Ant revision guidance and scripts

Ant

Apache ant is a build tool that can be used to perform various activities of the development life-cycle. It can be used for testing, compiling, building, archiving and deploying an application.

Because Apache Ant was developed using Java it can be run on a range of platforms and can be considered platform independent.

Ant builds file are formatted using XML with the file structured as a series of targets. A target represents a stage in the build process. For each target there is a corresponding resource file which must be produced. For example this could be a java archive file or a Java class for compilation.

For each target there are typically a number of source files and object files. For example when compiling java files the source files are .java files and the object files .class files. The ant Java program will compile the source file if the object file is older than the source file or if the object file doesn't exist.

For some targets, there may be other targets which must be built before them. For example code must be compiled before it can be tested. In this case 1 target depends on another. So for each target we can specify one or more other targets as dependencies,

A dependency of 1 target, can be another target, this means there can be a chain of targets all depending on one another. See the following example, compile depends on init, so init has to be checked and run before compile,

```
<target name="init">
    <mkdir dir="build"/>
 </target>
<target name="compile" depends=”init”>
    <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
</target>
```

Each target has a name and enclosed a number of ant task elements which are needed to complete that stage of the build. Each target can be marked as depending on one or more than one other targets, each dependency will be checked first, before the main target is run. These dependencies then form a free which is search backwards to determine what targets must be run first, second, third etc.

Each target also has a set of rules on which to build it, each rule can do the following types of actions, compile a file, run a JUnit test or run an external Java program. In this way Ant is totally extendable.

Ant provides the facilities of properties which can be used to customize a particular task, for example containing passwords or directory names.

The property values themselves can be defined in a properties file, passed in on the command line when running ant or defined in the ant build file using the property task.

Example

<property name="src" value="source"/>

When properties are used, there name is pre-fixed with

${ and closed with } for example ${src}.

When a property has been set it is immutable, this means its value cannot be changed later on.

Compiling code javac task

There is a task built into Ant which allows it to compile java programs.

Here is an example of the use of this task, called javac.

```
<javac srcdir="src"
    destdir="build\classes"
    classpath="junit.jar"
    debug="on"/>
```

srcdir is root of the java source file directory, this will be searched recursively (all sub-directories and their sub-directories for the destination, class path is the java class path, destdir is where the class files will be stored according to their packages javac will check if the source file is newer than its corresponding class file and skip the compile if it is not.

Ant task examples

javac

Compile java source files.

jar

Produce java archive

junit
Runs Java unit testing framework JUnit

junitreport
Produces HTML report from Junit xml report files

Ant scripts

See directory antScripts

For each script examine the xml and run the script as instructed.

example1.xml

Simple hello world example, to run type

ant –f example1.xml

example2.xml

This shows an ant property being set and then echoed.

Type

ant –f example2.xml

Now try the same script but set the property outside and see what happens.

ant -Dsrc=new_dir -f example2.xml

The output should be

propertyExample:

    [echo] new_dir

This is because the property is immutable, once set on the command line at is not changed inside the ant script. You cannot generally override the value of the property once it is set.

example3.xml

This shows java compiling.

ant –f example3.xml

If you run the file more than once notice it skips the compiling, unless you change the java code or delete the class file.

example4.xml

This shows what happens if you use the wrong package name or the incorrect directory for the java source file.

ant –f example4.xml

You will notice if you run this lots of times, it compiles the file every single time, this is because the class file is stored in the directory according to the package name. Inspect the files in src3 and build3.

example5.xml

Run as

ant –f example5.xml

This case is similar to example4, the Java source file has a package defined but in this case it is not stored in the correct source directory. Again the file will be compiled every time because the class file is stored according to its package definition. Examine the build5 directories and the src5 directories.

Running Junit from Ant

See the following example:

```
<target name="test" depends="compiletest">
        <junit printsummary="on" haltonfailure="no" fork="true">
      <classpath>
        <path refid="class.path"/>
      </classpath>
      <formatter type="brief" usefile="false" />
      <batchtest>
        <fileset dir="testsrc" includes="**/*Test.java" />
      </batchtest>
   </junit>
```

The class path is defined by the reference id which is defined in another class.

The formatters show what type of output is required, in this case a brief report but not stored in a file (usefile=”false”).

This will look to test all the files with the given pattern below testsrc

Possible outcomes of test

Success
The test passed, all assertions were successful.

Failure
The test failed, this means one or more than 1 assertion failed, this will usually stop the build except if haltonfailure=”no”

Error
This means there was an error for example a null pointer exception when running 1 of the tests. In this case again the build will stop and report an error unless haltonerror=true.

Notice the different between haltonfailure and haltonerror.