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.

1.1. 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 and cpptasks
- 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++

1.2. Getting necessary third party software

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

Expat XML Parser

You can get expat binaries from http://sourceforge.net/projects/expat/.

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/

1.3. Property Files

To aid in the portability of the ANT scripts, a number of property files are used. The script will decide which to use based on the platform in which it is currently running. The property files are found in `ws-axis/c` with the following naming convention:

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

A number of example property files are provided for Windows, Linux, AIX and Solaris, it is intended that you update these files to suit your development and buid environment. This includes 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 XML Parsers to use:
  - For each one to be built, set the following to true:
    `xmlParser.xml4c`
    `xmlParser.xerces`
    `xmlParser.expat`
- Select which transport implementation to use:
  - For each one to be built, set the following to true:
    `transport.axis`
    `transport.axis2`
    `transport.libwww`
- 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 Xerces as XML parser, axis2 transport implementation and both the Apache 1.3 and Apache 2.0 modules.

1.4. 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`.
  - 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
- `EXPAT_HOME` - location of Expat installation *(if using Expat*)
- `XERCES_HOME` - location of Xerces installation (*if using Xerces*)
- `XML4C_HOME` - location of XML4C installation (*if using XML4C*)
- `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*)

1.5. Getting a CVS checkout

Visit http://ws.apache.org/ Click on “axis” and then on “CVS Repository” to find details on how to access the CVS Repository.
In short summary:
Anyone can checkout the 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 loginpassword: anoncvs cvs -d :p
```

The checkout of the repository will be created in the current directory in a folder named “`ws-axis`”
The checked out folder ws-axis/c will be referred to as `[CHECKOUT_HOME]` from this point on.

1.6. 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 carry out the following:

- Build Axis C Client library
- Build Axis C Transport library
  - Axis, axis2, libwww configurable through the property files
- Build Axis C XML Parser library
  - Expat, Xerces-C, etc configurable through the property files
- Build Axis C Server library
- Build Axis C Apache module
  - Apache 1.3 or Apache 2.0 configurable through the property files
- Build Axis C Simple Server Executable
  - Configurable through the property files
- Build and package WSDL2Ws tool
- Generate API Documentation
- Package artefacts into a distributable structure
  - This structure is configurable through the property files
- Validate all artefacts have been correctly generated

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

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

This will carry out the following:

- Package artefacts in to a binary release distributable
- Package artefacts in to a source release distributable

To remove artefacts from a previous build use the following command:

```
ant clean
```

1.7. 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. Although it the source release package will automatically select the trace instrumented source code.

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 in a

directory somewhere and Axis C++ will write out trace to that file. Omitting ClientLogPath from axiscpp.conf switches trace off.

1.8. 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 `buildIntialize.xml`, eg:
   <condition property="linux">
   <os name="Linux"/>
   </condition>
2. Update platform property within `initialize` target in `buildIntialize.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 `buildIntialize.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 `buildIntialize.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 `cc` task within each of `compileAxisClient`, `compileAxisTransport`, `compileAxisXMLParser`, `compileSimpleAxisServer`, `compileAxisServerEngine`, `compileApache13Module` and `compileApache20Module` targets, 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>