Axis C++ Developer's Guide

<!-- -->

1. Contents

This guide is meant to be read by those of you who would like to check out the code and create either a new transport layer or new parser layer. If you just want to check out the code and build it yourself e.g. because you want to port it to a new new platform then please see the build guide.

Before going through this guide, please go through the Windows User Guide and also be familiar with how to use binaries.

2. Checking out source code

The source code for Axis C++ is held in SVN. Instructions for using the Apache SVN system can be found here. The Axis C++ repository is http://svn.apache.org/repos/asf/webservices/axis/trunk/c.

3. Creating and Building your own transport layer

In very extreme cicumstances you might need to implement your own transport layer. If this is the case then you need to read thie following section.

When creating your own transport layer refer SoapTransport.h header file for API.To see an example implementation refer AxisTransport.h and AxisTransport.cpp

Implement SoapTransport.h interface according to rules described in the header file.
Transport layer is built separately from Axis. Then Axis loads transport dynamic library through following export functions which you also have to implement.

CreateInstance (SoapTransport *pOut)- Used by Axis to create an instance of your transport class
DestroyInstance (SoapTransport *pIn) - Used by Axis to destroy the created transport class

instance

Compile your transport code and build a dynamic library. Add the name of your transport library to axis configuration file (axiscpp.conf) so that Axis can find your library at runtime.

4. Debugging client problems using the MockServer

Within the ant test framework we have what we call a 'MockServer'. This is a 100 percent Java tool that allows you to send back any response that you wish to the client. This tool can be particularly useful for debugging problems in the client that come from the mailing list. You need the users WSDL (to create the stubs) and ask them to capture the response from the server (or you can make this by-hand if you wish it's just a bit harder).

Here are some simple instruction on using the MockServer utility we use for testing Axis c++ client without a server outside of the mockserver. Compile mock server java code (found within **<Axis extract root >/tests/utils/monitor**)

Run the mock server: **java org.apache.test.MockServer -p <port> -r <server response file**>

The server response file is a text file containing the full HTTP response, it will look something like this:

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.1
Content-Type: text/xml; charset=utf-8
Content-Language: en-GB
Transfer-Encoding: chunked
1ad
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header/>
<soapenv:Body>
<addResponse xmlns="http://localhost/axis/Calculator">
<addReturn>5</addReturn>
</addResponse>
</soapenv:Body>
</soapenv:Envelope> 0
```

The easiest way to generate this file is to capture the communication from a real server but if

that isn't possible you will need to generate by hand (plenty of examples within <extract root>/test/auto_build/testcases/output) Rather than providing the specific chunk size (1ad, on the line ahead of the payload) you can provide ### and the MockServer will calculate the correct size.

If your testcase makes multiple calls to the web service, you can simply append all the requests (in the correct order!) within the server response file.

Once the test has been completed you need to run the following to stop the server:
java org.apache.test.StopMockServer -p <port> -h <host>

5. Creating and Building your own parser

In very extreme cirumstances you may want to write your own parser layer. If you do then you need to read this section.

Note:Implement XMLParser.h interface according to the rules described in the header file.

When creating your own parser refer XMLParser.h header file for API. To see an example implementation refer SoapParserXerces.h and SoapParserXerces.cpp.

Axis loads the parser dynamic library through following export functions which you also have to implement.

CreateInstance() - Used by Axis to create an instance of your parser class
DestroyInstance() - Used by Axis to destroy the created parser class instance

Compile your parser code and build a dynamic library. Add the name of your parser library to Axis configuration file (axiscpp.conf) so that Axis can find your library at runtime

6. Adding support for extra platforms

If you are working on a platform not currently supported by Axis C++ the community would be very grateful for your assistance in providing support for that platform.

Instructions for adding an extra platform to the ANT build scripts can found here.

If you need to make any modifications to the code specific to your platform we ask that you make the appropriate updates in the platform abstraction layer, by doing the following:

- Create new `platforms\[platform]\PlatformSpecific[platform].hpp` file (and possibly also `.cpp` file), ensuring all macros and methods already in platform abstraction layer a copied across and updated as required.

- Update `platforms\PlatformAutoSense.hpp` to correctly detect platform and include correct header file.

If you need to change something that has not been previously abstracted please ensure you update the other platforms to contain the existing code.