL: /express30/site/basicsite/services/soaptutorial

Method: POST

Request filename: Specify the file containing the test document.

Add an entry to the Header table (the table in the top right corner)

HeaderName: SOAPAction

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

Change the value for the Content-type entry to: text/xml;charset=utf-8

Press Execute to test the service.

When you call the service, the response is shown in the bottom half of the tool:

HTTP Status: 200 (if successful).

Headers: Shows all the headers that the service has returned.

Response: The returned XML document (if successful).

Check whether the HTTP status is 200 and the returned document is correct.

Problem solving

Check message log

If an error occurs, start by going to the WebsydianExpress message log in the administration interface and check whether any errors have been reported.

Check setup of site

Go to Site configuration→Site settings. Check that the value specified for “URL extension for identifying service request” is “/services”.

Go to Global settings→Global settings and check whether the “Temp file location” is an existing folder and whether the Request log is enabled.

Check temporary files

When the request log is enabled – the temporary files that are generated when a request is handled are saved. Go to the temporary files folder and check whether the request has generated any files.

- If there are no files

  Check again if the requestlog is enabled.

  The most likely cause is that the URL is not correct – check that you have specified the correct server name (if you are not running on localhost) and port. Check the URL.

- If there is only a request file (IN_nnnn.xml)

  If there is no message in the log that states that WebsydianExpress was unable to identify/call the service handler, you should try debugging the service handler to find out if it is called – and if any error occurs.

- If there are both request (IN_nnnn.xml) and response files (OUT_nnnn.xml)

  The number following IN_ and OUT_ is the request id. This means that the request and response files for one request always will have the same number.

  In the case where both files exist – check whether the response is empty. If it is, debug the service handler to check where the error occurs.

Creating a WSDL for the service

Even though the service is available, most service consumers (clients) will not be able to call the service without having a formal description of the service.

For SOAP-based services, the de facto standard is to use WSDL files for this purpose.

Generate schema files

To be able to generate a WSDL file for the service, you will need to start by generating W3C schema definitions for the request and response documents.

Create exe files for

Response.CreateSchema and Request.CreateSchema.

Deploy the files to the PlaceObjectsHere folder.

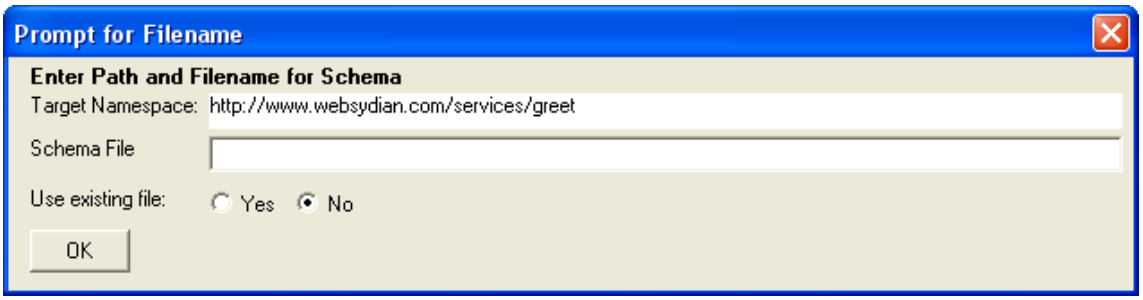
Create a BAT file that contains the following lines of code using notepad.

Replace with the actual implementation names of the two CreateSchema functions.

```
SET PATH=%PATH%;..\WebsydianExpress;..\WebsydianExpress\Runtime
[implementationname of Request.CreateSchema].EXE
[implementationname of Response.CreateSchema].EXE
```

After saving the BAT file, use Explorer to run it.

When you run one of these exe file, the following panel will open.



The “Target Namespace” field is a pre-filled output field that informs you which namespace the generated schema will describe.

As the two documents belong to the same namespace, we recommend that you save the definitions to the same file (as one W3C schema describes one namespace).

You do this by specifying the same filename when running both exe files – and set the “Use existing file” to “Yes” for the second exe.

Run the Response.CreateSchema exe file.

Specify c:\temp\greet.xsd for “Schema File”.

