<Build by/> Ant

Joerg Dolak

dolak@in.tum.de

What Is Ant?

- A build tool like make
- Open source
  - from the Apache Jakarta project
  - http://jakarta.apache.org/ant
- Implemented in Java
- Used to build many open source products
  - such as Tomcat and JDOM

Why Use Ant Instead of make?

- Ant is more portable
  - Ant only requires a Java VM (1.1 or higher)
  - make relies on OS specific commands to carry out it’s tasks
  - make can be used under Windows using Cygwin (a UNIX emulator) but that’s a big install! … ~37 meg.
- Ant targets are described in XML
  - make has a cryptic syntax
  - make relies proper use of tabs that is easy to get wrong
    - you can’t see them
- Ant is better for Java-specific tasks
  - faster than make since all tasks are run from a single VM
  - easier than make for some Java-specific tasks
    - such as generating javadoc, building JAR/WAR files and working with EJBs

How Does Ant Work?

- Ant commands (or tasks) are implemented by Java classes
  - Example: the testbench is exactly that, a set of custom commands
  - many are built-in
  - others come in optional JAR files
  - custom commands can be created
- Each project using Ant will have a build file
  - typically called build.xml since Ant looks for this by default
- Each build file is composed of targets
  - these correspond to common activities like compiling and running code
- Each target is composed of tasks
  - executed in sequence when the target is executed
  - like make, Ant targets can have dependencies
    - for example, modified source files must be compiled before the application can be run

use the `-buildfile` command-line option to specify a build file with a different name

How Does Ant Work? (Cont'd)

- Targets to be executed
  - can be specified on the command line when invoking Ant
  - if none are specified then the default target is executed
  - execution stops if an error is encountered
    so all requested targets may not be executed

A GUI front-end to Ant called **Antidote** is being developed.

Not necessarily a good thing

- Each target is only executed once
  - regardless of the number of other targets that depend on it
  - for example
    - the "test" and "deploy" targets both depend on "compile"
    - the "all" target depends on "test" and "deploy"
      but "compile" is only executed once when "all" is executed
- Some tasks are only executed when they need to be
  - for example, files that have not changed since the
    last time they were compiled are not recompiled

IDE Integration

- Ant can be integrated with the following Java IDEs
  - JBuilder
    - using AntRunner
  - SUNs Forte for Java
  - VisualAge for Java
- See the Ant User Manual for more details
  - in docs\manual\index.html

Typical Project Directory Structure

- project directory
  - holds files such as a README for the project and build.xml
  - classes directory
    - holds Java bytecode files
  - doc directory
    - holds project documentation
    - api directory
      - holds generated javadoc files
  - docroot directory
    - for web-based applications
    - holds files that must be copied to a special web server directory such as CSS, DTD, HTML, XML and XSL files
  - lib directory
    - holds files such as JAR and WAR files
  - src directory
    - holds Java source files

Sample Build File

(contains common targets used for servlet projects)

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Web App." default="deploy" basedir=".">

  <!-- Define global properties. -->
  <property name="appName" value="shopping"/>
  <property name="buildDir" value="classes"/>
  <property name="docDir" value="doc"/>
  <property name="docRoot" value="docroot"/>
  <property name="junit" value="/Java/JUnit/junit.jar"/>
  <property name="srcDir" value="src"/>
  <property name="tomcatHome" value="/Tomcat"/>
  <property name="servlet" value="${tomcatHome}/lib/servlet.jar"/>
  <property name="warFile" value="${appName}.war"/>
  <property name="xalan" value="/XML/Xalan/xalan.jar"/>
  <property name="xerces" value="/XML/Xalan/xerces.jar"/>
```

relative directory references are relative to this

target that is run when none are specified

Some of these are used to set “**classpath**” on the next page. Others are used in task parameters.

Where possible, use **UNIX-style paths** even under Windows.
This is not possible when Windows directories on drives other than the current drive must be specified.

Sample Build File (Cont'd)

```
<path id="classpath">
  <pathelement path="${buildDir}"/>
  <pathelement path="${xerces}"/>
  <pathelement path="${xalan}"/>
  <pathelement path="${servlet}"/>
  <pathelement path="${junit}"/>
</path>
```

used in the compile, javadoc and test targets

```
<target name="all" depends="test,javadoc,deploy"
 description="runs test, javadoc and deploy"/>
```

means that the test, javadoc and deploy targets must be executed before this target

doesn't have any tasks of its own; just executes other targets

Sample Build File (Cont'd)

```
<target name="clean" description="deletes all generated files">
  <delete dir="${buildDir}"/> <!-- generated by the prepare target -->
  <delete dir="${docDir}/api"/> <!-- generated by the javadoc target -->
  <delete>
    <fileset dir=".">
      <include name="${warFile}"/> <!-- generated by the war target -->
      <include name="TEST-*.txt"/> <!-- generated by the test target -->
    </fileset>
  </delete>
</target>

