Axis C++ Windows User Guide

<!-- --> <!-- -->

1. Axis C++ Windows User Guide

1.1. Creating And Deploying your own Web Service

Before you follow this guide, please make sure that you have followed the Windows Installation guide

Note: The Expat XML Parser module is not currently maintained and also contains some bugs. So it is removed from the 1.5 release.

Definitions:
Axis_Extract -> The folder to which the Axis c++ binary distribution is extracted
[Axis_Folder] -> The deploy folder of the binary distribution which is copied to the apache installation

1.2. Creating the web service

Currently axis supports two methods to create and deploy a Web Service.
Method 1) A top down approach where you start with a WSDL.
Method 2) A bottom up approach where you start with a pre-written web service.
Here we discuss the first approach since the tool to support Method 2 (i.e wcg.exe) is in a

primitive and frozen state.
Here the document is written with the idea that the user uses Visual C++ (VC). But the user could use this guide with a different IDE of his choice.

Method 1

This method assumes that the user has written the wsdl of the service which he needs to deploy. In this method user will start with this wsdl and the tool will generate the web service skeleton and other required files.

1) There is a folder called "simple" inside the samples/server folder in your axiscpp binary distribution. Inside this you can find the relevant wsdl for the calculator sample. Get the wsdl (eg:Calculator.wsdl)
2) Run the WSDL2WS tool (refer the section below 'to use the WSDL2WS tool on the command line') and generate the server side skeletons and wrappers. These files will be in two new folders which are generated from the tool called 'ServerOut' and 'ClientOut'.
3) Create a VC workspace.
4) Create a 'Win32 Static Library' project in this workspace.
5) From the generated 'ServerOut'folder, add the following files to this project.
Calculator.cpp Calculator.h
6) Set the include path to the include directory of the binary distribution (These include files are in Axis_Extract/include/).
7) Fill the empty methods of the generated skeletons.
8) Generate the lib (eg: MyCalculator.lib)
9) Now create a 'Win32 Dynamic-Link Library' project.
10) From the generated 'ServerOut'folder,add the following files to this project.
CalculatorService.cpp, CalculatorWrapper.cpp and CalculatorWrapper.h
11) Set the include path to the include directory of the binary distribution.
12) Add the above created lib (Calculator.lib) as the input library of this project.
13) Build and create the DLL. (Calculator.dll)

1.3. How to use the WSDL2WS tool on the command line

To use WSDL2Ws java tool on the command line you require jdk1.4 or above.
To use WSDL2Ws java tool you have to set the CLASSPATH Environment Variable to point to the following latest jar files.
Note: The latest jar files are in http://apache.towardex.com/ws/axis/1_2beta/
axis.jar
commons-discovery.jar
commons-logging.jar
jaxrpc.jar
saaj.jar
wsdl4j.jar

xml-apis.jar
The CLASSPATH Environment Variable should have the absolute paths of the jars (including the jar file name) given as a semicolon separated list.
Open a command window. Change directory to Axis_Extract\lib\axis. Create a folder of your choice and we will call this folder as [Wsdl2ws_Folder].
Now copy the wsdl file (eg.Calculator.wsdl) which you use, to the folder [Wsdl2ws_Folder].
Copy the file wsdl2ws.jar from Axis_Extract\lib\axis to [Wsdl2ws_Folder]
Then change the directory to [Wsdl2ws_Folder] and run the following command to generate the server side skeletons and wrappers.
java -classpath .\wsdl2ws.jar;.;%CLASSPATH% org.apache.axis.wsdl.wsdl2ws.WSDL2Ws Calculator.wsdl -o./ServerOut -lc++ -sserver
If the file generation is successful the tool will display the files that it has generated. The skeletons and wrappers will be generated in [Wsdl2ws_Folder]\ServerOut.
Run the following command to generate the client stubs.
java -classpath .\wsdl2ws.jar;.;%CLASSPATH% org.apache.axis.wsdl.wsdl2ws.WSDL2Ws Calculator.wsdl -o./ClientOut -lc++ -sclient
The generated client stubs will be in [Wsdl2ws_Folder]\ClientOut
Note: More details on WSDL2Ws Tool can be found by clicking on the following link WSDL2Ws Tool