Run the Request.CreateSchema exe file.

Specify c:\temp\greet.xsd for “Schema File”. (or choose another folder).

Set “Use existing file” to Yes.

This generates a schema file with the following content. This schema file describes both the Request and the Response documents.

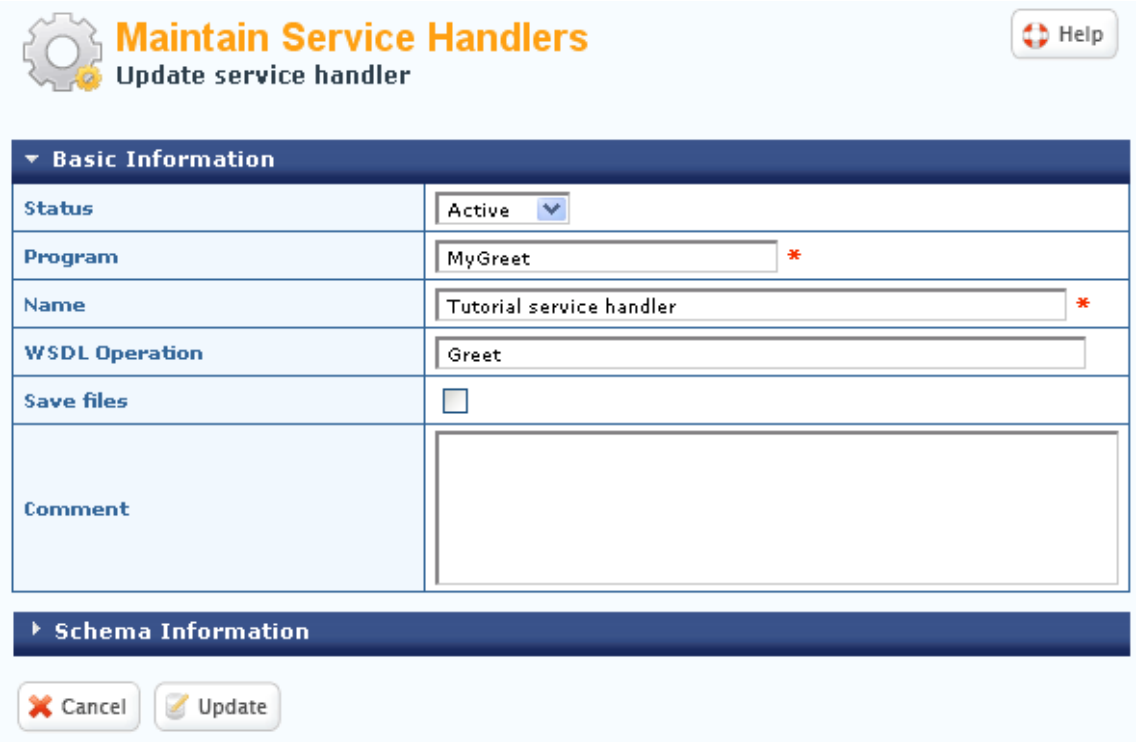
```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.websydian.com/services/greet" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
- <xs:element name="Request">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="firstName" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="lastName" type="xs:string" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