<target name="compile" depends="prepare"
 description="compiles source files">
  <javac srcdir="${srcDir}" destdir="${buildDir}" classpathref="classpath"/>
</target>

<target name="deploy" depends="war,undeploy"
 description="deploys the war file to Tomcat">
  <copy file="${warFile}" tofile="${tomcatHome}/webapps/${warFile}"/>
</target>
```

means that the prepare target must be executed before this target

compiles all files in or below srcDir that have no .class file or have been modified since their .class file was created; don't have to list specific file names as is common with make

classpath is defined on page 9

makes the servlet available through Tomcat; Tomcat won't expand the new war file unless the corresponding webapp subdirectory is missing

could use the FTP task to copy files to a remote location

Sample Build File (Cont'd)

```
<target name="dtd" description="generates a DTD for Ant build files">
  <antstructure output="build.dtd"/>
</target>
```

generates a DTD that is useful for learning the valid tasks and their parameters

```
<target name="javadoc" depends="compile"
 description="generates javadoc from all .java files">
  <delete dir="${docDir}/api"/>
  <mkdir dir="${docDir}/api"/>
  <javadoc sourcepath="${srcDir}" destdir="${docDir}/api"
   packagenames="tramp.application.*" classpathref="classpath"/>
</target>
```

generates javadoc for all .java files in or below srcDir.

classpath is defined on page 9

can't just use a single * here and can't use multiple *'s

```
<target name="prepare" description="creates output directories">
  <mkdir dir="${buildDir}"/>
  <mkdir dir="${docDir}"/>
</target>
```

creates directories needed by other targets if they don't already exist

Sample Build File (Cont'd)

```
<target name="test" depends="compile" description="runs all JUnit tests">
  <!-- Delete previous test logs. -->
  <delete>
    <fileset dir=".">
      <include name="TEST-*.txt"/> <!-- generated by the test target -->
    </fileset>
  </delete>

  <taskdef name="junit"
   classname="org.apache.tools.ant.taskdefs.optional.junit.JUnitTask"/>
  <junit printsummary="yes">
    <classpath refid="classpath"/>
    <batchtest>
      <fileset dir="${srcDir}"><include name="**/*Test.java"/></fileset>
      <formatter type="plain"/>
    </batchtest>
  </junit>
</target>
```

runs all JUnit tests in or below srcDir

junit.jar must be in the **CLASSPATH** environment variable for this to work. It's not enough to add it to <path id="classpath"> in this file.

classpath is defined on page 9

** specifies to look in any subdirectory at any depth

Sample Build File (Cont'd)

```
  <target name="undeploy" description="undeploys the web app. from Tomcat">
    <delete dir="${tomcatHome}/webapps/${appName}"/>
    <delete file="${tomcatHome}/webapps/${warFile}"/>
  </target>

  <target name="war" depends="compile" description="builds the war file">
    <war warfile="${warFile}" webxml="web.xml">
      <classes dir="${buildDir}"/>
      <fileset dir="${docRoot}"/>
    </war>
  </target>

</project>
```

Makes the servlet unavailable to Tomcat

creates a web application archive (WAR) that can be deployed to a servlet engine like Tomcat

Contains HTML, JavaScript, CSS and XSLT files

Ant Setup

- Download
  - download jakarta-ant-bin.tar.gz and jakarta-ant-1.4-optional.jar from http://jakarta.apache.org/builds/jakarta-ant/release/v1.4/bin/
- Unzip
  - set the ANT_HOME environment variable to the location where Ant will be unzipped … perhaps **/usr/local/Ant**
  - unzip jakarta-ant-1.3-bin.tar.gz into $ANT_HOME ("DON'T USE STUFFIT EXPANDER!!!")
    - additional task documentation not included with this download can be obtained from http://jakarta.apache.org/cvsweb/index.cgi/jakarta-ant/docs/
  - move jakarta-ant-1.3-optional.jar to $ANT_HOME/lib
    - only necessary to use optional Ant tasks such as FTP, JUnit and EJB tasks
    - all JAR files in ANT_HOME/lib are automatically added to CLASSPATH by ant.sh which is run when ant is invoked

Ant Setup (Cont'd)

- Other environment variables
  - set JAVA_HOME to be the location where the JDK is installed
    - for example /usr/local/jdk1.3
  - add to CLASSPATH
    - a JAXP-compliant XML parser such as Xerces
      - download zip file marked "latest binaries" from http://xml.apache.org/dist/xerces-j
      - unzip it and add **xerces.jar** to CLASSPATH
  - add to PATH
    - $ANT_HOME/bin

Using Ant

- ant -projecthelp
  - lists targets in build.xml of the current directory
  - example output

