

# Task Dependencies: ant

Steven J Zeil

February 25, 2013

## Contents

<b>1 The ant Command</b>	<b>3</b>
<b>2 Build Files</b>	<b>3</b>
2.1 Targets . . . . .	4
2.2 Properties . . . . .	8
2.3 File Sets and Lists . . . . .	9
2.4 Path Sets . . . . .	11
2.5 Filters . . . . .	11
2.6 Tasks . . . . .	12
<b>3 Case Studies</b>	<b>14</b>
3.1 Code Annotation . . . . .	14
3.2 Projects with Multiple Sub-Projects . . . . .	18
3.3 Extend or Exec? . . . . .	22
<b>4 Eclipse/Ant Integration</b>	<b>34</b>

**ant**

**ant** is a build manager based upon a task dependency graph expressed in an XML file

- **ant** devised by James Davidson of Sun, contributed to Apache project (along with what would eventually become TomCat), released in 2000
- Quickly became a standard tool for Java projects
  - slower to move into other arenas

.....

**What's Wrong with make?**

**ant** is actually an acronym for Another Neat Tool.  
But why do we need “another” tool for build management?

- **make** works by issuing command to **/bin/sh**
  - That's not portable.
- The commands that people write into their makefile rules are generally not portable either:
  - Commands themselves are system-dependent (e.g., **mkdir**, **cp**, **chmod**)
  - Paths are system-dependent (/ versus \, legal characters, quoting rules)
  - Path lists are system-dependent (: versus ;)

.....

**Other Criticisms**

- Some feel that **make** is too low-level with tis focus on individual files
  - Some will feel that **ant** is too high-level
  - But this is the apparent rationale for moving the focus from file dependencies to task dependencies.
- The makefile syntax is arcane and hard to work with.
  - And XML syntax isn't?

.....

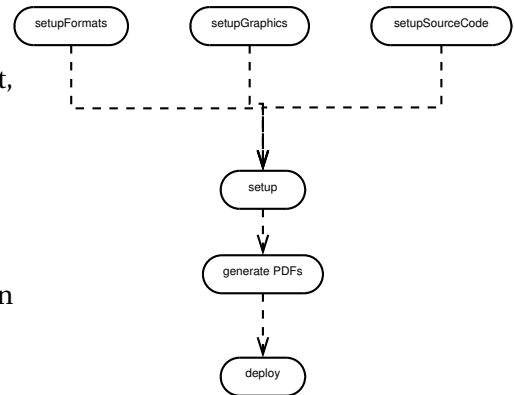
## 1 The ant Command

### ant

- **ant** looks for its instructions in a file named, by default, `build.xml`
- The **ant** command can name any target to be built, e.g.,

`ant setup`

- If no target is given, **ant** builds a target explicitly listed in `build.xml` as a default for the project.



.....

### ant Options

Some useful options:

- k**, **-keep-going** “Keep going.” Don’t stop the build at the first failure, but continue building any required targets that do not depend on the one whose construction has failed.
- f filename** Use *filename* instead of the default `build.xml`. Also `-file` or `-buildfile`
- Dproperty=value** Sets a property (similar to **make**’s variables)

.....

## 2 Build Files

### Build Files

The commandant build file is an XML file.

- The build file describes a project.
  - The project has a name and a default target.

```
<project name="382Website" default="deploy">
  <description>
    Extract Metadata Extractor – top level
  </description>
  ⋮
</project>
```

.....

## 2.1 Targets

### Targets

At its heart, a build file is a collection of *targets*.

- A target is an XML element and, as attributes, has a name and, optionally,
  - a list of dependencies
  - a condition
  - a human-readable description
- The target can contain multiple *tasks*, which contain the actual “commands” to get things done.

**ant** targets correspond, roughly, to **make**'s “artificial targets”.

.....

### Example of Targets

```
<project name="JavaBuild" default="deploy"> ❶
  <description>
    Example of a simple project build
  </description>
  <target name="compile" description="Compile src/.../*.java into bin/"> ❷
    <mkdir dir="bin" /> ❸
    <javac srcdir="src" destdir="bin"
      debug="true" includeantruntime="false"/>
    <echo>compiled </echo>
  </target>
  <target name="unittest" depends="compile" unless="test.skip"> ❹
    <mkdir dir="test-reports" />
    <junit printsummary="on" haltonfailure="true"
      fork="true" forkmode="perTest">
      <formatter type="plain" />
      <batchtest todir="test-reports">
    <fileset dir="bin">
      <include name="**/Test*.class" />
      <exclude name="**/Test*$.class" />
    </fileset>
    </batchtest>
```

## Task Dependencies: ant

---

```
    </junit>
  </target>

  <target name="deploy" depends="unittest" description="Create project's Jar file">
    <jar destfile="myProject.jar">
      <fileset dir="bin"/>
    </jar>
  </target>
</project>
```

- ❶ The project has a name and default target
- ❷ A basic target. It is named “compile” and has a description (which may be picked up by some IDEs)
- ❸ This target has 3 tasks. It creates a directory, compiles Java source code, and prints a message when completed.
  - The fact that the tag names resemble familiar commands is intended as self-documentation, but is not otherwise significant.
  - The tag names actually map to Java class names that implement the task.
- ❹ This target illustrates both a dependency and a condition.

The tasks within this target would not be executed if I invoked **ant** like this:

```
ant -Dtest.skip=1
```

However, the `unittest` task would still be considered to have succeeded, in the sense that tasks that depend on it would be allowed to run.

