



Building Simple Java Projects



Agenda

1

Structure of a build file

2

Some useful Targets

3

Some useful Tasks

Objectives

At the end of this module, you will be able to:

- Understand the structure of build.xml
- Understand some important targets and tasks in ANT

Structure of a Build File



A sample build file – build.xml

```
<project name="My first Ant"
default="one">

<target name="one">
<echo>From one</echo>
</target>

<target name="two">
<echo message="I'm from two"/>
</target>

</project>
```

Type the following contents
in a notepad and save it as
build.xml

Output :

D:\>ant

Buildfile: D:\build.xml

one:

[echo] From one

BUILD SUCCESSFUL

Total time: 0 seconds

Open the command prompt and type
ant in the location where the
build.xml file is saved

Structure of the Build file - <project>

- The root tag of the build.xml file is **project**
- Every build.xml will have a single Project
- Every Project will have at least one **target**
- Every target will have one more **task** elements

<project name="My first Ant" default="one">

- The above code snippet indicates that
 - The name of the project is My first Ant
 - The default attribute is assigned “one”
 - Here the target ‘one’ gets automatically executed as soon as we execute the ant command
 - If you call ant without any arguments it will execute one
 - If you issue the ant command as ‘ant two’ it will execute target two

Assignment

1. Create build.xml in a notepad file
 2. Save it in D:\
 3. Open command prompt, go to D drive and issue the **ant** command
 4. Note the output generated
-
1. In the same command prompt, type **ant two**
 2. Note down your results
-
1. Now remove the default attribute in project tag
 2. In the command prompt, give **ant** command and note your observations

<target>

- All targets must have a name such as “compile”, “init”, “clean”
- Can have a “description” to explain the target’s purpose
- Targets can depend on each other
 - If a target depends on another target it will demand that the other target gets executed first

```
<target name="jar" depends="compile">  
.....</target>
```

- Multiple targets can be specified in depends; they will be executed in the order listed

```
<target name="jar" depends="prepare, compile">  
</target>
```


Structure of the Build file - <target>

- The build file can contain many number of targets
- Each target consists of a series of tasks that are specific actions for ant to take
- A target can have the following attributes
 - name (required)
 - depends (comma separated list of other targets)
 - if
 - unless
 - description

An Example of if and unless attribute

```
build1.xml - Notepad
File Edit Format View Help

<project name="sample" default="one">
  <property name="name" value="john"/>

  <target name="one" if="name">
    <echo message="Welcome ${name}"/>
  </target>

</project>
```

```
C:\> C:\WINNT\system32\cmd.exe

C:\>ant -f build1.xml
Buildfile: C:\build1.xml

one:
    [echo] Welcome john

BUILD SUCCESSFUL
Total time: 0 seconds

C:\>_
```

If the attribute is declared the target gets executed

```
build1.xml - Notepad
File Edit Format View Help

<project name="sample" default="one">
  <property name="name" value="john"/>

  <target name="one" unless="name">
    <echo message="Name property is not defined"/>
  </target>

</project>
```

```
C:\> C:\WINNT\system32\cmd.exe

C:\>ant -f build1.xml
Buildfile: C:\build1.xml

one:

BUILD SUCCESSFUL
Total time: 0 seconds

C:\>
```

The target gets executed if the property is not declared

Assignment

1. Understand why the name attribute is mandatory in any target
2. Create few more targets called three, four and five. Let target one be dependent on two, let two be dependent on three, three be dependent on four, four be dependent on five
3. Issue ***ant one***, and note your observations
4. Issue ***ant three*** and note down your observations
5. Declare a property called '***name***' as shown in the previous slide and try ***if*** and ***unless*** attribute

Some important targets



Target - init

- The first target you will probably specify is “init”
 - Used to set up properties that you will use throughout your build file
 - Properties can be set directly in the build file or by specifying all the properties in a property file
 - The advantage of setting variable here is, it gives you one place to make changes that ripple through the entire file
 - Using a properties file is even better because the syntax is simpler

```
<target name="init" >  
  <property file="build.properties" />  
</target>
```

