End-to-End Calculator Sample Tutorial

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

1. Contents

1.1. Introduction

The purpose of this page is to give the user a step-by-step guide to create a simple web service client/server example using Apache Axis.
It is assumed that the user has downloaded the latest version of the Axis code (from page http://ws.apache.org/axis/cpp/download.html) and they also have the prerequisites as described in the Pre-requisite section (from page http://ws.apache.org/axis/cpp/install-guide.html) and has set-up any environment variables required by Axis or its prerequisite applications.

1.2. Getting Started

Before beginning, check that there is the following directory structure and that you have all of the libraries on the 'path' (especially all the prerequisites). Below is a typical directory structure that you should expect to find if you have just downloaded and unzipped the package.

```
+- Axis
| +- samples
| | +- client
| | | +- calculator
| | | | +- Client.cpp
: : : :
| | +- server
| | | +- calculator
```

| | | +- Calculator.cpp
: :
| +- deploy
| | +- wsdls
| | | +- calculator.wsdl
: :
| +- lib
| | +- axis
| | | +- wsdl2ws.jar
| | | +- axis_client.lib or libaxis_client.so
| | | +- axis_server.lib or libaxis_server.so : :

Fig 1. Portion of a typical Axis directory structure containing the required sample client/server components.

Note: The user must also have the correct level of Java on the build/test machine.

1.3. Creating the server side code

The first stage in creating the server side code is to create the server side stubs for the web service. This can be done very simply by using the Java application wsdl2ws.jar. This application will create both client and server side 'stubs' from which the user can create their web service application. (These 'stubs' remove the need for the user to have a deep understanding of how web services work, but the user should have some appreciation and at least a working knowledge of web service basics.) The key to web services is the WSDL file. This file contains a complete description of what methods are available in the service as well as the parameters names and types required by each method. In short, it provides a description of the interface. For this example, we will be using the calculator wsdl. In the following command line example, <*inst_dir*> is assumed to be the directory to which Axis was unzipped and <*samp_dir*> is assumed to be the directory in which the server side calculator sample is to be created. To create the server side stubs for the calculator sample, type the following at the command line:-

Windows/Unix
java –classpath <*inst_dir*>/lib/axis/wsdl2ws.jar;<*inst_dir*>/lib/axisjava/axis.jar;<*inst_dir*>/lib/axisjava/commons-discovery.jar;<*inst_dir*>/ org.apache.axis.wsdl.wsdl2ws.WSDL2Ws <*inst_dir*>/deploy/wsdls/calculator.wsdl -sserver -o<*samp_dir*>/Server

Fig 2. Command line to create the server side stubs from the calculator.wsdl

After running wsdl2ws, there should now be a new directory under the <*samp_dir*> called 'Server' which should contain 9 files as follows:-

AxisServiceException.cpp AxisServiceException.hpp Calculator.cpp Calculator.hpp CalculatorService.cpp CalculatorWrapper.cpp CalculatorWrapper.hpp deploy.wsdd undeploy.wsdd

Fig 3. Files produced by running the wsdl2ws application with the calculator wsdl file.

Note: wsdl2ws will only generate files if they do not already exist. If the wsdl changes, the user must remember to delete the 'old' stubs before using wsdl2ws to generate the updated ones.

1.4. Building the server side code

Before building the server side you might want to familiarise yourself with the content of the generated files. In particular, the Calculator.cpp file which will contain the server side of the web service methods. If we concentrate on the 'add' method, the raw code produced by the stub is as follows:-

```
xsd__int Calculator::add(xsd__int Value0,xsd__int Value1) { }
```

Fig 4. Raw server side Calculator::add method code produced by the wsdl2ws application.

At present, the code will not compile because there is no 'return' value. Also, there is no method code content. Before the code has any use, the programmer will have to add the necessary program logic that the method is supposed to provide. In this case, it is to add the two numbers in the parameter list and return the result.
You can either add the content or use the already prepared Calculator.cpp file supplied in the <*inst_dir*>/samples/server/calculator directory.

```
xsd__int Calculator::add( xsd__int Value0, xsd__int Value1) { return Value0 + Value1; }
```

Fig 5. Server side Calculator::add method from the samples/server/calculator directory.

Once the logic within the Calculator.cpp file has been completed, the server side library can be built using the appropriate c/c++ compiler. Below are the compiler commands for each platform:-

Windows	Unix
cl *.cpp /GX /MD /D "WIN32" /I<*inst_dir*>/include /link<*inst_dir*>/lib/axis/axis_server.lib /DLL /OUT:"Calculator.dll"	gcc *.cpp -I <*inst_dir*>/include -L<*inst_dir*>/lib/axis/axis_server.so -o Calculator.so

Fig 6. Command line to create the server side calculator library

After compilation, the appropriate library file should have been created.

1.5. Deploying the server side code

The SimpleAxisServer application uses a wsdd file to identify the connection between a web service name and the server library. The axiscpp.conf file contains a tag to identify the location of the wsdd file to be used by the server. The axiscpp.conf file can be found in the directory identified by the AXISCPP_DEPLOY environment variable. Below is the line containing the tag that identifies where the server wsdd file is located.
WSDDFilePath: <*samp_dir*>/Server/deploy.wsdd

Fig 7. Server WSDD tag in the axiscpp.conf used by SimpleAxisServer to locate services to be deployed.