.....

### Task versus File Dependencies

**ant** targets correspond, roughly, to **make**'s “artificial targets”.

So this build file

```
<project name="JavaBuild" default="deploy"> ❶
  <description>
    Example of a simple project build
  </description>

  <target name="compile" description="Compile src/.../*.java into bin/"> ❷
```

## Task Dependencies: ant

---

```
<mkdir dir="bin" />
<javac srcdir="src" destdir="bin"
  debug="true" includeantruntime="false"/>
<echo>compiled </echo>
</target>

<target name="unittest" depends="compile" unless="test.skip">
  <mkdir dir="test-reports" />
  <junit printsummary="on" haltonfailure="true"
    fork="true" forkmode="perTest">
    <formatter type="plain" />
    <batchtest todir="test-reports">
      <fileset dir="bin">
        <include name="**/Test*.class" />
        <exclude name="**/Test*$.class" />
      </fileset>
    </batchtest>
  </junit>
</target>

<target name="deploy" depends="unittest" description="Create project's Jar file">
  <jar destfile="myProject.jar">
    <fileset dir="bin"/>
  </jar>
</target>
</project>
```

is roughly equivalent to this makefile

```
JAVAFILESsrc=$(shell find src/ -name '*.java')
JAVAFILES=$(JAVAFILESsrc:src/%=)
CLASSFILES=$(JAVAFILES:%.java=%.class)
TESTFILES=$(shell find src/ -name 'Test*.java')
TESTS=$(TESTFILES:src/%.java=)

deploy: unittest
  cd bin; jar cvf myProject.jar `find . -name '*.class'`

unittest: build
  cd bin; for test in $(TESTS); do
    java $$test;
```

## Task Dependencies: ant

---

```
done
build:
  cd src; javac -d ../bin -g $(JAVAFILES)
```

though a “real” makefile author would probably write this:

```
JAVAFILESsrc=$(shell find src/ -name '*.java')
JAVAFILES=$(JAVAFILESsrc:src/=%)
CLASSFILES=$(JAVAFILES:%.java=%.class)
TESTFILES=$(shell find src/ -name 'Test*.java')
TESTS=$(TESTFILES:src/%.java=%)

deploy: myProject.jar
unittest: testReport.txt
build: $(CLASSFILES)

myProject.jar: testReport.txt $(CLASSFILES)
  cd bin; jar cvf myProject.jar `find . -name '*.class'`

testReport.txt: $(CLASSFILES)
  -rm testReport.txt
  cd bin; for test in $(TESTS); do
    java $$test >> testReport.txt;
  done

bin/%.class: src/%.java
  cd src; javac -d ../bin -g $*.java
```

.....

### Make Efficiency

If we do

```
make
make
```

The second command does not actually perform any steps.

.....

### Ant Efficiency

What happens if we do

```
ant
ant -Dskip.test=1
```

- Each of the tasks *is* executed, but
  - The javac task knows not to re-compile Java files with up-to-date class files
  - The javac task knows not to update Jar files that are newer than all of the files being added.
- So some level of incremental behavior gets built into many of the individual tasks.
- If we remove the -Dskip.test=1, however, the tests will be re-run.

.....

## 2.2 Properties

### Properties

Properties are named string values.

- Can be set from the command line or via a <property and a few other tasks
- Accessed as `${propertyName}`
- Properties are immutable: once set, attempts to re-assign their values are ignored
- By convention, properties names are grouped into faux hierarchies with ''
  - e.g., compile.src, compile.dest, compile.options

.....

### The <property Task

Two basic modes:

- `<property name="compile.options" value="-g -O1"/>`  
Sets this property to “-g -O1”
- `<property name="compile.src" location="src/main/java"/>`  
Sets this property to the *absolute path* to the directory/file named.
- The / and \ characters are changed as necessary to conform to the OS on which **ant** is being run

.....



### Additional <property Variants

- <property file="project.default.properties"/>

Loads property values from a file, written as a series of *property=value* lines

```
courseName=CS795
baseurl=https://secweb.cs.odu.edu/~zeil/cs795SD/s13
homeurl=https://secweb.cs.odu.edu/~zeil/cs795SD/s13/Directory/topics.html
email=zeil@cs.odu.edu
```

- <property environment="env"/>

Copies the OS environment variables into the build state, prefaced by the indicated prefix

- e.g., \${env.PATH}

.....

## 2.3 File Sets and Lists

### File Sets and Lists

- A file set is a collection of existing files
  - can be specified using wild cards
- A file list is a collection of files that may or may not exist
  - Must be specified explicitly without wild cards

.....

### File Sets

```
<fileset file="src/main.cpp"/>
<fileset dir="src"
  includes="main.cpp utility.h utility.cpp"/>
<fileset dir="src" includes="*.cpp,*.h"/>
```

- More commonly seen as a nested form

```
<fileset id="unitTests" dir="bin">
  <include name="**/Test*.class"/>
  <exclude name="**/*$.class"/>
</fileset>
```

## Task Dependencies: ant

---

- The id in the prior example allows later references:

```
<fileset refid="unitTests" />
```

.....

### File Lists

```
<filelist dir="src"
  files="main.cpp utilities.h utilities.cpp" />
```

- Can also use id or refid attributes

.....