Target - prepare

- The “prepare” target depends on “init”
- This target is used to create the **directory structure** that is needed

```
<target name="prepare" depends="init">  
    <mkdir dir="${build.dir}"/>  
    <mkdir dir="${build.dir}\META-INF"/>  
</target>
```

- Notice the `${build.dir}` syntax; this is dereferencing a property that was set in the “init” target

Tatget - clean

- “clean” is useful for enabling clean builds
- Generally clean just deletes stuff from previous runs to ensure that no artifacts persist between builds

```
<target name="clean" depends="init">  
    <delete dir="${build.dir}"/>  
</target>
```

- For clean build, you should execute the “clean” target before trying to build anything

Target - compile

- Compile some/ all of your source files in this target

```
<target name="compile" depends="prepare">
  <echo message="Compiling now..." />
  <javac srcdir="${src.dir}"
        destdir="${build.dir}" includes="**/*.java" />
</target>
```

- “**/*.java” notation means “all files recursively” from ‘srcdir’ that ends with .java

Target - jar

- Once you've compiled everything, we should create the jar file

```
<target name="jar" depends="compile">  
    <jar destfile="${dist.dir}/${name}.jar"  
        basedir="${build.dir}"/>  
</target>
```

- This builds a jar file
 - The name of the jar file is as pointed by the property \${name}
 - The Jar file will be created In the directory as pointed by \${dist.dir}
 - It includes everything from the \${build.dir} directory

Built in tasks



Built-In Tasks

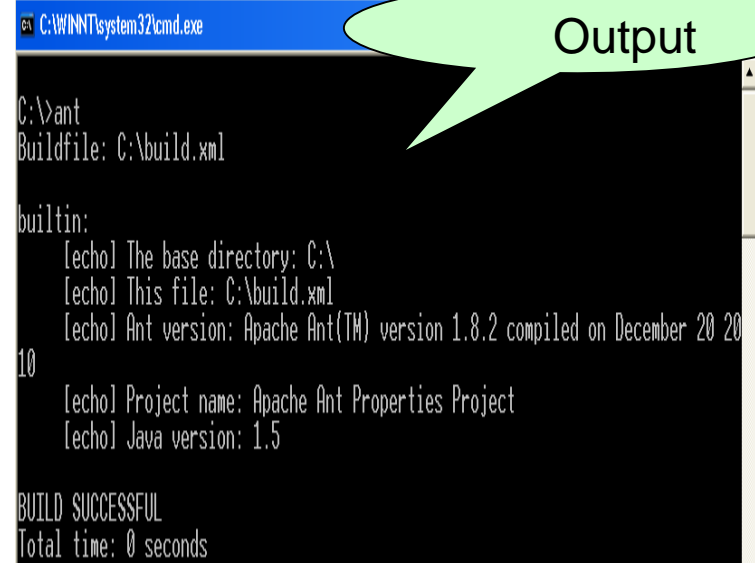
- We will now look at some of the many built-in tasks that ship with Ant
- We will show some of the most common options; the others are left as exercise...
- Ant comes with extensive documentation on these tasks showing all possible options
- We will begin with “property”, “javac” and “jar” and go into the others in alphabetical order

Working with Properties

- Hard-coding directory paths and filenames is not a good idea in any area of programming
- Properties comes handy in this case
- The \${} notation is used by Ant to retrieve the value of the Property

Displaying default properties of Ant

```
<project name="Ant Properties Project"
basedir="." default="builtin">
  <target name="builtin">
    <echo message="The base directory:
${basedir}"/>
    <echo message="This file: ${ant.file}"/>
    <echo message="Ant version: ${ant.version}"/>
    <echo message="Project name:
${ant.project.name}"/>
    <echo message="Java version:
${ant.java.version}"/>
  </target>
</project>
```



