Software Development Tools

COMP220

Seb Coope

Ant: Datatypes and Properties and Filesets

These slides are mainly based on “Java Development with Ant” - E. Hatcher & S.Loughran. Manning Publications, 2003

Ant Datatypes and Properties

Now, after getting started in previous lectures , we will consider Ant concepts in more detail .

Two most foundational concepts of Ant are

- *datatypes*
- *properties*

Ant datatypes overview

To build a typical Java project we mostly deal with

files and paths (such as classpaths).

This leads to Ant *datatypes* :

- fileset
- path
- and several others

Filesets overview

Fileset is a common entity to manipulate for such tasks as compiling, packaging, copying, deleting, and documenting.

Fileset is a group of files represented like

```
<fileset dir="src"
         includes="**/*.java"
         id="source.fileset"/>
```

- `dir` is mandatory attribute to denote a base (or root) directory of the *fileset* – here `src`. Files in the fileset can be found in a directory tree starting from this fileset base directory.
- `includes` attribute shows which files from this directory to include.
- `id` attribute is a reference which can be used later wherever this *fileset* is required to use (possibly repeatedly).

Filesets overview

For example, copying source code to another directory using the above `id="source.fileset"` could be done by `<copy>` task by using the `inverse' `refid` attribute as follows:

```
<copy todir="backup">
  <fileset refid="source.fileset"/>
</copy>
```

Paths overview

A *path* can be defined in a build file to be used for compilation with

`<javac>` task,

and reused for execution with

`<java>` task.

Classpath can be easily and tightly controlled by Ant.

This reduces CLASSPATH configuration problems, both for compilation and execution.

Examples will be presented later.

Properties overview

- Ant's property handling mechanism allows for *parameterizing* the build file by string-specified items.
- For example, we can change a build to use a different version of library (JAR file) by one command like this:

```
>ant -Dstruts.jar=/home/ant/newstruts/struts.jar
```

- In this example `struts.jar` after `–D` (no white space!) represents an *Ant property* (or *parameter*) with the *assigned value* "`/home/ant/newstruts/struts.jar`".
- Build file uses special syntax `${struts.jar}` to *refer* to this property.
- A key feature of an Ant property is its *immutability* :

  - once a property is set, it *resists change*.

Datatypes and Properties with `<javac>`

The `<javac>` task is an Ant's *version* of Java source *compilation command* `javac` with associated *switches*.

Let us *compare* Sun's

- JDK 1 `javac` command-line compiler switches to Ant's `<javac>` task attributes.

This is *shown in the following table*.

A comparison of `javac` command-line compiler switches to Ant's `<javac>` task attributes

JDK's `javac` switches	Ant's `<javac>` attributes
`-g` (generate all debugging info)	`debug="yes"`
`-g:none` (generate no debugging info)	`debug="no"`
`-verbose` (output messages about what the compiler is doing)	`verbose="true"`

(The `-g` option tells the compiler to include debugging information [in the compiled class] for future use by the debugger `jdb`

A comparison of `javac` command-line compiler switches to Ant's `<javac>` task attributes

JDK's `javac` switches	Ant's <javac> attributes/subelements

`-classpath`
(specify where to find referenced class files and libraries)
	`<classpath>`
	`  <pathelement`
	`  location="lib/some.jar"/>`
	`</classpath>`

`-sourcepath`
(specify where to find input source files)
	`<src path="src"/>`
	or
	`srcdir="src"`

`-d` (specify where to place generated class files)	`destdir="build/classes"`

Datatypes and Properties with <javac> (cont.)

Consider Java *compilation* with Ant utilizing Ant's *datatypes* (paths and filesets), *properties* and *references* to datatypes:

```
<javac destdir="${build.classes.dir}"
       debug="${build.debug}"
       srcdir="${src.dir}"
       includeAntRuntime="no">
   <include name="**/*.java"/>
   <classpath refid="compile.classpath"/>
</javac>
```

- `build.classes.dir, build.debug, src.dir` are *property names.*
- We *refer* to an Ant *property*, e.g. as in
  `srcdir="${src.dir}",`
  by using `${...}`.
- This will work if we have already somehow assigned *separately* a *value* of the property such as `src.dir` to be the *real* directory name `src`.
- Compare this with the *direct* reference to the *value* like in
  `srcdir="src"`
  where `src` is the *real* directory name.

Datatypes and Properties with <javac> (cont.)

It was assumed in the above example that *build file* also contains somewhere *path element* like

```
<path id="compile.classpath">
  <pathelement location="${lucene.jar}"/>
  <pathelement location="${jtidy.jar}"/>
</path>
```

refid="compile.classpath" in the previous slide *refers* to this path element because of id="compile.classpath".

It *shows where to find* two JAR files needed for the compilation.

See more on "${...}" notation on the next slide.