### Mappers

- Allow for a transformation of file names
- Some commands use a file set to describe inputs, then a mapper to describe outputs

```
<listset dir="src" includes="*.cpp" />
<globmapper from="*.cpp" to="*.o" />
```

would map each file in src/\*.cpp to a corresponding .o file

```
<listset dir="bin" includes="**/Test*.java" />
<packagemapper from="*.class" to="*" />
```

would map a compiled unit test file project/package/TestADT.class to project.package.TestADT

- There are several other mappers as well

.....

### Selectors

Selectors provide more options for selecting file than simple include/exclude based on the names.

```
<fileset id="unitTestSrc" dir="src">
  <include name="**/Test*.java" />
  <contains text="@Test" casesensitive="no" />
</fileset >
```

(Our previous examples assumed that unit tests would be identified by file name. Here we look instead for the JUnit4 @Test annotation.)

- Other selectors replicate several of the tests from the classic Unix **find** command

.....



## 2.4 Path Sets

### Path Sets

Used to specify a sequence of paths, usually to be searched.

```
<classpath>
  <pathelement path="{env.CLASSPATH}"/>
  <fileset dir="target/classes">
    <include name="**/*.class"/>
  </fileset>
  <filelist refid="third-party_jars"/>
</classpath>
```

.....

### Referencing Path Sets

- For reason unclear to me, you cannot name classpaths and re-use them directly, but must do it this way

```
<path name="test.compile.classpath">
  <pathelement path="{env.CLASSPATH}"/>
  <fileset dir="target/classes">
    <include name="**/*.class"/>
  </fileset>
  <filelist refid="third-party_jars"/>
</path>
:
<classpath refid="test.compile.classpath"/>
```

.....

## 2.5 Filters

### Filters

Filters are used to modify the outputs of some commands by performing various substitutions:

```
<copy file = "../.. / templates / @ { format } . tex "
      tofile = "$ { doc } - @ { format } . ltx ">
  <filterset >
    <filter token = "doc" value = "$ { doc } "/>
    <filter token = "relPath" value = "$ { relPath } "/>
    <filter token = "format" value = "@ { format } "/>
  </filterset >
</copy >
```

A filter set replaces tokens like @doc@ by a string, in this case the value of the property \${doc}

.....

### Filter Chains

Filter chains offer a variety of more powerful options, e.g.,

```
<loadfile property="doctitle" srcfile="${doc}.info.tex">
  <filterchain >
    <linecontains>
      <contains value="\title {"/>
    </linecontains>
    <tokenfilter >
      <replaceregex pattern=" *\title [{}([{}]*)[{}]"
                   replace="\1"/>
    </tokenfilter >
  </filterchain >
</loadfile >
```

- `loadfile` loads an entire file into a property
- The filter extracts the contents of a LaTeX `\title{...}` command

.....

## 2.6 Tasks

### Tasks

The Ant Manual has a good breakdown on these.

- Consistent with their XML structure, tasks can be parameterized via attributes or nested XML attributes
  - Sometimes you can do the same thing either way.
- Look at:
  - File tasks: `copy`, `delete`, `mkdir`, `move`, `fixCrLf`, `sync`
  - Compile tasks: `javac`, `depend`
  - Archive, documentation, testing tasks
  - Execution tasks: `java`, `exec`, `apply`

.....

### Extending Ant

- Ant has a built-in macro capability
- More powerful extension is accomplished by adding Java classes, mapped onto task names:

```
<project name="code2html" default="build">

  <taskdef classpath="JFlex.jar"
           classname="JFlex.anttask.JFlexTask"
           name="jflex" />
  :

  <target name="generateSource">
    <mkdir dir="src/main/java"/>
    <jflex file="src/main/jflex/code2html.flex"
          destdir="src/main/java"/>
    <jflex file="src/main/jflex/code2tex.flex"
          destdir="src/main/java"/>
    :
  </target>
</project>
```

.....

### Finding Extensions

- Many Java-oriented tools (e.g. JFlex) come with an ant task as part of the package.
- Other are contributed by users of the tool, (e.g. LaTeX)
- Some general-purpose Ant libraries.  
e.g., antcontrib adds

- C/C++ compilation
- If and For-loop
- outofdate (a make-like file dependency wrapper)
- enhanced property tasks (e.g., URL encoding)

.....

### 3 Case Studies

#### 3.1 Code Annotation

##### Code Annotation Tool

The steps involved in building this tool are:

1. Run the program **jflex** on each file in `src/main/jflex`, generating a pair of `.java` files that get placed in `src/main/java`
2. Compile the Java files in `src/main/java`, placing the results in `target/classes`
3. Compile the Java files in `src/test/java` (using the `target/classes` compilation results, placing the results in `target/test-classes`).
4. Run the JUnit tests in `target/test-classes`.
5. If all tests pass, package the compiled classes in `target/classes` into a `.jar` file.

.....

