WEBSYDIAN™

Web Service Tutorial

Calling a web service

Revision date 14 August 2009

Table of Contents

Overview

This tutorial will show you how to create the necessary definitions and programs for calling a SOAP-based web service.

The web service is hosted at the Websydian site. This means that to test the application, you will have to have access to the Internet.

For each call of a web service, three separate steps are performed:

1. An XML document containing the request data are created
2. An http request containing the request document is sent to the service provider – and a response containing a response document is received.
3. The response document is read and the data contained in the document is retrieved.

To be able to create programs that perform these three steps, you have to first define the structure of the request and the response document in Plex.

The tutorial will show you how to use a WSDL file describing a specific web service to generate the necessary definitions of the XML documents and will show how you make the programs necessary for calling the web service.

Document description

This example document shows the structure of the request document that will be created and sent to the web service:

```
<?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>
```

This is a simple document that just contains the first and last name of a person.

This example document shows the structure of the response document that will be returned by the service and read by the program calling the service:

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

The one thing in the two documents that might be a bit confusing is:

xmlns:p1="http://www.websydian.com/services/greet".

This is a namespace declaration, which just specifies that everything that is prefixed with p1 belong to the namespace:

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

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	WinC	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*
WSYBASE	DWA – Windows	v.6.1	v.6.1
WSYDOM	MSXML	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 TransacXML tool

To be able to use the TransacXML import tool to create the definitions of the request and the response documents, you have to install the tool.

Download the latest version of the import tool from the download application at: **www.websydian.com**

If you are not already a registered user, you will have to register to access the download application.

Run the downloaded exe file to install the tool.

Importing document definitions

Explanation

The first step is to define the structure of the request and the response document in the Plex local model.

In most cases, the service provider determines the structure of the response and request documents.

The description of the request and response documents can be in many forms, but as the use of web services has evolved, the WSDL (Web Service Description Language) standard has become the most common way to describe services.

The web service used by this tutorial also provides a WSDL; this WSDL can be found here:

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

Websydian provides an import tool that makes it possible to import the structure of the request and response documents described in WSDL files.

The installation of this tool is described above (in the preparations section).

The installation will have created a shortcut to the import tool in the start menu under Websydian/TransacXMLImporters.

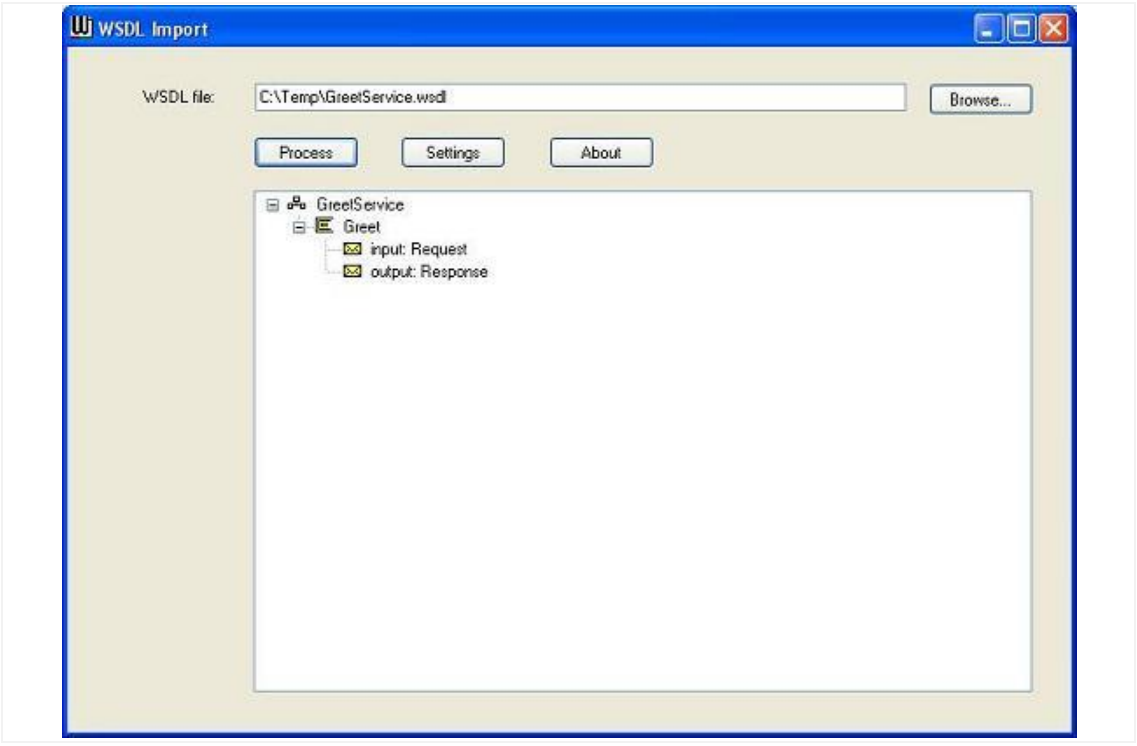
Import the document definitions

Run the installed wsdl import program.