Output

```
C:\>ant
Buildfile: C:\build.xml

builtin:
[echo] The base directory: C:\
[echo] This file: C:\build.xml
[echo] Ant version: Apache Ant(TM) version 1.8.2 compiled on December 20 20
10
[echo] Project name: Apache Ant Properties Project
[echo] Java version: 1.5

BUILD SUCCESSFUL
Total time: 0 seconds
```

Setting properties in a build.xml

- The <property> element is defined as a **task**, unlike the <project> and <target> elements
- The <property> elements can be included inside a target depending on which target has been selected
- Properties can also be set at the beginning of a build file so that they apply to the entire build

```
<target name="properties.custom">  
  <property name="version" value="1.1"/>  
  <echo message="Version. = ${version}"/>  
</target>
```

Properties file

- All the property values used in a build.xml can be grouped in a separate 'properties' file
- The property values are immutable
- If you set a property value in the properties file you cannot change it in the build file

Sample.properties

```
web.lib.dir=${web.dir}/WEB-INF/lib  
build.classes.dir=build/classes
```

Using that in build.xml

```
<property file="Sample.properties" />
```

Assignment

1. Write a build.xml to print some of the built-in ANT properties
2. Name the file as built-in.xml and execute it
3. Create an xml file with user defined properties and display all of them
4. Store the properties in a file called build.properties and print them using the xml file
5. Create the targets called init (which is used to initialize properties), prepare (which is used to create directories) and clean (which is used to delete the build directory) and check how it works

<javac>

- The javac task compiles java source into class files, just as the javac command-line tool does
- If you store your source code in directory structures that match the package structure of your application, then Ant will recompile only those files that have changed (very useful)
- It's possible to use different compilers
- For eg:
 - classic - the standard compiler of JDK 1.1/1.2
 - modern - the standard compiler of JDK 1.3/1.4/1.5/1.6/1.7/1.8)
 - gcj - the gcj compiler from gcc
- This can be specified by setting the global **build.compiler** property

<javac> (Contd.).

```
<javac srcdir="${src.dir}" destdir="${build.dir}"  
  Optimize="off" debug="on" deprecation="on"  
  Classpath="${classpath}" includes="**/*.java"  
  excludes="**/*Test*" />
```

- This task will
 - Compile all the java source files, except those files which have “Test” in their name names in the \${src.dir} directory
 - Place the compiled classes in the \${build.dir}
 - Include debug information in the .class files
 - No need to perform any optimization
 - Should show deprecation warnings

<javac> (Contd.).

- The “srcdir” attribute is optional if there are nested “src” tags

```
<javac ..>  
  <src path="${src.dir}"/>  
  <src path="${other.src.dir}"/>  
  <exclude name="com/xyz/**/*.*.java" />  
</javac>
```

- The task above will
 - Compile source files in the directory specified by both the \${src.dir} and \${other.src.dir} properties and their children
 - It exclude files in the com.xyz package and any sub package

<javac> (Contd.).

- If “destdir” is omitted, the compiled .class files will be placed in the source directory itself
- “debug”, “optimize” and “deprecation” are all “off” by default

```
<javac srcdir="${src.dir}"/>
```

<jar>

- Creates a jar file

```
<jar destfile="${dist.dir}/${name}.jar" basedir="${build.dir}" />
```

- The *basedir* attribute is the reference directory from where to jar
- This includes everything under `${build.dir}` in the jar file based on the “name” property, located in the directory specified by the “dist.dir” property
- Includes a MANIFEST.MF file

<jar> (Contd.).

