Axis C++ Linux User Guide

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

1. Axis C++ Linux User Guide

Contents

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

Introduction

Welcome to Axis C++, the opensource c++ implementation of SOAP !

What is SOAP?

SOAP is an XML-based communication protocol and encoding format for inter-application communication. Originally conceived by Microsoft and Userland software, it has evolved

through several generations and the current spec, SOAP 1.2 is fast growing in popularity and usage. The W3C's XML Protocol working group is in the process of turning SOAP into a true open standard, and as of this writing has released a working draft of SOAP 1.2, which cleans up some of the more confusing areas of the 1.1 spec. SOAP is widely viewed as the backbone to a new generation of cross-platform cross-language distributed computing applications, termed Web Services. What is Axis C++? Axis C++ is essentially a SOAP engine.

This version is written in C++. Axis C++ SOAP engine adopts most of Axis Java architecture. But it has some major architectural innovations over Axis Java in order to achieve greater performance and efficiency.

What's in this release?

- Soap engine with both client and server support

- Support for both SOAP 1.1 and SOAP 1.2

- WSDD based deployment with dynamic deployment tools.

- Support for all basic types, Complex types and Arrays

- WSDL2WS tool for building C/C++ components

- Server side – Skeletons and Wrappers

- Client side – Stubs

- WSDL2WS tool that generates wrappers, which perform the following functions. These wrappers act as RPC
  Providers.

- Serialization

- Deserialization

- Method invocation

- WSDLs hosted statistically in the server.

- Standalone server (with HTTP support)

- Web server modules for Apache 1.3 & Apache2 (Linux/Windows)

- Basic Wrapper Class Generator tool.

- Web interface to the deployed services and their WSDL s.

- Sample web services and client applications.

- Document style web services support

Axis C++ now delivers the following key features

- Speed: Axis uses SAX (event-based) parsing to acheive significantly greater speed

- Flexibility

- Stability , Component oriented Deployment

- Transport Framework

- WSDL support

Axis C++ 1.1 supports the Web Service Description Language, version 1.1, which allows you to easily build stubs to access remote services, and also to automatically export machine-readable descriptions of your deployed services from Axis. We hope you enjoy using Axis c++ 1.1. Please note that this is an open-source effort - if you feel the code could use some new features or fixes, please get involved and lend a hand! The Axis developer community welcomes your participation. Let us know what you think! Please send feedback about the package to axis-user@xml.apache.org

Installing Axis and Using this Guide

See the Axis Installation Guide for instructions on installing Axis C++

Before running the examples in this guide, you'll need to make sure that your environment variables and other configurations are set correctly as described in Installation guide. In addition you need

- j2SDK1.4

installed and configured.

Let's take a look at a sample Calculator service client that will call methods of a Calculator service deployed on Axis C++.

When starting with the valid WSDL file to use Axis C++ you have to get started with the tool called WSDL2Ws which is written in Java. source for WSDL2Ws tool is in

$AXISCPP_HOME/src/wsdl

You need the following latest jar files which are in

http://apache.towardex.com/ws/axis/1_2beta/ please include them in the **CLASSPATH** .

- 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 colon separated list

Here is a sample **/home/axisuser/.bash_profile** file where we specified those

AXIS_JARS_HOME="$AXISCPP_HOME/lib/axisjava"

AXIS_JARS="$AXIS_JARS_HOME/axis-ant.jar:$AXIS_JARS_HOME/axis.jar:$AXIS_JARS_HOME/commons-discovery.jar:$AXIS_JARS_HOME/commons-logging.jar:$AXIS_JARS_HOME/jaxrpc.jar:$AXIS_JARS_HOME/log4j-1.2.4.jar:$AXIS_JARS_HOME/saaj.jar:$AXIS_JARS_HOME/wsdl4j.jar"

JAVA_HOME="/usr/java"

PATH="$PATH:$JAVA_HOME/bin:."