Enter the location of the wsdl file:

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

Press “Process” to parse the WSDL.



The tree structure shows a representation of the content of the WSDL file. The WSDL file defines one operation (greet) that receives an input message (Request) and returns an output message (Response).

The import tool is used to import the document definitions for the request and response documents. You start the import of a message by double-clicking on the message node on the structure.

Double-click on the Request message.

This launches the schema import panel for the schema that describes the request message.

This panel provides you with options to override different parts of the import behavior. For the tutorial, just leave everything as it is.

Press Generate to create a Plex XML import file.

Open the Plex local model and select Tools→Import→XML Import...

Clear the input field and press Browse.

Find the generated Plex import file. This file is named pleximport.xml.

Press OK.

Plex performs the import of the definitions.

You can disregard all warning messages created by the Plex XML import. If any error messages are created, they will have to be investigated.

Refresh the Object Browser. Choose to view entities.

The definitions for the request document have been generated and placed as an unscoped entity named request.

The Request entity corresponds to the top element Request in the request document, while the two fields Request.Fields.firstName and Request.Fields.lastName correspond to the fields scoped by the Request element.

In the model, you can find the namespace information in the two source codes Namespace and prefix that is scoped by the Request entity.

To better understand the definitions, you might want to compare the definitions in the Plex model with the example request document shown in the Overview/Document definitions section of the document.

Close the Schema import panel.

Double click the response document in the tree structure.

On the Schema Import panel press Generate to generate the Plex import file.

Open the Plex local model and select Tools→Import→XML Import...

Clear the input field and press Browse.

Find the generated Plex import file. This file is named pleximport.xml.

Press OK.

Refresh the object browser.

The response document has been imported – the definitions have been placed as an unscoped entity Response.

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 call a SOAP based web service, you need a number of service functions and definitions for the XML envelope that will be wrapped around the request and response documents.

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

This creates all the definitions necessary for creating both the subscriber and the provider role. In this tutorial, only the subscriber part is created so the function handling the requests for a provider is set to implement no.

Create the triples:

Source object	Verb	Target object
Subscriber	is a ENT	WSYSOAP/HttpSoap
Subscriber.Services.SoapProcessor	implement SYS	No

Make the Subscriber entity and all the scoped objects real.

Create a WebServiceCaller function

The abstract WebServiceCaller function supports the three tasks that must be performed when calling a web service;

1. Creating the request document.
2. Making an http request to the provider
3. Reading the response document

You must implement an instance of this function. The data required to generate the request document will be delivered as input parameters and the data returned in the response document will be returned as output parameters.

Create the following triples:

Source object	Verb	Target object
TutorialWebServiceCall	is a FNC	WSYSOAP/WebServiceCaller

Source object	Verb	Target object
	input view	Request.Data
	output view	Response.Data
	local FLD	WSYBASE/ServiceURL
	local FLD	WSYSOAP/SOAPAction
	message MSG	URL
	message MSG	SOAPAction

When calling the web service, you must specify the URL of the service and the SOAP Action that specifies which operation to call (in most cases a service provides several possible operations using the same URL).

You can find the information about the URL and the SOAPAction in the WSDL file.

The URL is found as the value of the soap:address location attribute in the service section of the WSDL:

```
<wsdl:service name="TutorialService">
- <wsdl:port binding="tns:TutorialServiceSoapBinding" name="TutorialServicePort">
    <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      location="http://www.websydian.com/services/greet" />
  </wsdl:port>
</wsdl:service>
```

The SOAP Action is found as the soapAction attribute for the greet operation in the binding section of the WSDL:

```
<wsdl:binding name="TutorialServiceSoapBinding" type="tns:TutorialServicePortType">
  <soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="Greet">
    <soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      soapAction="http://www.websydian.com/services/greet" />
  + <wsdl:input>
  + <wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Specify: **http://www.websydian.com/services/greet** as the literal value of the message TutorialWebServiceCall.URL.

Specify: **http://www.websydian.com/services/greet** as the literal of the message TutorialWebServiceCall.SOAPAction.

Open the action diagram for TutorialWebServiceCall.

Add the specified code to the action diagram:

Post Point: Create request document:

```
Call Request.InsertRow
  Map with:
  Variable Input:
    RequestDocument<ObjectStoreReference>
```

```
    RequestDocument<ObjectDocument>
    RequestDocument<ObjectDocument>
  Variable Data:
    Variable Input
```

Post Point Call soap generator:

```
Format Message  Message: TutorialWebServiceCaller.SOAPAction,
Local<SOAPAction>