##### The Code Annotation Tool Build

```
<project name="code2html" default="build">
  <record name="ant.log" action="start" append="false" /> ❶
  <taskdef classpath="JFlex.jar" classname="JFlex.anttask.JFlexTask"
    name="jflex" /> ❷
  <echo>loading build-${os.name}.paths</echo>
  <include file="build-${os.name}.paths"/> ❸
  <target name="generateSource"> ❹
    <mkdir dir="src/main/java"/>
    <jflex file="src/main/jflex/code2html.flex"
      destdir="src/main/java"/>
    <jflex file="src/main/jflex/code2tex.flex"
      destdir="src/main/java"/>
    <jflex file="src/main/jflex/list2html.flex"
      destdir="src/main/java"/>
    <jflex file="src/main/jflex/list2tex.flex"
```

## Task Dependencies: ant

---

```
        destdir="src/main/java"/>
</target>

<target name="compile" depends="generateSource"> ⑤
    <mkdir dir="target/classes"/>
    <javac srcdir="src/main/java" destdir="target/classes"
        source="1.6" includeantruntime="false"/>
</target>

<target name="compile-tests" depends="compile"> ⑥
    <mkdir dir="target/test-classes"/>
    <javac srcdir="src/test/java" destdir="target/test-classes"
        source="1.6" includeantruntime="false">
        <classpath refid="testCompilationPath"/>
    </javac>
</target>

<target name="test" depends="compile-tests"> ⑦
    <mkdir dir="target/test-results/details"/>
    <junit printsummary="yes"
        haltonfailure="yes"
        >
    <formatter type="xml"/>
    <batchtest todir="target/test-results/details">
        <fileset dir="target/test-classes">
            <include name="**/*Test*.class"/>
        </fileset>
    </batchtest>
    <classpath refid="testExecutionPath"/>
</junit>
<junitreport todir="target/test-results">
    <fileset dir="target/test-results/details">
        <include name="TEST-*.xml"/>
    </fileset>
</junitreport>
</target>

<target name="build" depends="test"> ⑧
    <jar destfile="codeAnnotation.jar" basedir="target/classes">
```

## Task Dependencies: ant

---

```
<manifest>
  <attribute name="Main-Class"
    value="edu.odu.cs.code2html.Code2HTML"/>
</manifest>
</jar>
</target>

<target name="clean">
  <delete dir="target"/>
</target>
</project>
```

This is a fairly “stock” build for a Java project.

Nonetheless, there are multiple steps that would not be handled by typical IDE build managers.

- ❶ Not all tasks need to be within targets.

Properties are usually not, but this is an example of a more “active” task - it copies the **ant** output into a log file.

- ❷ Establishes the `<jflex>` tag, provided in `JFlex.jar`

- ❸ Includes an XML file of additional **ant** commands.

- The name of the file loaded includes the operating system name `${os.name}`, so this allows for customization.

```
<project>
  <path id="testCompilationPath">
    <pathelement location="/usr/share/java/junit4.jar"/>
    <pathelement location="/usr/share/java/hamcrest-library.jar"/>
    <pathelement path="target/classes"/>
  </path>

  <path id="testExecutionPath">
    <pathelement location="/usr/share/java/junit4.jar"/>
    <pathelement location="/usr/share/java/hamcrest-library.jar"/>
    <pathelement path="target/classes"/>
    <pathelement path="target/test-classes"/>
  </path>
</project>
```



```
<project>
  <path id="testCompilationPath">
    <pathelement location="./junit4.jar"/>
    <pathelement location="hamcrest-library.jar"/>
    <pathelement path="target/classes"/>
  </path>

  <path id="testExecutionPath">
    <pathelement location="./junit4.jar"/>
    <pathelement location="hamcrest-library.jar"/>
    <pathelement path="target/classes"/>
    <pathelement path="target/test-classes"/>
  </path>
</project>
```

- **ant** has various tasks for including other files into a build
  - \* `loadfile`: loads properties in plain-text form
  - \* `include`: shown here, loads XML **ant** instructions
  - \* `import`: Similar to `include`, but `import` allows the imported targets to be overridden by the importer. `include` does notThe manual page for `import` has a good example showing the difference.
- ④ In this target we use the `flex` tag which was loaded as an extension earlier.  
This creates several Java files in `src/main/java`.
- ⑤ All Java source in `src/main/java` is compiled, placing the resulting `.class` files in `target/classes`
  - Note the classpath, which was included from our OS-dependent path files.
- ⑥ All Java source in `src/test/java` is compiled, placing the resulting `.class` files in `target/test-classes`
  - The destination directory is distinct to make it easier to later package up the “real” deliverable classes, omitting the test drivers.
  - The classpath here is different, because it must include both the code we have already compiled and the JUnit package.
- ⑦ Run the tests and generate a basic summary report.
- ⑧ Package up the application into a Jar file, selecting one of the 4 executables as the default to be run when the jar file is launched by double-clicking or via `java -jar`
- ⑨ A typical clean-up task,

- Note that this target does not depend on the others. It is intended to be invoked separately.
- The task is made simpler by keeping the compilation binaries in a separate directory.
- In larger projects, I might have done the same with the JFlex-generated sources, so that they would be cleaned as well.

.....

### 3.2 Projects with Multiple Sub-Projects

#### Managing Subprojects with ant

Typical key ideas:

- Gather common structures into a build file that can be included or imported by each subproject.
- Create a top-level build file that
  - Performs project-wide initialization
  - Distributes common targets to individual subproject builds

.....