CLASSPATH="$CLASSPATH:./:$JAVA_HOME/lib:$AXIS_JARS:"

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC AXIS_JARS_HOME

AXIS_JARS JAVA_HOME CLASSPATH

Now

There are two options to create the wsdl2ws.jar tool
1st Option is using ANT (This is more reliable)
In your PATH environment variable add path to <antinstall dir>/bin.
$ cd $AXISCPP_HOME/src/wsdl/
$ ant

2nd Option (Some users have experienced difficulties in using this option)

$ cd $AXISCPP_HOME/src/wsdl/

$ mkdir temp

$javac -d ./temp -sourcepath . ./org/apache/axis/wsdl/wsdl2ws/*.java

$cd temp

$jar -cvf wsdl2ws.jar org

$cp -f wsdl2ws.jar $AXISCPP_HOME/lib/axis

add this jar as the first entry into the classpath as well.(In the binary distribution you don't need to create this jar. It is already in $AXISCPP_HOME/lib/axis)

Server side Skeleton And Wrappers Generated by the wsdl2ws Tool

We use the sample at

$AXISCPP_HOME/samples/server/simple

We use this sample to demonstrate the generation of serverside skeletons and how to deploy a web service using it.

Inside this folder you will find Calculator.wsdl file using which we generate skeleton and Wrappers. Here is the command line arguments to generate the skeleton.

***important:In this sample we generate the skeltons using Calculator.wsdl and wsdl2ws tool. But in the folder you will find already generated files. If you wish to use those without generating new ones you can do so. We recommend that you deploy the sample with the already generated files in the first round and later do the same with code generated from Calcuator.wsdl.**

cd $AXISCPP_HOME/samples/server/simple

% java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws Calculator.wsdl -lc++ -sserver

Note: If you give **-o. /GenClassesServer** then the server create a folder named GenClassServer and put the source there. Otherwise the source is put in the current folder where the tool is run.

Building and Deploying the Service

To build the service library

g++ -shared -I$AXISCPP_HOME/include -olibmyservice.so *.cpp

libmyservice.so is the name you give to your service library. You can give any name you wish. But remeber to prefix with lib and suffix with .so

Copy this service library into $AXICPP_DEPLOY/lib

Modify the **$AXIS_HOME/conf/server.wsdd** . (You have a sample server.wsdd file entry given below appropriately filled for this service).

<?xml version="1.0" encoding="UTF-8"?>
<deployment
xmlns="http://xml.apache.org/axis/wsdd/"xmlns:C="http://xml.apache.org/axis/wsdd/providers/c">
<service>
<service name ="Calculator" provider ="CPP:RPC" description:"Simple calculator web service"> <parameter name="classname" value = "/usr/local/Axis/webservices/libcalculator.so" />

<parameter name= "allowedMethods" value="add sub mul div "/> </service>

</deployment>

Start the Apache server

$ /usr/local/apache/bin/apachectl start

Now open a browser and enter the link http://localhost/axis If the service is correctly deployed then it will be displayed in a table of deployed services which contain information such as service name, link to wsdl and a description of the service.

Client side Stubs Generated by the wsdl2ws Tool

WSDL2Ws tools will generate the stubs for the client side. You will have C++ Client class and header file.

$ cd $AXISCPP_HOME/samples/client/simple

$ cp -f $AXISCPP_HOME/samples/server/simple/Calculator.wsdl ./

***important:In this sample we generate the stubs using Calculator.wsdl and wsdl2ws tool. But in the folder you will find already generated files. If you wish to use those without generating new ones you can do so. We recommend that you run the sample with the already generated files in the first round and later do the same with code generated from Calcuator.wsdl.**

$ java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws Calculator.wsdl -lc++ -sclient

Note: again if you specify **-o./GenClassesClient** you will have source generated inside GenClassClient folder instead of current folder where the tool is run. Before compiling the client you have to write a class which contain a main method in which Calculator instance is created and its methods are called.

Then fill the samples with the relevant business logics .

Then fill the main method in a file as follows

```
#include "Calculator.h"
#include<stdio.h>
int main()
 {
  Calculator c;
  int intOut;
  c.add(20, 40, intOut);
  printf("result is = %d\n", intOut);
  return 0;
 }
```

Then build by
cd $AXISCPP_HOME/samples/client/simple

g++ *.cpp -I$AXISCPP_HOME/include -L$AXISCPP_DEPLOY/lib -ldl -laxiscpp_client -ocalculator
Then to run the calculator sample

./calculator add 10 5 http://locathost/axis/Calculator

Building Serverside of provided Samples

Basically this will include all the Interoptest Samples and calculator sample.

cd $AXISCPP_HOME/samples

$ sh autogen.sh

$ sh runconfig

$ make

$ make install

Once you type the above command all the server samples will be deployed in $AXISCPP_DEPLOY/lib folder.

you also have sample $AXISCPP_DEPLOY/conf/server.wsdd_linux file which you should rename to server.wsdd,
which contain all the neccessary entries for these sevices. Sample clients will be installed in $AXISCPP_DEPLOY/bin

Restart Apache

To run the samples
$ cd $AXISCPP_DEPLOY/bin

$ ./base http://localhost/axis/base

to run all the samples at once
sh run_interoptests.sh Note: local host and port 80 is assumed. If you have different use.