- Also supports includes and excludes

```
<jar basedir="${build.dir}"  
destfile="${dist.dir}/${name}.jar"  
    includes="**/*.class"  
    excludes="**/*Test*.class" />
```

- Supports a nested <metainf> tag to specify what should be included in the META-INF directory (you can specify multiple <metainf> tags to pull files from different directories)

```
<jar ...>  
    <metainf dir="${etc.dir}" includes="**/*.xml" />  
</jar>A
```

Assignment

1. In a folder called src, create some java files. Using a build.xml compile all those .java files and keep the .class files in a folder called build. Include the init, prepare, clean and compile targets and test
2. To the above build.xml add the jar target and test

<ant>

- It is used to call ant from within ant on a different buildfile
- Useful for building subprojects (e.g. partitioning large projects)

```
<ant dir="./test" antfile="build.xml" target="compile"  
      output="test.log" >  
      <property name="param1" value="1.0" />  
</ant>
```

- If “antfile” is omitted, “build.xml” is assumed
- If “target” is omitted, the default target is used
- If “output” is omitted, output goes to the console

Assignment

1. Create 2 xml files called build1.xml and build2.xml file. Create a target called target1 in build1.xml and using <ant> call invoke target called target2 which is there in build2.xml.
2. Declare a property called 'course' in build1.xml file and initialize it with a value called 'ant'. In build2.xml file also declare a property called 'course' and initialize it with a value called 'junit'.
In target2 of build2.xml print the value of the property 'course' and check what happens
3. Now remove the property called 'course' from build1.xml and run build1.xml and check your result

<antcall>

- Calls another target within the same buildfile
- Can pass parameters to it
- Useful for reusing targets

```
<target name="foo">
    <echo message="param1="${param1}" />
</target>

<target name="bar">
    <antcall target="foo">
        <param name="param1" value="test"/>
    </antcall>
</target>
```

<antcall> example

```
build.xml - Notepad
File Edit Format View Help

<project name="MyProject" default="one">
  <target name="test" >
    <echo message="From test"/>
  </target>

  <target name="one">
    <echo message="I'm from One"/>
    <antcall target="test"/>
    <echo message="Back to One"/>
  </target>

</project>
```

```
C:\WINNT\system32\cmd.exe

C:\antsample\antcall>ant
Buildfile: C:\antsample\antcall\build.xml

one:
    [echo] I'm from One

test:
    [echo] From test
    [echo] Back to One

BUILD SUCCESSFUL
Total time: 0 seconds

C:\antsample\antcall>_
```

<available>

- Sets a property if a specified resource is available on the current system
- The resource could be a directory, a file, a class in the classpath, or a JVM system resource.
- Useful for targets that can't/ shouldn't be executed under certain circumstances

```
<!-- sets the "has.junit" property to "true" if the
      TestCase class is found in the classpath →
<available property="has.junit"
      classname="junit.framework.TestCase" >
      <classpath refid="classpath" />
</available
```

<available> (Contd.).

- Looks for the file called “foo.txt” and sets the “file.there” property if found

```
<available property="file.there" file="./test.txt" />
```

- Looks for the “test.properties” resource in the classpath and sets the property if found

```
<available properties="test.there"  
  resource="test.properties" >  
  <classpath refid="classpath" />  
</available>
```

<condition>

- Sets a property if a given series of requirements are met
- Several nested conditions are available
 - and, or, not, os, available, uptodate, equals, istrue, isfalse,...
- Sets the property javamail.complete if both the JavaBeans Activation Framework and JavaMail are available in the classpath.

```
<condition property="javamail.complete">  
  <and>  
    <available  
      classname="javax.activation.DataHandler"/>  
    <available classname="javax.mail.Transport"/>  
  </and>  
</condition>
```

<copy>

- Copy a directory, excluding some files

```
<copy todir="../../test">  
  <fileset dir="${src.dir}" excludes="**/*Test*.java"  
  />  
</copy>
```

- Copy a directory, renaming the files !

```
<copy todir="../../test" >  
  <fileset dir="${src.dir}" />  
  <mapper type="glob" from="*" to="*.bak" />  
</copy>
```

<echo>

- Prints a message to the console or a file
- Can expand properties just like other tasks
- If the “file” parameter is omitted, the console is used
- The “append” parameter works with the “file” parameter when an output file exists
- Works like the “echo” command (UNIX/DOS)
- Send a static message to the console

```
<echo message="now compiling.." />  
<echo> This is another way to do it </echo>
```