- <xs:element name="Response">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="greeting" type="xs:string" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Generate WSDL

Open the menu item Web services→ Web services in the administration interface.

Press the “Handlers” button.

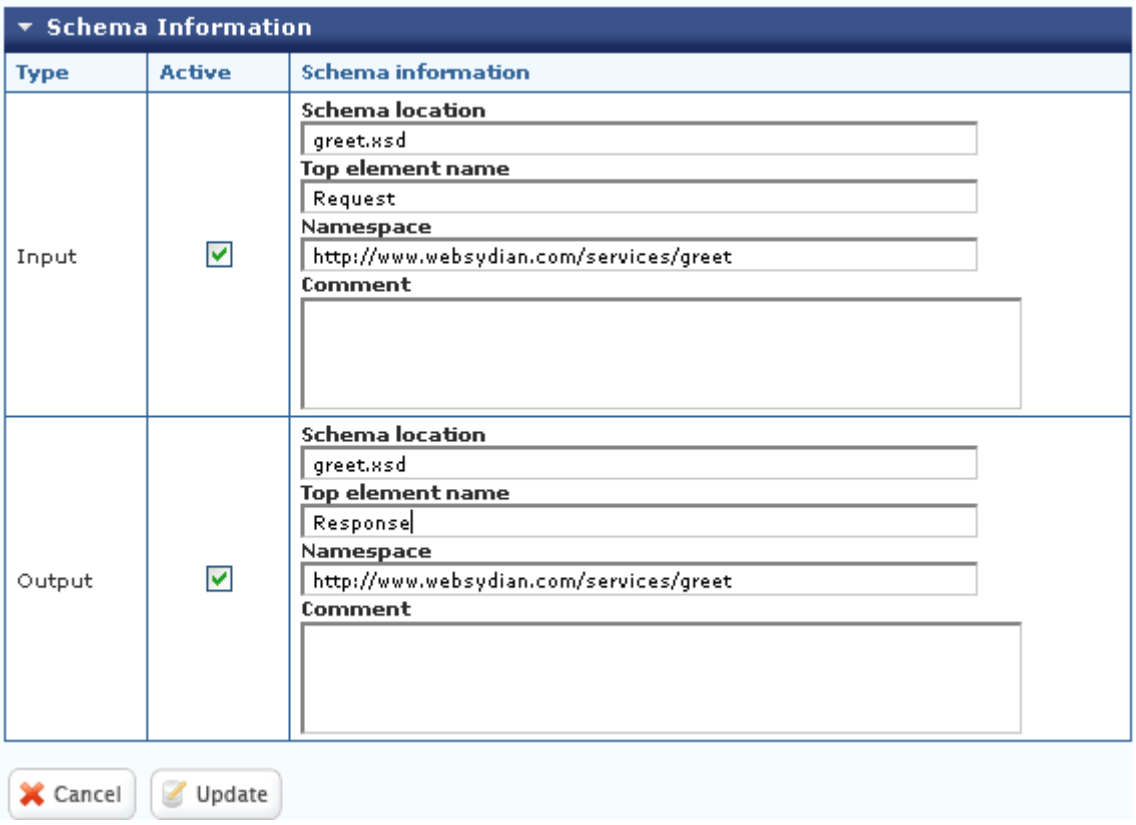
Press the update button for the MyGreet handler.



Enter:

Greet for “WSDL Operation”.

Open the Schema Information section by clicking on the blue bar.



Enter:

Input/Schema location: greet.xsd

Input/Top element name: Request

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

