

# Axis CPP - ANT Build Guide

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

## 1. Axis C++ ANT Build Guide

This document provides instructions for using and extending the ANT based build for the AXIS C++ project.

### Contents

- [Preparing system](#)
- [Getting necessary third party software](#)
- [Property Files](#)
- [Getting the source code](#)
- [Setting the Environment](#)
- [Running the ANT build](#)
- [Enabling Trace and Debug](#)
- [Adding an extra platform](#)

### Preparing system

To use the ANT based build you will need to install the following:

- Apache ANT
  - Available from <http://ant.apache.org>
  - These scripts have been developed and tested using version 1.6.x
- Java SDK - required for running of ANT scripts, and the compilation of WSDL2Ws tool
  - Require version 1.4+
- Ant-Contrib - provide numerous extensions to ANT, including the compilation of C/C++.
  - Available from <http://ant-contrib.sourceforge.net>
  - Place JARs into [ANT\_INSTALL\_DIR]/lib.
  - Require both ant-contrib (v0.6) and cpptasks (v1.0b3)
- (Optional) Doxygen - Used for generating API documentation
  - Available from <http://www.doxygen.org>
- Compiler / Linker
  - For Windows - Microsoft Visual C++ v6
  - For Linux - gcc / g++
  - For AIX - IBM Visual Age for C++
  - For Solaris - cc
  - For HP-UX - aC++

- For OS400 - icc

### Getting necessary third party software

Axis Cpp Developers use the Xerces-cXML Parsers to build the Axis Cpp. Additionally, you can opt to build Apache mod files for Apache 1.3 or 2.0.

### Xerces-C XML Parser

You can get Xerces-C binaries from <http://xerces.apache.org>.

### Apache

You can get Apache 1.3 or 2.0 from <http://httpd.apache.org/>

### Property Files

The behaviour of the ANT scripts is controlled by the values specified in `build.common.properties`. To aid in the portability of the ANT scripts, there are also a number of platform specific property files which override properties as required. The ANT scripts will automatically determine the correct file to be used based on the platform in which it is currently running. The property files are found in `ws-axis/c/build` with the following naming convention:

```
build.common.properties build.[platform].properties
```

A number of example property files are provided for Windows, Linux, AIX and Solaris. It is intended you update these files to suit your development and build environment, including location of third party software dependencies and target packaging structure.

These property files also allow you to make some selection on which artefacts will be produced by the build:

- Select which Apache module to produce:
  - For each one to be built, set the following to true:  
`server.apache13`  
`server.apache20`
- Select whether to build Simple Axis Server executable:
  - If you wish to build this, set the following to true:  
`server.simpleAxisServer`

The default selections are for both the Apache 1.3 and Apache 2.0 modules.

### Setting the Environment

Before running ANT the following environment variables must be set:

- `ANT_HOME` - location of ant installation
- `JAVA_HOME` - location of java installation
- `PATH` - to include `[ANT_HOME]/bin` and `[JAVA_HOME]/bin`.

## Axis CPP - ANT Build Guide

- Also ensure doxygen and compilers are available on the system path.

The default property files make use of the following environment variables to locate the various third party software dependencies.

- `AXISJAVA_LIB` - location of Axis Java JAR files, as required for WSDL2Ws tool
- `XERCES_HOME` - location of Xerces installation
- `APACHE_HOME` - location of Apache 1.3 installation (*if building Apache 1.3 module*)
- `APACHE2_HOME` - location of Apache 2.0 installation (*if building Apache 2.0 module*)

### Getting the source code

See [here](#) for more detail on gaining access to the code in SVN.

Note: The ANT scripts were written based on the previous CVS structure and have not been updated to the SVN structure, so you will need to ensure your checkout is into the following structure:

```
[build root]\ws-axis\c
```

The checked out folder `ws-axis\c` will be referred to as `[CHECKOUT_HOME]` from this point on.

### Running the ANT build

Once you have configured your environment and property files the build is a simple two step process. The first step is to build all the generated artefacts. At the comment prompt change to `[CHECKOUT_HOME]` and run:

```
ant
```

This will build the following:

- Client library
- Transport libraries
- XML Parser library
- Server library
- Apache modules
  - Apache 1.3 or Apache 2.0 configurable through the property files
- Simple Server Executable
  - Configurable through the property files
- WSDL2Ws tool
- API Documentation

The second step is to package the generated artefacts. From `[CHECKOUT_HOME]` run:

```
ant -f package.xml
```

This will package into two distributables:

- binary
- source

### Enabling Trace and Debug

By default, the ANT build scripts do not produce libraries with trace or debug symbols. To include these make use of one of the following to build:

```
ant buildWithTrace
ant buildWithDebug
ant buildWithTraceAndDebug
```

The packaging step remains the same, but when packaging the source distributable it will automatically select the trace instrumented source code if available.

When trace is selected, the ant build adds in trace entry and exit statements into many of the methods in Axis C++. Then at runtime, in `axiscpp.conf`, set `ClientLogPath` to a file and Axis C++ will write out trace to that file. Omitting `ClientLogPath` from `axiscpp.conf` switches trace off.

### Adding an extra platform

The AXIS community would greatly appreciate your input, if you're working on a platform not currently supported by the ANT scripts.

Below, are the steps required to add an additional platform;

1. Add platform detection to `pre-init` target in `buildInitialize.xml`, eg:  

```
<condition property="linux">
  <os name="Linux"/>
</condition>
```
2. Update platform property within `initialize` target in `buildInitialize.xml`, eg:  

```
<condition property="platform" value="Linux">
  <isset property="linux"/>
</condition>
```
3. Provide an additional property file in `ws-axis/c` to match your platform. This uses the naming convention `build.[platform].properties`, where `platform` is as specified in step 2.
4. Provide compiler definition for platform in `buildInitialize.xml`, include a condition check for the correct platform and any debug flags should be conditional on the debug property being set, eg: 

```
<compiler id="Linuxgcc" name="g++" if="linux">
  <compilerarg value="-g" if="debug"/> <compilerarg value="-Wall"/>
  <compilerarg value="-Wshadow"/>
  <compilerarg value="-O2"/>
  <defineset>
    <define name="ENABLE_AXIS_EXCEPTION"/>
    <define name="HAVE_CONFIG_H"/>
    <define name="PIC"/>
  </defineset>
  <includepath path="${dir.include}"/>
</compiler>
```

Note: Compilers may extend one another, which can be useful if an

additional platform uses the same compiler, but maybe only small variations in the parameters.

5. Provide `linker` definition for platform in `buildInitialize.xml`, include a condition check for the correct platform and any debug flags should be conditional on the debug property being set, eg: `<linker id="LinuxLinker" name="g++" libtool="true" if="linux">`  
`<linkerarg value="-g" if="debug"/>`  
`<libset libs="stdc++"/>`  
`</linker>`Note: As for compilers, linkers may extend one another.
6. Add new `compiler` and `linker` to the various `cc` tasks used within the ANT scripts, eg: `<cc outfile="${dir.bin}/${transportLibraryName}" objdir="${dir.objects}" exceptions="true" failonerror="false" outtype="shared" multithreaded="true">`  
`<!-- Compilers -->`  
`<compiler refid="Linuxgcc"/>`  
`<compiler refid="AIXxlc"/> ...`  
`<!-- Linkers -->`  
`<linker refid="LinuxLinker"/>`  
`<linker refid="AIXLinker"/>`  
`...`  
`</cc>`