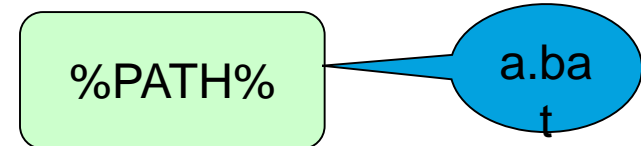
- Append a dynamic message to a file

```
<echo message="myvar=${myvar}" file="log.txt"  
  append="yes" />
```

<exec>

- Executes a system command
- Command line arguments should be specified as nested <arg> elements
- The <exec> task delegates to Runtime.exec
- The command shell executable 'cmd' should be executed using the /c switch

```
<target name="help">  
  <exec executable="cmd">  
    <arg value="/c"/>  
    <arg value="a.bat"/>  
  </exec>  
</target>
```



Output

The screenshot shows a Windows command prompt window titled "C:\WINNT\system32\cmd.exe". The command prompt shows the following text:

```
C:\>ant  
Buildfile: C:\build.xml  
  
help:  
[exec] 'C:\Program' is not recognized as an internal or external command,  
[exec] operable program or batch file.  
[exec]  
[exec] C:\>C:\Program Files\Java\jdk1.5.0_10\bin;D:\app\aname\product\11.1  
0\db_1\bin;C:\Program Files\Oracle\jre\1.3.1\bin;C:\Program Files\Oracle\jre\1.  
1.8\bin;C:\Program Files\Java\jdk1.5.0_10\bin;C:\Program Files\Oracle\bin;C:\Pro  
gram Files\Java\jdk1.5.0_10\bin;D:\Anitha\java revamp\ant\apache-ant-1.8.2\bin;  
D:\Anitha\java revamp\ant\apache-ant-1.8.2\bin  
[exec] Result: 1  
  
BUILD SUCCESSFUL  
Total time: 0 seconds
```


<javadoc>

- Generates javaDoc from your source files
- Much easier than running javadoc at the command line
- You really need to structure your source according to your package structure for best results
- Here's the simple case, generating the docs for the packages listed, in `${src.dir}`, placing the generated files in `${doc.dir}`

```
<javadoc sourcepath="${src.dir}" destdir="${doc.dir}"  
        packagenames="foo.*,bar.*,baz.*" />
```

<javadoc> (Contd.).

- Some of the useful parameters are
 - sourcepath – where the sources to process live
 - destdir – where to store the generated HTML files
 - author –includes the @author information
 - version – includes the @version information
 - access –which level of access should be included (e.g public, private); protected is default
 - packagenames – those packages to include in the doc
 - excludepackagenames –packages not to include

<mail>

- Used to send email to the email address specified
- Useful for overnight builds
- The log of what happened can be sent to you in the morning
- Also useful if a job break to notify someone immediately

```
<mail from="anitha.ramesh@wipro.com"  
  tolist="you@yourbox.com" subject="Nighty build  
  results" files="log.txt" />
```

It will go as an
attachment

<mail> (Contd.).

- Could also send email based on a message string

```
<mail from="anitha.ramesh@wipro.com"  
  tolist="you@yourbox.com" subject="Nighty build  
  results" message="Finished building  ${name}"  />
```

- 'tolist' can be list of email addresses separated by commas

<mkdir>

- Create a directory

```
<mkdir dir="${build.dir}/temp" />
```

- Create an entire directory structure !

```
<mkdir dir="${build.dir}/foo/bar/baz" />
```

<move>

- Like copy, only it moves things
- Like the Unix “mv” program, can be used to rename things

```
<move file="foo.java" tofile="foo.save" />
```

- Or just move a file

```
<move todir="${java.dir}">  
  <fileset dir="${src.dir} includes="**/foo*.java" />  
</move>
```

<parallel>

- Runs two or more Ant tasks (not targets) simultaneously in separate threads
- Useful if you have extra horsepower on a box or have several tasks that are independent and it lasts for a while
- Caution must be exercised to not parallelize tasks that might crash (eg. Two javac tasks)

```
<parallel>  
  <copy todir="${remove.drive.one}" .../>  
  <copy todir="${remove.drive.two}" .../>  
</parallel>
```

<tstamp>

- By default, sets the DSTAMP, TSTAMP and TODAY properties in the build file
- These properties can be referenced later in the file
- Set DSTAMP, TSTAMP and TODAY

```
<tstamp/>
```

- Specifies a property name and a format

```
<tstamp>  
  <format property="TODAY_US"  
    pattern=" dd-MMMM-YYYY" />  
</tstamp>
```


<tstamp> (Contd.).

