Handler Tutorial

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

1. Handler Tutorial

For Linux
For Windows

2. Linux

Introduction to Handlers
Sample Handlers (building, running)
Creating your own Handlers
API Notes for Handler writers

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:xxx
Here xxx should be the path to the “client.wsdd” file.

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 Header 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 directory and then you could execute the following.
make
make install
The handler so file will be created at $AXISCPP_DEPLOY/lib directory

Configuring the Handler

Now edit the AXISCPP_DEPLOY/etc/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="AXISCPP_DEPLOY/lib/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 AXISCPP_DEPLOY with the exact relative path which AXISCPP_DEPLOY points to. (eg: type="/usr/local/axiscpp_deploy/etc/libeshhandler.so)
Now you are almost done to run your server side handler.
Restart the Apache server and that's 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.
make
make install

The handler so file will be created at $AXISCPP_DEPLOY/lib/.

Configuring the Handler

Now edit the AXISCPP_DEPLOY/etc/client.wsdd to include the handler for a particular service.

```
<service name="Calculator" provider="CPP:RPC" description="Calculator web service">
<requestFlow name="CalculatorHandlers">
<handler name="TestHandler" type="AXISCPP_DEPLOY/lib/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 AXISCPP_DEPLOY with the exact relative path which AXISCPP_DEPLOY points to. (eg: type="/usr/local/axiscpp_deploy/lib/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 entry in the AXISCPP_DEPLOY/etc/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_DEPLOY/bin/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)

The Handler API and details for Handler writers

Now you have seen some sample Handlers so that you can explore more on Handlers. The following sections helps you for the same.

In order to get access to the DeSerializer the handler writer can use the following code block.

```
// -----
.....
IHandlerSoapDeSerializer* pIHandlerSoapDeSerializer;
pIMsg->getSoapDeSerializer(&pIHandlerSoapDeSerializer);
.....
-------//
```

In order to get access to a incoming HeaderBlock the handler writer can use the following code block.

```
// -----
.....
IHeaderBlock*                                                    pIHeaderBlock=
pIHandlerSoapDeSerializer->getHeaderBlock("echoMeString",
"http://soapinterop.org/echoheader/");
.....
-------//
```

In order to manipulate the above accessed HeaderBlock the handler writer can use the following code block.

```
// -----
.....
if (pIHeaderBlock != NULL) {
  const BasicNode* pBasicNode= pIHeaderBlock->getFirstChild();
  const AxisChar* pachHeaderValue;
  if (pBasicNode != NULL)
  {
    if((pBasicNode->getNodeType()) == CHARACTER_NODE) {
    pachHeaderValue= pBasicNode->getValue();
  } else {
    pachHeaderValue = "This was not the expected value Ros";
  }
  } else
  {
   pachHeaderValue = "pBascNode is NULL";
   }
  AxisChar* pachTmpValue = (AxisChar*) malloc(strlen(pachHeaderValue) + 4);
  strcpy(pachTmpValue, pachHeaderValue);
  pachTemp = "EchoStringHeaderHandlerPr1.id";
  pIMsg->setProperty(pachTemp, pachTmpValue);
  free(pachTmpValue);
  } else {
  //do some thing
  //AxisChar* pachTmpValue = "Default values since no reqeust SOAP header";
  //pachTemp = "EchoStringHeaderHandlerPr1.id";
   //pIMsg->setProperty(pachTemp, pachTmpValue);
  //free(pachTmpValue);
  }
```

.....
-------//
In order to get access to the incoming SOAP Body the handler writer can use the following code block.
To get the body as a AxisChar*
// -----
.....
IHandlerSoapDeSerializer* pIHandlerSoapDeSerializer;
pIMsg->getSoapDeSerializer(&pIHandlerSoapDeSerializer);
AxisChar* pSoapBody = pIHandlerSoapDeSerializer->getBodyAsChar();
.....
-------//
To get the body as a decoded base64 stream.
// -----
.....
IHandlerSoapDeSerializer* pIHandlerSoapDeSerializer;
pIMsg->getSoapDeSerializer(&pIHandlerSoapDeSerializer);
xsd__base64Binary bb = pIHandlerSoapDeSerializer-&gtgetBodyAsBase64Binary();
.....
-------//

Notes:
Have a look at the following classes at the API docs to see all the available functions and their respective descriptions. (You can even look at the relevent .h/.hpp header files for the API comments)
IhandlerSoapDeSerializer
IHeaderBlock
BasicNode

The BasicNode API is similar (not exactly the same, but ..) to the DOM and is written as a tree traversing API.
With the sample code samples provided above and with the API notes a developer will easily be able to write and play around his/her own Handlers.
This tutorial will be updated frequently with the new additions and specially with your suggestions.

3. Windows

Introduction to Handlers
Sample Handlers (building, running)

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 Header 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 **(ClientHandlers.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 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)

The Handler API and details for Handler writers
Now you have seen some sample Handlers so that you can explore more on Handlers. The following sections helps you for the same.
In order to get access to the DeSerializer the handler writer can use the following code block.