Output/Schema location: greet.xsd

Output/Top element name: Response

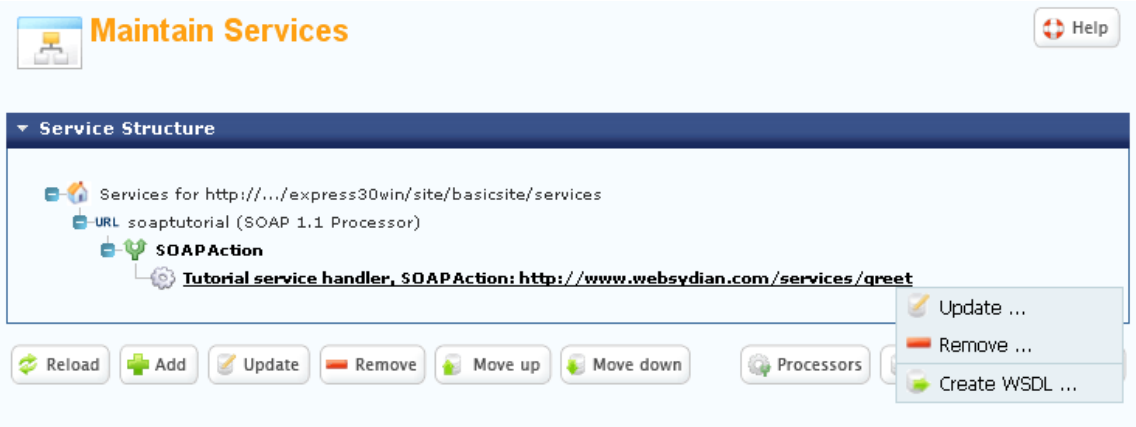
Output/Namespace: http://www.websydian.com/services/greet

Press Update

This updates the service handler with information about the expected input and output format for the handler.

Return to the service structure.

Select the service handler, right-click and select Create WSDL



This pops a panel that prompts you to open or save the generated WSDL.

> Choose to save the file.
>
> Save the file to the folder where you saved the schema file.

At this point, you have a WSDL file and a W3C schema file.