$ sh run_interoptests.sh -u http://yourserver:yourport/axis

Handlers

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

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

CLIENTWSDDFILEPATH:Axis\conf\client.wsdd

After entering this entry your AXIS_HOME/axiscpp.conf configuration file will look like:

AXISLOGPATH: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 RedHat linux

Building echoStringHeaderHandler (A server side handler sample)

The build files are available at AXISCPP_HOME/samples/server/echoStringHeaderHandler. Change your current directory to this direcotory and then you could execute the following.

sh autogen.sh

sh runconfig

make

make install

The handler so file will be created at $AXIS_HOME/handlers/custom/echoStringHeaderHandler.

Configuring the Handler

Now edit the AXIS_HOME/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_HOME/handlers/custom/echoStringHeaderHandler/libeshhandler.so">

</handler>

</requestFlow>

<responseFlow name="CalculatorHandlers">

<handler name="ESHHandler" type="AXIS_HOME/handlers/custom/echoStringHeaderHandler/libeshhandler.so">

</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 so in the server.wsdd file. Replace the AXIS_HOME with the exact relative path which AXIS_HOME points to. (eg: type="/usr/local/apache2/Axis/handlers/custom/echoStringHeaderHandler/libeshhandler.so)

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

Restart the Apache server and that is it.

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 RedHat linux

Building testHandler (A client side handler sample)

The build files are available at AXISCPP_HOME/samples/client/testHandler. Change your current directory to this direcotory and then you could execute the following.

sh autogen.sh

sh runconfig

make

make install

The handler so file will be created at $AXIS_HOME/handlers/client/test_handler.

Configuring the Handler

Now edit the AXIS_HOME/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_HOME/handlers/client/test_handler/libtest_client_handler.so">

</handler>

</requestFlow>

</service>

Note: Make sure you specify the correct path of the handler so in the client.wsdd file. Replace the AXIS_HOME with the exact relative path which AXIS_HOME points to. (eg: type="/usr/local/apache2/Axis/handlers/client/test_handler/libtest_client_handler.so)

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_HOME/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 AXISCPP_HOME/samples/client/simple/calculator)

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 (AXISCPP_HOME/samples/server/echoStringHeaderHandler)

b. testHandler (AXISCPP_HOME/samples/client/testHandler)

Getting a CVS checkout

Visit http://ws.apache.org/ click on "axis" and then on "CVS Repository" to find details on accessing the CVS Repository. It will have instructions similar to the following."Anyone can checkout source code from our anonymous CVS server. To do so, simply use the following commands (if you are using a GUI CVS client, configure it appropriately):

cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login

password: anoncvs

cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic checkout -d <your local folder> ws-axis/c"

SSL Client

To build the ssl channel library configure with
configure --with-axis2-ssl=PATH
Add the following entry to the axiscpp.conf
Channel_ssl:/usr/local/axiscpp_deploy/lib/libaxis2_ssl_channel.so
Note: If you don't add the above entry, lib will be taken from LD_LIBRARY_PATH

Then send your request with https://...
Axis2Transport loads the ssl channel library when it is https and sends your request through ssh tunnelling.
Currently I use openssl libraries for ssh tunnelling.
The API to write a new ssl channel library(using a library other than openssl) is in **src/transport/SSLChannel.hpp**
All openssl ssl related implementations are in **src/transport/axis2/ssl folder**

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.

Compile the sessionhandler using the sources in c\samples\client\session\headers\sessionhandler
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 following command from within c\samples\client\session\headers\sessionclient
g++ CounterClient.cpp gen_src/*.cpp -Igen_src
I$AXISCPP_HOME/include -L$AXISCPP_DEPLOY/lib -ldl
laxiscpp_client -oclient
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.

PDF

PDF