#### A Top-Level Build

```
<project name="course" default="build">
  <record name="ant.log" action="start" append="false" />
  <!-- Invoke this build file as
    ant
    - to build the desired documents
    ant clean
    - to clean out all generated files
  -->
  <property file="course.properties"/>
  <macrodef name="iterate" ❶
    <attribute name="target"/>
    <sequential>
      <subant target="@{target}">
```

## Task Dependencies: ant

---

```
<fileset dir=".">
  <include name="*/**/build.xml"/>
  <exclude name="templates/**"/>
</fileset>
</subant>
</sequential>
</macrodef>

<dependset>
  <srcfileset file="course.properties"/>
  <targetfileset file="course.info.tex"/>
</dependset>
<available property="course.info.tex.exists"
  file="course.info.tex"/>

<target name="courseSetup" unless="course.info.tex.exists">
  <copy file="templates/course.info.tex" toFile="course.info.tex">
    <filterset>
      <filtersfile file="course.properties"/>
    </filterset>
  </copy>
  <replace file="course.info.tex" token="~" value="%7E"/>
  <echo file="course.info.tex" append="true">
    % This file is generated automatically from course.properties
  </echo>
  <echo file="course.info.tex" append="true">
    % Do not edit. Any changes are likely to be overwritten.
  </echo>
</target>

<target name="emailSetup">
  <copy todir=".">
    <fileset dir="." file="templates/sendEmail.html"/>
    <filterset>
      <filtersfile file="course.properties"/>
    </filterset>
  </copy>
</target>
```

## Task Dependencies: ant

```
<target name="setup" depends="courseSetup,emailSetup" ⑤
  description="Prepare all source files prior to running LaTeX"
  >
  <iterate target="setup"/>
</target>

<target name="build" depends="courseSetup,emailSetup" ⑥
  description="Generate all documents">
  <iterate target="build"/>
</target>

<target name="zip" depends="build" ⑦
  description="Generate all documents and prepare a zip file that can be uploaded to and unpa
  >
  <tstamp>
  <format property="today" pattern="yyyy-MM-dd_hh-mm"/>
  </tstamp>
  <zip destfile="${courseName}_${today}.zip" level="9">
  <fileset dir=".">
  <exclude name="**/*~"/>
  <exclude name="**/*.aux"/>
  <exclude name="**/*.fdb_latexmk"/>
  <exclude name="**/*.fls"/>
  <exclude name="**/*.log"/>
  <exclude name="**/*.out"/>
  <exclude name="**/*.toc"/>
  <exclude name="**/*.nav"/>
  <exclude name="**/*.snm"/>
  <exclude name="**/*.vrb"/>
  <exclude name="**/*.tex"/>
  <exclude name="**/*.ltx"/>
  </fileset>
  </zip>
</target>

<target name="deploy" depends="build" ⑧
  description="Generate all documents and sync with the deployment directory (usually a websi
  >
  <sync todir="${deploymentDestination}"
  includeEmptyDirs="true" granularity="2000">
```

## Task Dependencies: ant

```
    <fileset dir=".">
    <exclude name="**/*~"/>
    <exclude name="**/*.aux"/>
    <exclude name="**/*.fdb_latexmk"/>
    <exclude name="**/*.fls"/>
    <exclude name="**/*.log"/>
    <exclude name="**/*.out"/>
    <exclude name="**/*.toc"/>
    <exclude name="**/*.nav"/>
    <exclude name="**/*.snm"/>
    <exclude name="**/*.vrb"/>
    <exclude name="**/*.tex"/>
    <exclude name="**/*.ltx"/>
    </fileset>
    <preserveintarget>
    <include name="**/*.ht*"/>
    </preserveintarget>
  </sync>
</target>

<target name="clean">
  <iterate target="clean"/>
</target>

<target name="cleaner">
  <iterate target="cleaner"/>
</target>

<target name="cleanest">
  <iterate target="cleanest"/>
</target>
</project>
```

### ❶ A macro declaration

- This macro takes a parameter, “target”
- It uses the task `subant` to invoke **ant** recursively on that target (`@target`) for each `build.xml` file that it finds in a subdirectory (at any depth)

\* except for subdirectories of templates

- ② This task is used for simulate **make**-like file dependencies. It deletes the target files if any of them are older than any of the source files.
  - Still a bit more coarse-grained than **make**
- ③ Sets a property to true/false depending on whether a file exists.
  - The file is the same one used as the target of the prior dependset
- ④ Part of the project-wide setup, skipped if the earlier dependset decided that the target file was already up-to-date.
- ⑤ The main target for project-wide setup.
- ⑥ The “build” target is issued to individual sub-projects using the earlier macro.
  - This is the default target.
  - Once we are satisfied with a build, we can issue a new **ant** command to perform either of the next two options.
- ⑦ After all the builds are done, we could build a zip file of the results.
  - The date and time of the build is included in the zip file name.
- ⑧ Or, instead of the zip file, we might sync the results with another directory (a website).

.....

### 3.3 Extend or Exec?

#### Extend or Exec?

As we move further from common Java project structures, the problem arises of how to issue commands for which we have no readily available **ant** task.

- Write our own task as a java class
  - A lot of work, particularly for a one-off
- If the desired command is actually a Java program, use the java task to launch it.
  - Can be reasonably portable
  - But some **ant** purists still find this objectionable
- Use the exec or apply tasks to simply run a command or non-Java program

## Task Dependencies: ant

---

- Portability becomes much more of an issue

.....

### Mitigating Factors

Although the community has contributed numerous extension tasks,

- Some have steep learning curves and/or tricky setup
  - e.g., cc task from ant-contrib
- Others may be incomplete or buggy

This can drive a project to use exec/apply even if **tasks** purportedly exist

.....