The WSDL file refers to the schema file.

You must keep the WSDL and the schema file in the same folder for this reference to work.

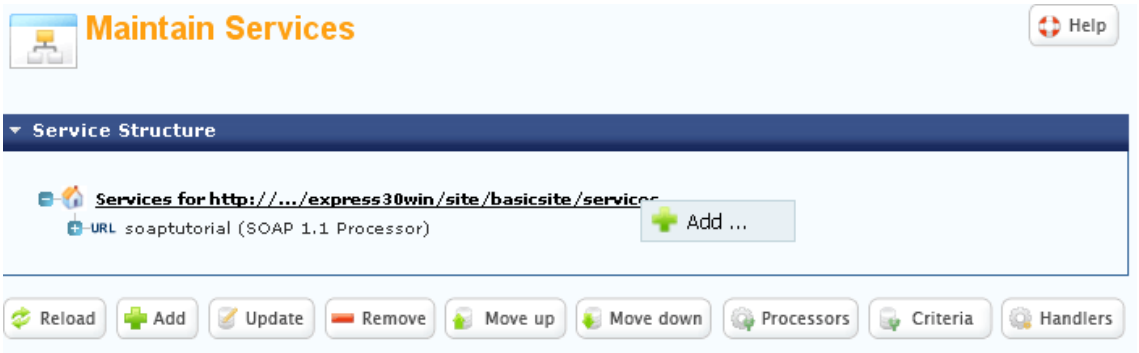
If you have access to a tool that can use a WSDL to generate an example request (e.g. XMLSpy or soapUI), try using the WSDL to call your service.

Publishing as a non-SOAP service

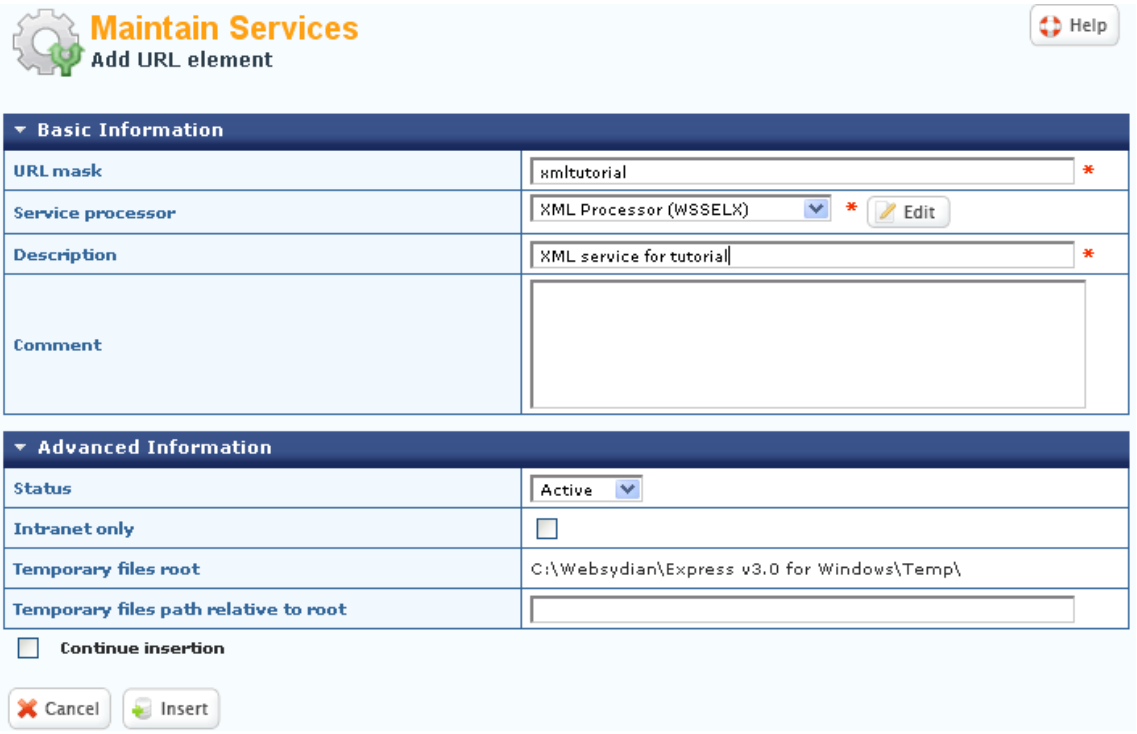
This part of the tutorial explains how you can use the same service handler to provide the service without SOAP.

Adding the URL element

You must create a new URL element.



> Right-click on the service root element and select Add.



Enter/select:

URL mask: xmltutorial

Service processor: XML Processor (WSSELX)

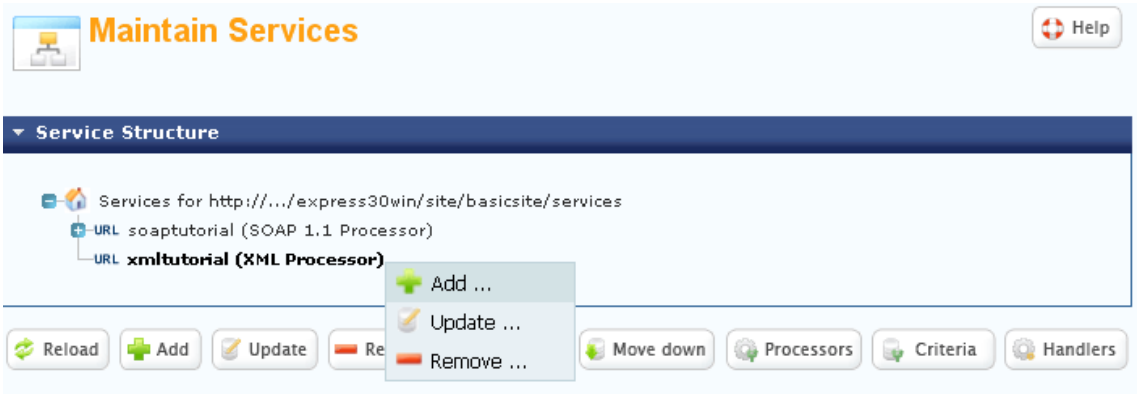
Description: XML service for tutorial

Press Insert.

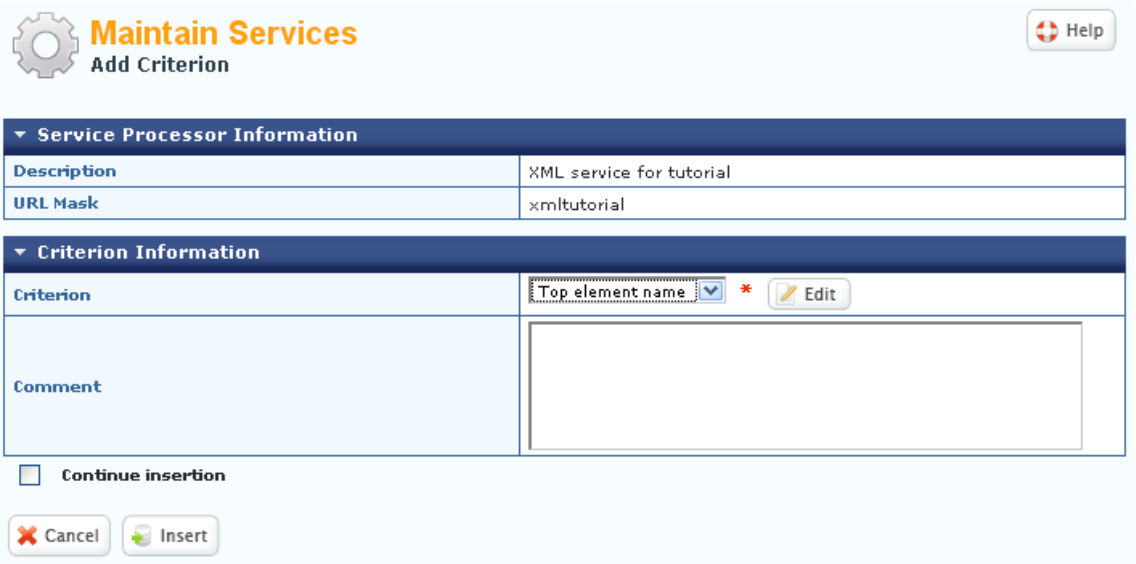
Add criterion

As this is not a SOAP-based service, using the SOAPAction criterion would be unnecessarily confusing.

Instead, the name of the top element of the request document will be used to determine which service to call.



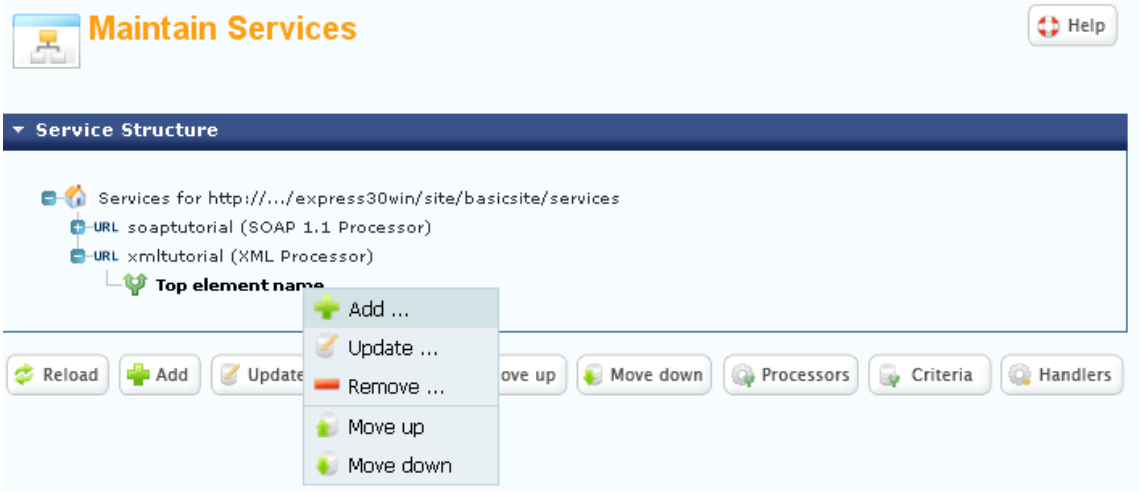
Right-click on the xmltutorial URL element and select Add.



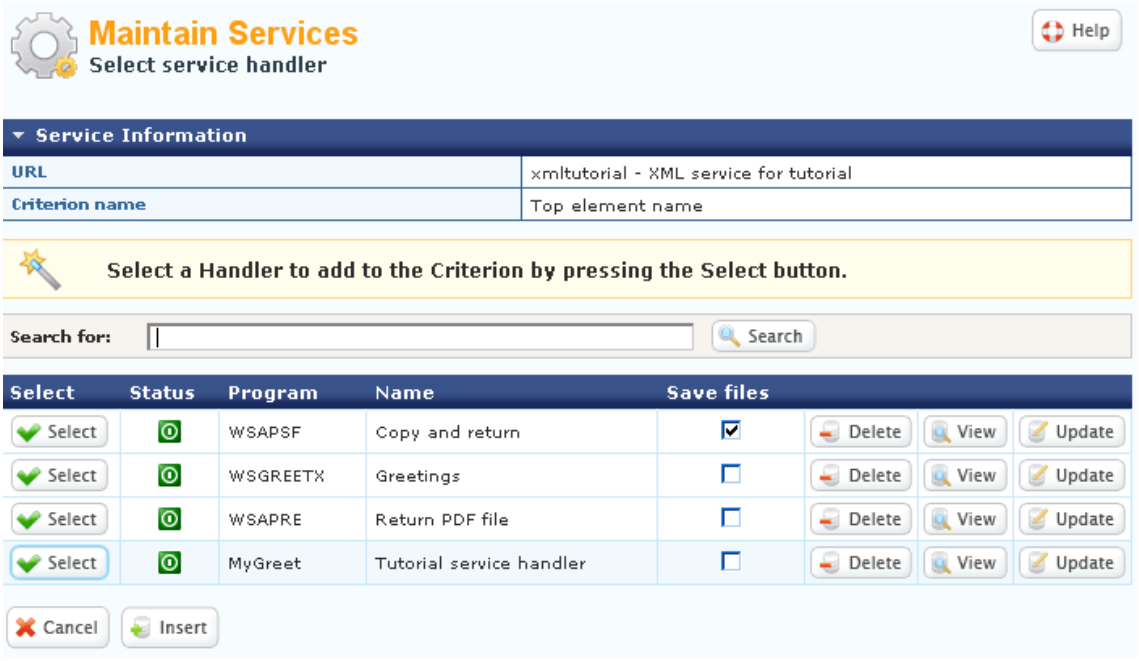
Select the Top element name criterion.

Press Insert.