```
// -----
.....
IHandlerSoapDeSerializer* pIHandlerSoapDeSerializer;
pIMsg->getSoapDeSerializer(&pIHandlerSoapDeSerializer);
.....
-------//
```

In order to get access to a incoming HeaderBlock the handler writer can use the following code block.

```
// -----
.....
IHeaderBlock* pIHeaderBlock=
pIHandlerSoapDeSerializer->getHeaderBlock("echoMeString",
"http://soapinterop.org/echoheader/");
.....
-------//
```

In order to manipulate the above accessed HeaderBlock the handler writer can use the following code block.

```
// -----
.....
if (pIHeaderBlock != NULL) {
   const BasicNode* pBasicNode= pIHeaderBlock->getFirstChild();
   const AxisChar* pachHeaderValue;
   if (pBasicNode != NULL)
   {
   if((pBasicNode->getNodeType()) == CHARACTER_NODE) {
   pachHeaderValue= pBasicNode->getValue();
   } else {
   pachHeaderValue = "This was not the expected value Ros";
```

```
}
} else
{
pachHeaderValue = "pBascNode is NULL";
}
AxisChar* pachTmpValue = (AxisChar*) malloc(strlen(pachHeaderValue) + 4);
strcpy(pachTmpValue, pachHeaderValue);
pachTemp = "EchoStringHeaderHandlerPr1.id";
pIMsg->setProperty(pachTemp, pachTmpValue);
free(pachTmpValue);
} else {
//do some thing
 //AxisChar* pachTmpValue = "Default values since no reqeust SOAP header";
//pachTemp = "EchoStringHeaderHandlerPr1.id";
//pIMsg->setProperty(pachTemp, pachTmpValue);
 //free(pachTmpValue);
}
....
-------//
```

In order to get access to the incoming SOAP Body the handler writer can use the following code block.
To get the body as a AxisChar*

```
// -----
.....
IHandlerSoapDeSerializer* pIHandlerSoapDeSerializer;
pIMsg->getSoapDeSerializer(&pIHandlerSoapDeSerializer);
AxisChar* pSoapBody = pIHandlerSoapDeSerializer->getBodyAsChar();
.....
-------//
```

To get the body as a decoded base64 stream.

```
// -----
.....
IHandlerSoapDeSerializer* pIHandlerSoapDeSerializer;
pIMsg->getSoapDeSerializer(&pIHandlerSoapDeSerializer);
xsd__base64Binary bb = pIHandlerSoapDeSerializer->getBodyAsBase64Binary();
.....
-------//
```

Note:
Have a look at the following classes at the API docs to see all the available functions and

their respective descriptions. (You can even look at the relavent .h/.hpp header files for the API comments)
1.IhandlerSoapDeSerializer
2.IHeaderBlock
3.BasicNode
The BasicNode API is similar (not exactly the same, but ..) to the DOM and is written as a tree traversing API.
With the sample code samples provided above and with the API notes a developer will easily be able to write and play around his/her own Handlers.
This tutorial will be updated frequently with the new additions and specially with your suggestions.