### Generating Course Website PDFs

Each document is a subproject with a build file like this:

```
<project name="docs" default="build">
  <import file="../../commonBuild.xml"/>
  <target name="makeSources" depends="properties">
    <docformat format="slides"/>
    <docformat format="web"/>
    <docformat format="printable"/>
    <docformat format="10-4x3"/>
    <docformat format="10-16x10"/>
    <docformat format="7-4x3"/>
    <docformat format="7-16x10"/>
  </target>
</project>
```

The “good stuff” is in the included commonBuild.xml

```
<project>
  <condition property="windowsos">
    <os family="windows" />
  </condition>
```



## Task Dependencies: ant

---

```
<target name="logging" unless="windowsos">
  <record name="ant.log" action="start" append="false" />
</target>

<property environment="env"/>

<basename property="doc" file="${basedir}"/>
<property name="baseParent" location=".."/>
<basename property="relPath" file="${baseParent}"/>

<property file="../../course.properties"/>

<macrodef name="docformat">
  <attribute name="format"/>
  <sequential>
<copy file="../../templates/@{format}.tex"
  tofile="${doc}-@{format}.ltx">
  <filterset>
    <filter token="doc" value="${doc}"/>
    <filter token="relPath" value="${relPath}"/>
    <filter token="format" value="@{format}"/>
  </filterset>
</copy>
<replaceregexp file="index.html"
  match="[\.]vis@{format} [{}display:none" replace=".vis@{format} {table-row}/>
<copy file="${doc}-@{format}.ltx" tofile="${doc}-@{format}.main.tex"/>
  </sequential>
</macrodef>

<available property="dblatex-is-installed"
  file="dblatex" filepath="${env.PATH}"/>

<target name="checkDB" depends="logging" if="dblatex-is-installed">
  <dependset>
<srcfileset file="${doc}.dbk"/>
<targetfileset file="${doc}.content.tex"/>
  </dependset>
</target>
```



## Task Dependencies: ant

---

```
<target name="checkForContent" depends="checkDB">
  <available property="content.exists" file="${doc}.content.tex"/>
</target>

<target name="convertDB" depends="checkForContent" unless="content.exists">
  <apply executable="dblatex" parallel="false">
    <arg value="--style=simple"/>
    <arg value="--type=tex"/>
    <arg value="--xsl-user=../../templates/dblatex-sjz.xsl"/>
    <fileset file="${doc}.dbk"/>
  </apply>
  <replace file="${doc}.dbk.tex" token="language=cpp" value="language=C++"/>
  <replace file="${doc}.dbk.tex" token="cyrcharcyrie{}" value="$epsilon$"/>
  <apply executable="csplit" parallel="false" addsourcefile="false">
    <arg value="${doc}.dbk.tex"/>
    <arg value="/title"/>
    <arg value="/^%/"/>
    <arg value="/mainmatter/+1"/>
    <arg value="/end{document}"/>
  </apply>
  <fileset file="${doc}.dbk.tex"/>
  <move file="xx01" tofile="${doc}.info.tex"/>
  <move file="xx03" tofile="${doc}.content.tex"/>
  <delete>
    <fileset dir="." includes="xx0*" />
    <fileset file="${doc}.dbk.tex" />
  </delete>
</target>

<target name="makeGraphics" depends="logging,epsGraphics,gifGraphics">
  <available property="epstopdf-is-installed"
    file="epstopdf" filepath="${env.PATH}"/>

  <target name="epsGraphics" depends="logging,diaGraphics,figGraphics"
    if="epstopdf-is-installed">
    <apply executable="epstopdf" parallel="false">
      <fileset dir="." includes="*.eps"/>
      <globmapper from="*.eps" to="*.pdf"/>
    </apply>
  </target>
</target>
```

## Task Dependencies: ant

---

```
    </apply>
    <dependset>
<srcfileset dir="." includes="*.eps"/>
<targetfileset dir="." includes="${doc}/*.pdf"/>
    </dependset>
</target>

<available property="dia-is-installed"
    file="dia" filepath="${env.PATH}"/>

<target name="diaGraphics" depends="logging" if="dia-is-installed">
    <apply executable="dia" parallel="false">
<arg value="-t"/>
<arg value="eps-builtin"/>
<fileset dir="." includes="*.dia"/>
<globmapper from="*.dia" to="*.eps"/>
    </apply>
</target>

<available property="fig2dev-is-installed"
    file="fig2dev" filepath="${env.PATH}"/>

<target name="figGraphics" depends="logging" if="fig2dev-is-installed">
    <apply executable="fig2dev" parallel="false">
<arg value="-L"/>
<arg value="eps"/>
<srcfile/>
<targetfile/>
<fileset dir="." includes="*.fig"/>
<globmapper from="*.fig" to="*.eps"/>
    </apply>
</target>

<available property="convert-is-installed"
    file="convert" filepath="${env.PATH}"/>

<uptodate property="gifConversion.notRequired">
    <srcfiles dir=".">
<include name="*.gif"/>
```

## Task Dependencies: ant

---

```
    </srcfiles>
    <globmapper from="*.gif" to="*.png"/>
  </uptodate>

  <target name="gifGraphics" depends="logging"
    if="convert-is-installed" unless="gifConversion.notRequired">
    <apply executable="convert" parallel="false" relative="true">
    <srcfile/>
    <targetfile/>
    <fileset dir="${basedir}" includes="*.gif"/>
    <globmapper from="*.gif" to="*.png"/>
    </apply>
  </target>

  <available property="ext-file-exists"
    file="${doc}.ext.tex" filepath="."/>

  <uptodate property="makeSourceCodeHTML.notRequired">
    <srcfiles dir=".">
    <include name="*.h"/>
    <include name="*.cpp"/>
    <include name="*.java"/>
    <include name="*.listing"/>
    </srcfiles>
    <globmapper from="*" to="*.html"/>
  </uptodate>

  <uptodate property="makeSourceCodeTex.notRequired">
    <srcfiles dir=".">
    <include name="*.h"/>
    <include name="*.cpp"/>
    <include name="*.java"/>
    <include name="*.listing"/>
    </srcfiles>
    <globmapper from="*" to="*.tex"/>
  </uptodate>

  <target name="makeSourceCodeTex" depends="logging"
    unless="makeSourceCodeTex.notRequired">
    <apply executable="java">
    <arg value="-cp"/>
```