1.4. Deploying your web service

Axis cpp user can use the AdminClient tool to deploy a service or can manually deploy. The first section shows you how to deploy your Web Service manually, without using the AdminClient tool.
Lets say that the apache installation folder is [Apache_Folder].
(The default installation is apache 1.3.X and the path is "C:\Program Files\Apache Group\Apache" and the path for apache 2.X is "C:\Program Files\Apache Group\Apache2")
1) Copy the above Calculator.dll to the folder [Apache_Folder]/Axis/webservices.
2) Add the following to the server.wsdd at the service level. Please make sure you add these lines at the correct place, i.e at service level. ([Apache_Folder]/Axis/conf/server.wsdd)

```
<service name="Calculator" provider="CPP:RPC" description="Calculator Web Service">
<parameter name="className" value="[Apache_Folder]\Axis\webservices\Calculator.dll"/>
<parameter name="allowedMethods" value="add subtract "/>
</service>
```

Now you have deployed your web service

1.5. Deploying your web service Using AdminClient Tool

The wsdl2ws Tool generates the deploy.wsdd and the undeploy.wsdd files which are needed for the AdminClient. Once we have these files, we have to deploy the web service (in this case the calculator service) with the AdminClient. We do this with the AdminClient.exe which comes with axiscpp binary distribution. A typical invocation of the AdminClient looks like this.

AdminClient <server> <Port> <wsddfile>

AdminClient localhost 80 deploy.wsdd

where local host would be the server where the Axis cpp server is hosted and 80 would be the port at which it runs.

1.6. Coding the client

With the WSDL2WS tool you have almost developed your client. What you have to do next is write a file which has a main method and create an object of the stub and invoke your methods on that.
1) Create a vc workspace.
2) Create a 'Win32 Console Application'.
3) Add files to this project from the above generated 'ClientOut' folder.
4) Set the include path to the include directory of the binary distribution.
5) Add the following libs to the library modules path of this project.
Axis_Extract/lib/axis/
Axisclient.lib
6) Create a file with a main method which looks similar to the following and add this file to this project.

```
#include "Calculator.h" int main() { Calculator c; int result = c.add(40, 20); printf("result = %d", result); return 0; }
```

7) Now build and create the Client.exe

1.7. Running your sample

1) Restart Apache.
2) Run the Calculator.exe

SUCCESS ! If you get the result, you are done.

1.8. Transport Library and Parser Library

HTTPTransport.dll and HTTPChannel.dll (Which can be found at Axis_Extract/bin) should be placed in the path, and should be specified as the value to the keys "Transport_http" and "Channel_HTTP" respectively in axiscpp.conf [Axis_Folder]/axiscpp.conf Or in the same place as the client.exe.

Rename AxisXMLParser_Xerces.dll to AxisXMLParser.dll and give the path of the AxisXMLParser.dll as the value of the key XMLParser in axiscpp.conf Or in the same place as the client.exe.
xerces-c_2_2_0.dll should be given in the path.

Axiscpp.conf file contains the following paths
LogPath:XXXX
WSDDFilePath:YYYY
Transport_http:ZZZZ
Channel_HTTP:BBBB
XMLParser:WWWW
XXXX is the path to a file named AxisLog (The log file)and YYYY is the path to the server.wsdd file.
i.e.
LogPath:[Apache_Folder]\Axis\logs\AxisLog.log
WSDDFilePath:[Apache_Folder]\Axis\conf\server.wsdd
Transport_http:[Apache_Folder]\Axis\libs\HTTPTransport.dll
Channel_HTTP:[Apache_Folder]\Axis\libs\HTTPChannel.dll
XMLParser:[Apache_Folder]\Axis\libs\AxisXMLParser.dll

1.9. Handlers

Handlers are pluggable components in Axis C++. We have included a set of sample handlers for your reference. You could write your own handlers by following the instructions given for the sample Handlers.