- Specify the jar file name using one of the default <tstamp> properties

```
<tstamp/>  
<property name="jarname"  
  value="\${name}-${DSTAMP}.jar" />
```

- Specify the jar file name using a custom property

```
<tstamp>  
  <format property="TODAY_US" pattern="d-MMMM-yyyy" />  
</tstamp>  
<property name="jar.name" value="\${name}-  
  \${TODAY_US}.jar" />
```

Assignment - Using ANT with Eclipse

Step 1: Create an Eclipse Project

Step 2: Create a Java program

Step 3: Right click project and create build.xml file

Step 4: Execute the build.xml

a. Right click build.xml and select Run→ Ant Build or

Enable ANT view by following

Window > Show View > Other > Ant > Ant

Practice all the ANT targets and tasks

Assignment - Using ANT with Eclipse

Step 1: Create an Eclipse Project

Step 2: Create a Java program

Step 3: Right click project and create build.xml file

Step 4: Execute the build.xml

a. Right click build.xml and select Run→ Ant Build or

Enable ANT view by following

Window > Show View > Other > Ant > Ant

Practice all the ANT targets and tasks given in this module

Quiz

1. Which of the following is a mandatory parameter for javac task?
 - a. destdir
 - b. srcdir
 - c. debug
 - d. Classpath

2. What will happen if we execute a build.xml file with the following
<project name="MyProject">
...
</project> tag as E:\>ant build.xml
 - a. It'll look for build.xml in E:\ and will execute it if found
 - b. If build.xml is not found in E:\ it'll throw an error message
 - c. Both a and b are correct

Quiz

3. What will happen when we issue the following command?

E:\>ant

- a. It'll look for build.xml in E:\ and will execute it if found
- b. If build.xml is not found in E:\ it'll throw an error message
- c. Both a and b are correct

4. What does the following indicate?

E:\> ant -f mybuild.xml compile

- a. The buildfile to be used is mybuild.xml
- b. The target to be executed in mybuild.xml file is 'compile' target
- c. Both a and b are correct

Quiz

5. What happens with the following execution

E:\> ant mybuild.xml

- a. It'll execute the mybuild.xml file if it is found in E drive
- b. The build will fail

6. How do we make the earlier command work?

- a. By changing the file name to build.xml and executing it as ant build.xml
- b. By using a -f option and executing it as ant -f mybuild.xml file
- c. Both a and b are correct

Quiz

7. Which of the following are build tools?
 - a. Ant
 - b. Makefile
 - c. Both a and b are correct

8. Which of the following are true regarding Ant?
 - a. Ant is platform dependent
 - b. Ant is not a programming language
 - c. Ant can compile source code
 - d. Ant can run unit tests
 - e. Ant can package compiled code and resources

9. Properties in Ant should be defined for
 - a. Any piece of information that needs to be configured
 - b. Any piece of information that might change
 - c. Both a and b are correct

Quiz

Match the following:

- | | |
|----------------------|---|
| 1. Author of ANT | a. points to ANT installation directory |
| 2. Build file of ANT | b. ANT built-in property |
| 3. ANT_HOME | c. build file in Unix environment |
| 4. ant.version | d. Duncan Davidson |
| 5. Makefile | e. build.xml |

References

1. Apache Ant 1.8.2 Manual(2012). *Overview of Apache Ant Tasks*. Retrieved on April 10, 2012, from, <http://ant.apache.org/manual/tasksoverview.html>



Thank You

