Handler Tutorial

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

1. Contents

2. Introduction to Handlers

Handlers are pluggable components in Axis C++. Handlers are pieces of code that you write to intercept the message either on the client or service side. We have included a set of sample handlers for your reference.
You can write your own handlers by following the instructions given for the sample Handlers. At the end of this section we have also added some more detailed API information.

3. Handler samples

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.

4. echoStringHeaderHandler (A server side handler sample)

4.1. Building the echoStringHeaderHandler

4.1.1. Linux

```
The build files are available at
<Axis install directory>/samples/server/echoStringHeaderHandler Change your current dir
The handler libeshhandler.so file will be created at <Axis install directory>/lib direc
```

4.1.2. Windows

```
The VC dsw file (ServerHandlers.dsw) is available at
<Axis Install directory>/vc/samples/server/ ServerHandlers.dsw.
Open this file and build the project echoStringHeaderHandler. Once the build is success
```

4.2. Configuring the echoStringHeader Handler

Edit the server.wsdd file (as created when you configured your server) to include the handler for a particular service.
In this instance we are using the Calculator server example that we have used in both the client and server setup examples. The example below shows how a linux file would look e.g. libeshhandler.so is used please vary the file according to the libraries you have created. This example shows the handler being deployed on both the incoming and outgoing message.

<service name="Calculator" provider="CPP:RPC" description="Simple Calculator Axis C++ Service ">
<requestFlow name="CalculatorHandlers"> <handler name="ESHHandler" type="<Axis installation directory>/handlers/custom/echoStringHeaderHandler/libeshhandler.so"> </handler> </requestFlow> <responseFlow name="CalculatorHandlers"> <handler name="ESHHandler" type="<Axis installation directory>AXISCPP_DEPLOY/lib/libeshhandler.so"> </handler> </responseFlow>
<parameter name="allowedMethods" value="add sub mul div "/>
<parameter name="className" value="<Axis installation directory>/webservices/libcalculator.so" />
</service>
Note: Make sure you specify the correct path of the handler so in the server.wsdd file.
Now you are almost ready to run your server-side handler.
You have to restart the Apache server so that it picks up its new configuration and then that's it !

4.3. Running the echoStringHeader Handler

Since this Handler is configured to the Calculator web service in the above step, this Handler will be executed when a client sends a SOAP request to the Calculator web service. Use the calculator client you created earlier.

5. testHandler (A client side handler sample)

5.1. Building the testHandler

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

5.1.1. linux

```
makemake install
The handler so file will be created at <Axis installation directory>/lib/
```

5.1.2. windows

```
The VC dsw file (ClientHandlers.dsw) is available at <Axis Installation directory>/vc/s
Once the build is successful you will find the DLL testHandler.dll) at <Axis Installati
```

5.2. Configuring the testHandler

Now edit the **<Axis installation directory>/etc/client.wsdd** to include the handler for a particular service.

In this example we will continue to use the Calculator service. The sample wsdd file outlined below shows a linux directory system - for other operating systems please use the correct path symbols.
Note: Up until this point you did not need a client wsdd file the client only requires a wsdd file when it has handlers.

<service name="Calculator" provider="CPP:RPC" description="Calculator web service"> <requestFlow name="CalculatorHandlers"> <handler name="TestHandler" type="<Axis Installation directory>/lib/libtest_client_handler.so"> </handler> </requestFlow> </service>

Note:ClientWSDDFilePathaxiscpp.conf

5.3. Running the testHandler

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..

6. The Handler API's

Now you have seen some sample Handlers you can explore more on Handlers. The following sections should help you understand that API's available to you in your handler code.
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.