## Task Dependencies: ant

---

```
<arg value="../../templates/codeAnnotation/codeAnnotation.jar"/>
<arg value="edu.odu.cs.codeAnnotation.Code2TeX"/>
<fileset dir=". ">
  <include name="**/*.h"/>
  <include name="**/*.cpp"/>
  <include name="**/*.java"/>
</fileset>
<globmapper from="*" to="*.tex"/>
  </apply>
  <apply executable="java">
<arg value="-cp"/>
<arg value="../../templates/codeAnnotation/codeAnnotation.jar"/>
<arg value="edu.odu.cs.codeAnnotation.Listing2TeX"/>
<fileset dir=". ">
  <include name="**/*.listing"/>
</fileset>
<globmapper from="*" to="*.tex"/>
  </apply>
</target>

<target name="makeSourceCodeHTML" depends="logging"
  unless="makeSourceCodeHTML.notRequired">
  <apply executable="java">
<arg value="-cp"/>
<arg value="../../templates/codeAnnotation/codeAnnotation.jar"/>
<arg value="edu.odu.cs.codeAnnotation.Code2HTML"/>
<arg value="-footer"/>
<arg value="&lt; footer/&gt;"/>
<fileset dir=". ">
  <include name="**/*.h"/>
  <include name="**/*.cpp"/>
  <include name="**/*.java"/>
</fileset>
<globmapper from="*" to="*.html.xml"/>
  </apply>
  <apply executable="java">
<arg value="-cp"/>
<arg value="../../templates/codeAnnotation/codeAnnotation.jar"/>
<arg value="edu.odu.cs.codeAnnotation.Listing2HTML"/>
<fileset dir=". ">
```

## Task Dependencies: ant

---

```
    <include name="**/*.listing"/>
  </fileset>
  <globmapper from="*" to="*.html.xml"/>
  </apply>
</target>

<target name="makeSourceCode" depends="makeSourceCodeHTML, makeSourceCodeTex"/>

<target name="makeExt" unless="ext-file-exists">
  <echo file="${doc}.ext.tex">% External documents referenced from here</echo>
</target>

<target name="properties" depends="logging,convertDB">
  <loadfile property="doctitle" srcfile="${doc}.info.tex">
<filterchain>
  <linecontains>
    <contains value="title{"/>
  </linecontains>
  <tokenfilter>
    <replaceregex pattern=" *title[{}([{}]*)[{}]"
    replace="1"/>
  </tokenfilter>
</filterchain>
  </loadfile>
  <echo file="title.xml">&lt;title doc="${doc}"&gt;${doctitle}&lt;/title&gt;</echo>
  <copy file="../../templates/index.html" tofile="index.html" overwrite="true">
<filterset>
  <filter token="doc" value="${doc}"/>
  <filter token="title" value="${doctitle}"/>
</filterset>
  </copy>
</target>

<target name="makeHTML" depends="makeSources">
  <dependset>
<srcfileset dir="." includes="*.html.xml"/>
<targetfileset dir="." includes="*.html"/>
  </dependset>
```

## Task Dependencies: ant

---

```
<xslt style="../../templates/modifyHTML.xsl" destdir=".">
<fileset dir=".">
  <include name="*.html.xml"/>
</fileset>
<globmapper from="*.html.xml" to="*.html.tmp"/>
</xslt>
<copy todir=".">
<fileset dir='.' includes="*.html.tmp"/>
<globmapper from="*.html.tmp" to="*.html"/>
<filterset>
  <filtersfile file="../../course.properties"/>
</filterset>
</copy>
<delete>
<fileset dir='.' includes="*.html.tmp"/>
</delete>
</target>

<target name="setup" depends="convertDB,makeHTML,makeGraphics,makeSourceCode,makeExt"
  description="Prepare all source files prior to running LaTeX"
  >
  <replaceregexp file="index.html" match="@[^@]+@" replace="none" flags="g"/>
</target>

<dependset>
  <srcfileset dir=".">
    <include name="*.tex"/>
    <include name="*.ltx"/>
  </srcfileset>
  <targetfileset dir="." includes="${doc}/*.pdf"/>
</dependset>

<uptodate property="pdfGen.notRequired">
  <srcfiles dir=".">
    <include name="*.ltx"/>
  </srcfiles>
  <globmapper from="*.ltx" to="*.pdf"/>
</uptodate>
```

## Task Dependencies: ant

---

```
<target name="genPDFs" depends="setup" unless="pdfGen.notRequired">
  <apply executable="latexmk" parallel="false">
    <arg value="-pdf"/>
    <env key="TEXMFHOME" value="../../templates//"/>
    <fileset dir="." includes="*.ltx"/>
    <globmapper from="*.ltx" to="*.pdf"/>
  </apply>
</target>

<target name="slides" depends="setup"
  description="Force a (single) latex run on the slides format - used for debugging slide c
  >
  <exec executable="pdflatex">
    <arg value="${doc}-slides.ltx"/>
    <env key="TEXMFHOME" value="../../templates//"/>
  </exec>
</target>

<target name="printable" depends="setup"
  description="Force a (single) latex run on the printable format - used for debugging non-
  >
  <exec executable="pdflatex">
    <arg value="${doc}-printable.ltx"/>
    <env key="TEXMFHOME" value="../../templates//"/>
  </exec>
</target>

<target name="web" depends="setup"
  description="Force a (single) latex run on the web format - used for debugging non-slide
  >
  <exec executable="pdflatex">
    <arg value="${doc}-web.ltx"/>
    <env key="TEXMFHOME" value="../../templates//"/>
  </exec>
</target>

<target name="bookweb" depends="setup"
  description="Force a (single) latex run on the Book web format - used for debugging non-s
  >
```