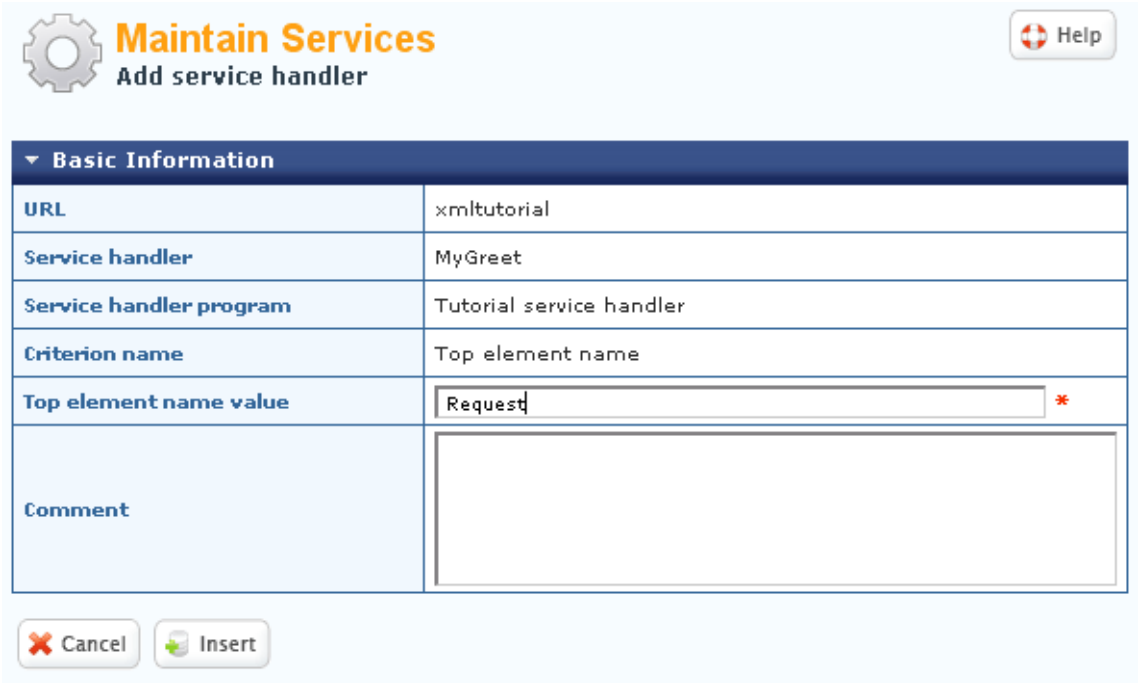
Add the service handler



Right-click on the Top element name criterion and select Add.



Select the service handler MyGreet .



Enter:

Top element name value: Request

Press Insert.

This gives this structure:

Test the XML service

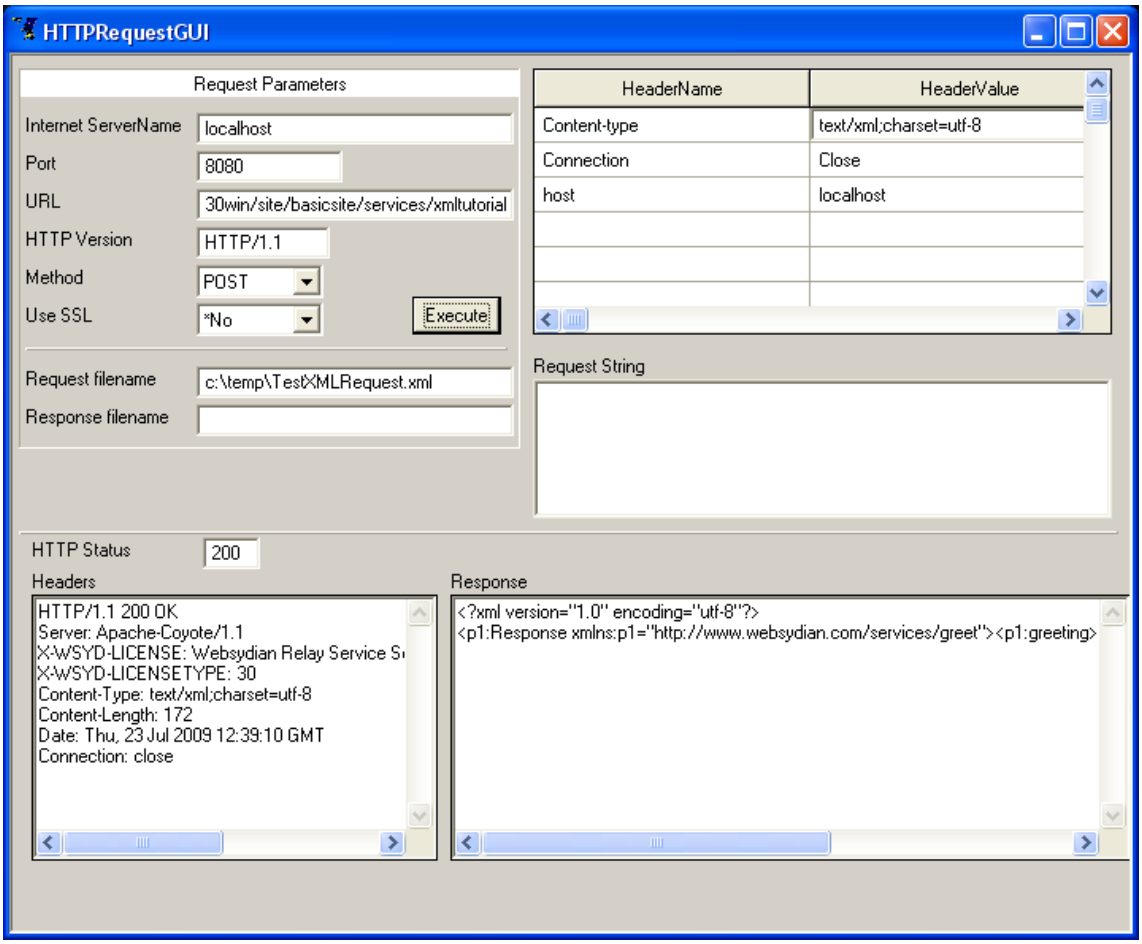
Create a test document

Use notepad to create an xml file named TestXMLRequest.xml with the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<ws:Request xmlns:ws="http://www.websydian.com/services/greet">
          <ws:firstName>John</ws:firstName>
          <ws:lastName>Doe</ws:lastName>
</ws:Request>
```

Run the test tool

Run the http-client test tool.



Enter the following values:

Internet ServerName: localhost

Port: 8080 (The port your Tomcat is using)

URL: /express30/site/basicsite/services/xmltutorial

Method: POST

Request filename: Specify the file containing the test document.

Change the value for the Content-type entry to: text/xml;charset=utf-8

Press Execute to test the service.

Notice that the response has no SOAP-envelope around it.

In this way, you can let the exact same function service SOAP and non-SOAP services.

As the WSDL generation of WebsydianExpress only works for SOAP-based services, you can't choose to generate a WSDL for this service.