Format Message Message: TutorialWebServiceCaller.URL,
Local<ServiceURL>

Call Subscriber.Services.CallSoapGenerator
  Map with:
  Variable InputDocument:
    RequestDocument<ObjectDocument>
    RequestDocument<ObjectStoreReference>
  Variable OutputDocument:
    ResponseDocument<ObjectDocument>
    ResponseDocument<ObjectStoreReference>
  Variable Connection:
    Local<SOAPAction>
    <charset.utf-8>
    Local<ServiceURL>
```

Post Point Read response document:

```
Call Response.GetFirstOccurrence
  Variable Input:
    ResponseDocument<ObjectStoreReference>
    ResponseDocument<ObjectDocument>
  Variable Parent:
    <ParentElement.NULL>
If Environment<*Returned status> == <Returned status.*Successful>
    Call Response.SingleFetch
        Map with:
        Variable Input:
            ResponseDocument<ObjectStoreReference>
            Response.GetFirstOccurrence/Output<ObjectNode>
If Environment<*Returned status> == <Returned status.*Successful>
    Set Output/Output = Response.SingleFetch/Data
Else
    +For Each Field Output/Output
    ++Set Empty
```

Create a test function

After performing the steps described above, you have:

- Defined the structure of both documents.
- Defined a soap entity that provides you with the functionality for calling the web service.
- Made a function that creates the request document, calls the service and reads the response.

In a normal situation, you would call this function somewhere in your own application – for the tutorial, you will create a small panel function to perform the call.

As the fields defined in the XML documents have no length, it is a good idea to define lengths for the fields to enforce a normal behavior on the panel.

Specify the following triples:

Source object	Verb	Target object
Request.Fields.firstName	length SYS	25
Request.Fields.lastName	length SYS	25
Response.Fields.greeting	length SYS	75

To define the test function, specify the following triples:

Source object	Verb	Target object
TestSubscriber	is a FNC	UIBASIC/UIBasicShell
TestSubscriber.Panel	displays view VW ...for	Request.Data Request
	displays view VW ...for	Response.Data Response
	Region VAR	Request
	Region VAR	Response

Using the panel editor, add a button called Test on the panel and assign an event called Test to the “Pressed” physical event of the button.

Specify the following code in the action diagram of the TestSubscriber function:

Post Point Events:

```
Event  Event: Test
     Get Request
     Call TutorialWebServiceCaller
          Map with:
              Variable Input:
              Variable Request
     Set Response = TutorialWebServiceCaller/Output
     Put Response
```

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

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

wininet.lib, ws2_32.lib.

These two lib files are delivered with Visual Studio – you should not 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

WSYBASE/DWA_Win_ObjectsToGenerateAndBuild

WSYSOAP/SOAPObjectsToGenerateAndBuild

Application

Generate and build:

All objects scoped by the Subscriber entity

All objects scoped by the Request entity.

All objects scoped by the Response entity.

The function TutorialWebServiceCall.

The TestSubscriber function.

Create an exe for the TestSubscriber function

Enter the following section in the TestSubscriber.ini file:

[TransacXML]
TEMPORARY_FILES=c:\temp\SoapGenerator\

DELETE_TEMPORARY_FILES=N

Create the temporary folder c:\temp\SoapGenerator – or specify an existing folder instead.

Copy the file WsydXml11.dll to your release library.

This file is found in the Deployment\Windows folder of your Websydian installation (use the one for your Plex version).

Specifying DELETE_TEMPORARY_FILES=N means that the temporary files will not be deleted. This means you can see the content of the temporary files and use it to search for errors.

Test the function

Execute the TestSubscriber.exe function.

Enter your first and last name – now the greeting field should be shown.

Find the folder you specified for the TEMPORARY_FILES setting in the ini file. This should now contain two files – one containing the request document and one containing the response document.

Problem solving

Check folder for temporary file

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

Check whether a Request document has been created

The Request documents are named SoapOutnnnn.xml

If no such document has been created, the request has not been sent to the service – debug the program to find out where the error occurs.

Check whether a Response document has been created

Check whether a Response document has been created

The Response documents are named Responsennnn.xml.

If the folder only contains a request document, you have created the request – but not received the reply.

One obvious reason for this could be that the service is unavailable.

You can check whether the service is available 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, and an XML-document stating that the request is invalid should be shown – as shown here:

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
- <SOAP-ENV:Body>
  - <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>No SOAP envelope found in request</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If this message is not shown and you get an error in the browser, the service is not available. Please inform **mailto:support@websydian.com** about this.

Check content of the Response document

Open the response document. Check whether the response document is as you expected.

An error document will be returned if the Request document contains invalid XML, if the format of the Request document is not correct, or if the SOAPAction header is missing or incorrect (or other errors occur).

If this happens, you must investigate the content of the Request document to find out if it is correct – and check the SOAPAction (remember, this is case sensitive – try copying it from the WSDL).