Properties with <javac>

- The "${...}" notation refers to an Ant *property* :
  - a mapping from a property name to a string value, referring to
    the compiling destination directory ${build.classes.dir},
    what debug mode to use ${build.debug},
    the source directory ${src.dir}, and
    JAR locations ${lucene.jar} and ${jtidy.jar}.
- Note that dot notation is used in
  naming properties, like above,
  or IDs, like compile.classpath

  This *imitates the natural language*.
- In particular, lucene.jar is considered here
  as the property name, *not* as the file name,
  however, they could coincide for the convenience.

Datatypes (paths and filesets) with <javac>

The subelement

<classpath refid="compile.classpath"/>

specifies a path by using a reference

indicating which previously defined path to use.

The previously defined path element (see Slide 12)

<path id="compile.classpath"> … </path>

- indicates which JAR files to use.

These JAR files are specified by the use of properties within the location attribute (see Slide 12).

The srcdir attribute of <javac> (see Slide 11)

- implicitly defines a fileset containing all files (to be compiled) in the specified directory tree.

The nested <include> of <javac> task specifies a pattern **/*.java

- this constrains the files to only Java source files (at any depth).

Paths in Ant

A *path*, or "path-like structure", is an ordered list of *pathelements*.

It is analogous to the Java CLASSPATH where each element in the list could be either

- a file
- or directory

separated by a delimiter.

Example in Ant:

```
<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/some.jar"/>
</classpath>
```

Or even shorter – for the single pathelement:

```
<classpath location="lib/some.jar"/>
```

- `location` attribute specifies a single file or directory.
- `path` attribute accepts colon- or semicolon-separated list of locations (like in the following slide), assuming this is the value of the property `${classpath}`.

Paths in Ant (cont.)

Example of a *list of locations*,
using `path` attribute (instead of `location`):

```
<classpath>
  <pathelement path="build/classes;lib/some.jar"/>
</classpath>
```

Or even shorter – for the single path element :

```
<classpath path="build/classes;lib/some.jar"/>
```

Both semicolon (`;`) and colon (`:`) above are allowed as separator.

Ant is "bi-slashed": use either forward-slash (/) or back-slash (\), regardless of operating system.

- Extremely *user friendly!*

Paths in Ant (cont.)

Paths can also include a *set of files*:

```
<classpath>
  <fileset dir= "lib">
    <include name="*.jar"/>
  </fileset>
</classpath>
```

Ant assumes *no order* within a `<fileset>`

Filesets

- Implicitly, all build processes such as compile, copy, delete, etc. operate on *sets of files.*
- Ant provides the fileset as *native datatype*.
  - It is difficult to imagine any useful build that does not use a fileset.
- Some tasks assume filesets *implicitly*,
- while other tasks support filesets *explicitly*.
- A fileset is a set of files rooted from a single directory.
- By default, a fileset specified with only a root directory will include *all* the files in that entire directory tree, including files in all sub-directories recursively (with some exceptions; see below on default exclude patterns).
- Filesets can appear in a build file either
  - inside tasks – the elements of targets, or
  - at the same level as targets.

Filesets (cont.)

Let us CREATE a new `copy.xml` file in `C:\Antbook\ch02\secondbuild`

by extending the build file `structured.xml` with *a new target* containing new `<copy>` *task:*

```
<target name="copy">
   <copy todir="new_build">
     <fileset dir="build"/>
   </copy>
</target>
```

First RUN `copy.xml`:

`ant -f copy.xml compile` (to do `init -> compile`)

Then RUN

`ant -f copy.xml copy` (to execute the above `<copy>` task)

Check what from `build` directory was copied into directory `new_build`.

Fileset examples

TRY to check – by creating appropriate build and other files – that

1. Fileset

```
<fileset dir="lib">
  <include name="*.jar"/>
</fileset>
```

includes all JAR files from the `lib` directory *non-recursively*, i .e. no subdirectories are considered.

2. Fileset

```
<fileset dir="test">
    <include name="**/*Test.java"/>
</fileset>
```

includes all `.java` files *in and below* the `test` directory that end with the string "`Test`".

Hint: Use `<copy>` task involving either the first or the second fileset to see which files are really copied.

Fileset examples (cont.)

3.

```
<fileset dir="web">
  <exclude name="**/*.jsp"/>
</fileset>
```

includes only non-JSP files in the `web` directory and below.

By default, `include` and `exclude` values are *case sensitive*.

But this *can be disabled* by specifying the attribute of `<fileset>`:

`casesensitive="false"`

`<include>` and `<exclude>` subelements in `<fileset>` serve as *patternsets*.

For example, `**/*.jsp` and `**/*Test.java` are *patterns*.

Fileset examples (cont.)

There is also a way to *abbreviate*

```
<fileset dir="web">
    <include name="**/*.jsp"/>
</fileset>
```

as

```
<fileset dir="web" includes="**/*.jsp"/>
```

by using *attribute* `includes`
instead of *subelement* `<include>`

Some default exclude patterns

Pattern	Typical program that creates and uses these files
`**/*~`	jEdit and other editors use this as previous version backup
`**/#*#`	editors
`**/.#*`	editors
`**/%*%`	editors

To turn off the automatic exclusion, use the `defaultexcludes` attribute:

```
<fileset dir="..." defaultexcludes="no">
```