Note: If more than one service is to be deployed then the user will have to concatenate the deploy.wsdd files generated by the wsdl2ws tool.

The contents of the automatically generated deploy.wsdd file should be as follows:-

```
<?xml version="1.0" encoding="UTF-8"?> <deployment xmlns=http://xml.apache.org/axis/wsdd/ xmlns:CPP=http://xml.apache.org/axis/wsdd/providers/CPP> <service name="Calculator" provider="CPP:RPC" description="Axis C++ web service"> <parameter name="className" value="/user/local/apache/axis/Calculator.dll"/> <parameter name="allowedMethods" value="add addRequest sub subRequest mul mulRequest div divRequest "/> </service> </deployment>
```

Fig 8. Contents of the deploy.wsdd file created by the wsdl2ws tool.

Before deployment, the wsdd file needs one alteration for the location of the library. Currently it is set to "/user/local/apache/axis/…", this needs to be edited to <*samp_dir*>\Server\Calculator.dll or <*samp_dir*>/Server/Calculator.so, depending on the operating system.
After making this alteration, the server side is now ready to run. More information on how to run the SimpleAxisServer can be found at http://ws.apache.org/axis/cpp/install-guide.html#Simple_Axis_Server_Installation_and_Configuration.

1.6. Creating the client side code

The first stage in creating the client side code is to create the client side stubs for the web service. This can be done very simply by using the Java application wsdl2ws.jar. This

application will create both client and server side ‘stubs’ from which the user can create their web service application. (These ‘stubs’ remove the need for the user to have a deep understanding of how web services work, but the user should have some appreciation and at least a working knowledge of web service basics.) The key to web services is the WSDL file. This file contains a complete description of what methods are available in the service as well as the parameters names and types required by each method. In short, it provides a description of the interface. For this example, we will be using the calculator wsdl. In the following command line example, <*inst_dir*> is assumed to be the directory to which Axis was unzipped and <*samp_dir*> is assumed to be the directory in which the client side calculator sample is to be created. To create the client side stubs for the calculator sample, type the following at the command line:-

Windows/Unix
java –classpath <*inst_dir*>/lib/axis/wsdl2ws.jar; <*inst_dir*>/lib/axisjava/axis.jar;<*inst_dir*>/lib/axisjava/commons-discovery.jar;<*inst_dir*>/lib/axisjava/commons-logging.j org.apache.axis.wsdl.wsdl2ws.WSDL2Ws <*inst_dir*>/deploy/wsdls/calculator.wsdl –sclient -o<*samp_dir*>/Client

Fig 9. Command line to create the client side stubs from the calculator.wsdl

After running wsdl2ws, there should now be a new directory under the <*samp_dir*> called ‘Client’ which should contain 2 files as follows:-

Calculator.cpp Calculator.hpp

Fig 10. Files produced by running the wsdl2ws application with the calculator wsdl file.

1.7. Building the client side code

Before building the client side you might want to familiarise yourself with the content of the generated files. In particular, the Calculator.cpp file which will contain the client side of the web service methods.

At present, there is no ‘client’ code, just the stub code. Before the stub code can be compiled, the programmer will have to provide a client program that will use one or more of the methods provided by the web service. In the following example, a client has been written that requires the function provided by the ‘add’ method in the calculator web service.

The client application writter can either provide their own client cpp file or use the already prepared Client.cpp file supplied in the <*inst_dir*>/samples/client/calculator directory, an extract from which follows:-

try { Calculator ws(“http://localhost:9080/axis/Calculator”); xsd__int iValue1 = 25; xsd__int iValue2 = 25; xsd__int iResult = ws.add(iValue1, iValue2); xsd__int iExpected = iValue1 + iValue2; cout << “Web Service: ” << iValue1 << “ + “ << iValue2 << “ = “ << iResult << endl; cout << “Local Machine:” << iValue1 << “ + “ << iValue2 << “ = “ << iExpected << endl; } catch(AxisException & e) { cout << "Axis

```
Exception : " << e.what() << endl; } catch( exception & e) { cout << "Exception : " << e.what() << endl; } catch( ...) { cout << "Unknown exception " << endl; }
```

Fig 11. Extract from the client side application that uses the 'add' method from the samples/server/calculator web service.

Once the logic within the Client.cpp file has been completed, the client side executable can be built using the appropriate c/c++ compiler. Below are the compiler commands for each platform:-

Windows	Unix
cl *.cpp /GX /MD /D "WIN32" /I<*inst_dir*>/include /link<*inst_dir*>/lib/axis/axis_client.lib /OUT:"Calculator.exe"	gcc *.cpp -I <*inst_dir*>/include -L<*inst_dir*>/lib/axis/libaxis_client.so -o Calculator

Fig 12. Command line to create the client side calculator executable

After compilation, the appropriate executable file should have been created.

1.8. Running the client side code

With the SimpleAxisServer application already running and with the correct wsdd file deployed, the Calculator client application can now be run from the command line as follows:-

Windows	Unix
Calculator add 25 25 http://localhost:9080/axis/Calculator	./Calculator add 25 25 http://localhost:9080/axis/Calculator

Fig 13. Command line to run the client side calculator executable.

More information on how to run the Client application can be found at http://ws.apache.org/axis/cpp/install-guide.html#Installing_Client.