Note: If you are using Client side Handlers you need to enter the following entry to the [Axis_Folder]/axiscpp.conf configuration file.

ClientWSDDFilePath:Axis\conf\client.wsdd

After entering this entry to your [Axis_Folder]/axiscpp.conf configuration file will look like:

LogPath:Axis\logs\AxisLog.txt

WSDDFilePath:Axis\conf\server.wsdd
ClientWSDDFilePath:Axis\conf\client.wsdd

Testing the sample Handlers

We have included the following sample Handlers for your reference.

1) echoStringHeaderHandler (A server side handler sample) This sample handler will simply echo (i.e send back) the string which you send in the SOAP request.
2)testHandler (A client side handler sample)

This sample handler will simply add a SOAP Header to the generated SOAP request.

Please note that these are very primitive sample handlers and are presented here to give you an idea about writing your own Handlers.

echoStringHeaderHandler

Building the Sample Handlers in VC

Building echoStringHeaderHandler (A server side handler sample)

The VC dsw file (ServerHandlers.dsw) is available at Axis_Extract/vc/samples/server/ ServerHandlers.dsw. Open this file and build the project echoStringHeaderHandler. Once the build is successful you will find the DLL (echoStringHeaderHandler.dll) at Axis_Extract/bin. If you see this DLL at the above location you are done with the first step.

Configuring the Handler

Now edit the [Axis_Folder]/conf/server.wsdd to include the handler for a particular service.

```
<service name="Calculator" provider="CPP:RPC" description="Simple Calculator Axis C++ Service ">
<requestFlow name="CalculatorHandlers">
<handler name="ESHHandler" type="Axis_Extract/bin/echoStringHeaderHandler.dll">
</handler>
</requestFlow>
<responseFlow name="CalculatorHandlers">
<handler name="ESHHandler" type="Axis_Extract/bin/echoStringHeaderHandler.dll">
</handler>
</responseFlow>
<parameter name="allowedMethods" value="add sub mul div "/>
<parameter name="className" value="Axis\webservices\Calculator.dll" />
</service>
```

Note: Make sure you specify the correct path of the handler dll in the server.wsdd file.

Now you are almost done to run your server side handler.
Restart the Apache server.

Running the Handler

Since this Handler is configured to the Calculator web service in the above step, this Handler will be executed when a client send a SOAP request to the Calculator web service.

testHandler

Building the Sample Handlers in VC

Building testHandler (A client side handler sample)

The VC dsw file (ServerHandlers.dsw) is available at Axis_Extract/vc/samples/client/ClientHandlers.dsw. Open this file and build the project TestHandler. Once the build is successful you will find the DLL (testHandler.dll) at at Axis_Extract/bin. If you see this DLL at the above location you are done with the first step.

Configuring the Handler

Now edit the [Axis_Folder]/conf/client.wsdd to include the handler for a particular service.

<service name="Calculator" provider="CPP:DOCUMENT" description="Calculator web service">
<requestFlow name="CalculatorHandlers">
<handler name="TestHandler" type="Axis_Extract/bin/testHandler.dll">
</handler>
</requestFlow>
</service>

Note: Make sure you specify the correct path of the handler dll in the client.wsdd file.

Now you are almost done to run your client side handler.