## Task Dependencies: ant

---

```
    <exec executable="pdflatex">
<arg value="\${doc}-bookweb.ltx"/>
<env key="TEXMFHOME" value="../../templates//"/>
    </exec>
</target>

<target name="build" depends="genPDFs"
    description="Generate all documents">
</target>

<target name="deploy" depends="build"
    description="Generate all documents and sync with the deployment directory (usually a web
    >
    <sync todir="\${deploymentDestination}/\${relPath}/\${doc}"
    includeEmptyDirs="true" granularity="2000">
<fileset dir=".">
    <exclude name="**/*~"/>
    <exclude name="**/*.aux"/>
    <exclude name="**/*.fdb_latexmk"/>
    <exclude name="**/*.fls"/>
    <exclude name="**/*.log"/>
    <exclude name="**/*.out"/>
    <exclude name="**/*.toc"/>
    <exclude name="**/*.nav"/>
    <exclude name="**/*.snm"/>
    <exclude name="**/*.vrb"/>
    <exclude name="**/*.tex"/>
    <exclude name="**/*.ltx"/>
</fileset>
<preserveintarget>
    <include name="**/*.ht*"/>
</preserveintarget>
    </sync>
</target>

<target name="clean"
    description="Remove temporary files created when running LaTeX"
    >
    <delete>
<fileset dir=".">
```



## Task Dependencies: ant

---

```
<include name="*.fdb_latexmk"/>
<include name="*.fls"/>
<include name="*.nav"/>
<include name="*.snm"/>
<include name="*.vrb"/>
<include name="*.log"/>
<include name="*.out"/>
<include name="*.h.html.xml"/>
<include name="*.cpp.html.xml"/>
<include name="*.java.html.xml"/>
</fileset>
</delete>
</target>

<target name="cleaner" depends="clean"
  description="Remove all files that can be easily regenerated, including the PDF documents"
  >
  <delete>
<fileset dir=".">
  <include name="index.html"/>
  <include name="${doc}/*.pdf"/>
  <include name="*.h.tex"/>
  <include name="*.cpp.tex"/>
  <include name="*.java.tex"/>
  <include name="*.h.html"/>
  <include name="*.cpp.html"/>
  <include name="*.java.html"/>
</fileset>
</delete>
</target>

<target name="cleanest" depends="cleaner"
  description="Remove all files that can be automatically regenerated"
  >
  <delete>
<fileset dir=".">
  <include name="${doc}/*.ltx"/>
  <include name="${doc}/*.main.tex"/>
  <include name="${doc}/*.aux"/>
  <include name="${doc}/*.toc"/>
```

```
</fileset>  
  </delete>  
</target>  
  
</project>
```

- Note use of exec to invoke the LaTeX document processor.

.....

## 4 Eclipse/Ant Integration

### Limitations of Eclipse Builder

- Cannot run code-preprocessing (e.g., JFlex)
- An Eclipse project is oriented towards producing a single output product (program, library, ...)
  - With C++ projects, a problem if you have a “real” product (e.g., a library) and a set of test drivers, each of which yields a distinct program executable.
  - Java projects have fewer problems (because executables don’t need separate processing), but what if you are planning to generate both
    - \* a binary distribution jar, and
    - \* a source distribution jar

.....

### Project Dependencies

Eclipse supports the idea of projects that depend on other projects, so you could do

- project1 produces the binary distribution jar
- project2 depends on project1 and produces a source distribution jar
  - These projects must reside together in a known relative path from one another
  - project2 is not automatically rebuild if project1 has changed
- Does not scale well.
  - For C++ are you going to have a distinct project for each test driver?

.....

### Eclipse/Ant Integration

Eclipse is generally **ant**-friendly.

- Drop a `build.xml` file into a project and Eclipse will recognize it.
  - Right-clicking on it will bring up options to run it, or to configure how to run it
    - \* including the selection of the target
    - \* some preference given to targets with descriptions
- Once **ant** has been run, the “Run Last Tool” button defaults to re-running it.
- But the default build is still Eclipse’s default build manager
  - For projects with elaborate classpaths, requires keeping both the Eclipse project description and the build file up-to-date and consistent.
  - Pre-compilation steps (e.g., tools that generate source code) are not re-run automatically when needed.

.....

### Eclipse Builders

Eclipse supports multiple, plug-able *builders*.

- Open Project Properties and go to “Builders”
  - In a typical java project, you have just the “Java Builder”
  - Click new to see options.  
In this case, select “Ant Builder”.
  - Fill in the main screen. Leave “Arguments” blank.
  - Go to the Targets tab. Select appropriate targets for  
Clean: Menu selection Project ->clean  
Manual build: What you want done after explicitly requesting a build  
Auto build: What you want done after a file has been saved/changed
- Return to the Builders list and uncheck the “Java Builder”

.....