Note: If you are using Client side Handlers you need to enter the ClientWSDDFilePath entry in the [Axis_Folder]/axiscpp.conf configuration file. (See above)

Running the Handler

Since this Handler is configured to the Calculator web service in the above step, this Handler will be executed when you run the calculator web service client. (It is at Axis_Extract/bin/Calculator.exe)

Handler Notes:

1) You can see the Handler behavior through the TCP Monitor. (TCP Monitor is a Axis Java

tool)
2) To get an idea of Handlers look at the Handler sample source files.
a. echoStringHeaderHandler (Axis_Extract/samples/server/echoStringHeaderHandler)
b. testHandler (Axis_Extract/samples/client/testHandler)

1.10. SSL Client

This section describes how to use an Axis C++ SSL secure client to access web services hosted on a secure web service.
The SSL implementation for the client in Axis C++ uses the openssl opensource library.
To aid to compile the secure channel dll
1. Install the openssl 0.9.7e binary distribution.
2. Copy the include files from the openssl installation to c\include\openssl in your distribution
3. Copy libs found in \lib\VC of the openssl installation to c\lib\openssl
Now use the vc project in c\vc\transport\Axis2\Axis2SSLChannel to compile the Secure channel dll.
Paste this dll where the Axis C++ client can load it (i.e %PATH%) OR specify the path to it in the axiscpp.conf under the key "Channel_ssl" (e.g Channel_ssl:c:\Axis2SSLChannel.dll)
Now when you run any client using a url of the form https://..... the client will use SSL to connect to the relevant secure service specified by the url. The client request must be directed at a "secure webserver" which has the relevant web service hosted.

1.11. Session Headers

The following text explains how to deploy and run the SOAP Header based sample client with Axis Java web service
Deploying the Web Service

c\samples\server\session\headers folder contains the sources (inside the counters folder, which is the package of these classes) needed to build the Axis java service needed to run the soap header based session client (These server side skeletons were generated from the Counter.wsdl)
Compile these java source files and deploy them in Axis java (visit http://ws.apache.org/axis/java/index.html on how to achieve this)
Put the following element in the section in the server-config.wsdd to enable SOAP header based session handling for Axis Java
<handler name="session" type="java:org.apache.axis.handlers.SimpleSessionHandler"/>

The following should be put in the server-config.wsdd of Axis java for this service to behave as having session scope
<service name="CounterService" provider="java:RPC">
<parameter name="scope" value="session"/>
<requestFlow>
<handler type="session"/>
</requestFlow>
<responseFlow>
<handler type="session"/>
</responseFlow>
<parameter name="allowedMethods" value="*"/>
<parameter name="className" value="counters.CounterSoapBindingImpl"/>
<namespace>http://xml.apache.org/axis/wsdd/</namespace>
</service>

Since Axis c++ doesn't support multiref yet, Axis java multiref should be disabled by putting the element
<parameter name="sendMultiRefs" value="false"/>
under <globalConfiguration>
Start Axis java (visit http://ws.apache.org/axis/java/index.html on how to achieve this)
Generating the client stubs and building the client and running the client.

Use the vc workspace \c\vc\samples\client\session\Headers\Headers.dsw to compile the client side handler for this sample
Run the command java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws ../Counter.wsdl -o./gen_src -lc++ -sclient from within c\samples\client\session\headers\sessionclient to generate the client stubs
Compile the client application using the vc workspace at
\c\vc\samples\client\session\Headers\Headers.dsw
Host the service in Axis java (Check c/samples/server/session/headers/readme.txt on how to do this).
Configure the client to use the provided client.wsdd from axiscpp.conf (make appropriate changes if necessary in the client.wsdd to the absolute path of the handler)
Run the tcpMonitor and configure it to check the conversation between the client and server
Run the client in the following fashion
sessionClient count 1 http://localhost:8080/axis/services/CounterService
Inspect the SOAP messages in tcpMonitor to see the values returned by the server incremented by 1 each time (as done throught the client). Counting starts at the value 97, which is set at the server side web service.

1.12. IPV6

The source is in src/transport/axis2/ipv6/

We need the 2 additional headers that comes with the IPv6Kit.
Can be downloaded from http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp
Extract the package and copy the tpipv6.h and wspiapi.h headers in inc folder to $AXISCPP_HOME/include.
That would compile the axis2ipv6 VC++ project.