```
Searching for build.xml ...
Buildfile: /cvsroot/tramp/build.xml
Main targets:

 clean          deletes all generated files
 compile        compiles source files
 deploy         deploys the war file to Tomcat
 dtd            generates a DTD for Ant build files
 javadoc        generates javadoc from all .java files
 prepare        create output directories
 test           runs all JUnit tests
 undeploy       undeploys the war file from Tomcat
 war            builds the war file
```

Targets with no description attribute are listed as “**Subtargets**” after the main targets.
These are typically only invoked by other targets via dependencies
or using the Ant and AntCall built-in tasks discussed later.

Using Ant (Cont’d)

- ant [*options*] [*target-names*]
  - runs targets with specified names,
    preceded by targets on which they depend
  - can specify multiple target-names separated by spaces
  - omit target names to run the default target
  - -D option specifies a property that can be used by targets and tasks

    `-D`*`property-name`*`=`*`property-value`*

    - can specify more than one of these
- ant -help
  - lists command-line options

Ant Output

- Indicates the tasks that were executed
  - for example

blank lines were removed
so this would fit on the page

```
Searching for build.xml ...
Buildfile: /cvsroot/tramp/build.xml
prepare:
    [mkdir] Created dir: /cvsroot/tramp/classes
compile:
    [javac] Compiling 26 source files to /cvsroot/tramp/classes
war:
      [war] Building war: /cvsroot/tramp/shopping.war
undeploy:
   [delete] Deleting directory /cvsroot/tramp/shopping
   [delete] Deleting: /cvsroot/tramp/shopping.war
deploy:
     [copy] Copying 1 files to /cvsroot/tramp/webapps
BUILD SUCCESSFUL
Total time: 5 seconds
```

Ant 1.3+ Built-In Tasks

(deprecated tasks omitted)

- Ant
  - calls a target in another build file
  - useful to build subprojects
- AntCall
  - calls a target in the same build file
- AntStructure
  - generates a DTD describing all known tasks
- Apply
  - executes a system command on a set of files only if they are newer than a "target" file
- Available
  - sets a property if a file, class in CLASSPATH, or system resource is present
  - can test for the property being set or not set using the "if" and "unless" attributes of the target element
- Chmod
  - changes permissions of files and directories (only under UNIX now)
- Copy
  - copies files and directories
- Cvs
  - executes any CVS command

Ant 1.3+ Built-In Tasks (Cont'd)

- Delete
  - deletes files and directories
- Echo
  - outputs a message to System.out or a file
- Exec
  - executes a system command
  - can restrict use to a specific OS
- ExecOn
  - like Exec but files and directories are passed as arguments to the system command
- Fail
  - exits the build and optionally prints a message
- Filter
  - used by tasks that copy files to replace all occurrences of an @ delimited string with another string
- FixCRLF
  - changes line endings in a set of files to the convention of the current OS
- GenKey
  - generates a key in a keystore which is a protected database of private keys associated with a digital certificate

Ant 1.3+ Built-In Tasks (Cont'd)

- Get
  - creates a copy of a remote file at a specified URL
    - can use http and ftp URLs
    - can automate software updates
- GUnzip
  - unzips a GZIP file
- GZip
  - creates a GZIP file from a file
- Jar
  - creates a JAR file from a set of files
- Java
  - runs a Java application
- Javac
  - compiles Java source files
- Javadoc
  - generates javadoc HTML files from Java source files
- Mail
  - sends email using SMTP
- Mkdir
  - creates a directory and any missing parent directories
- Move
  - moves files and directories to a new directory
- Patch
  - applies a "diff" to file

Ant 1.3+ Built-In Tasks (Cont'd)

- Property
  - sets properties that can be used in the current target and other targets
  - can load from a property file
- Replace
  - replaces all occurrences of a string with another string in a file
- Rmic
  - runs the rmic compiler on .class files of Java classes that implement java.rmi.Remote
- SignJar
  - uses javasign to add a digital signature to a jar or zip file
- Sql
  - executes a sequence of SQL statements specified in the build file or an external text file
  - output can be written to a file
- Style
  - applies an XSLT stylesheet to a set of XML files to produce a set of output files
  - supports any TrAX-compliant XSLT processor

Transformation API for XML

- Tar
  - creates a TAR file from a set of files
- Taskdef
  - defines a custom task that can be used in the project

See FTP example on page 28

Ant 1.3+ Built-In Tasks (Cont'd)

- Touch
  - creates a file if it doesn't exist
  - updates its modification time if it does
- Tstamp
  - sets the DSTAMP (ccyymmdd), TSTAMP (hhmm) and TODAY (month day year) properties to the current date/time
  - useful for creating files and directories with names that reflect their creation date/time
- Unjar
  - expands a JAR file
- Untar
  - expands a TAR file
- Unwar
  - expands a WAR file
- Unzip
  - expands a ZIP file
- Uptodate
  - sets a specified property if a specified file is newer than a set of source files
- War
  - creates a Web Application Archive from a set of files in a directory structure specified by the Java Servlet spec.
- Zip
  - creates a ZIP file from a set of files

Ant 1.3+ Optional Tasks

- .NET Tasks
  - supports C# and other Microsoft .NET technologies
- ANTLR
  - grammar translator generator
- Cab
  - creates a Microsoft CAB archive from a set of files
- Clearcase Tasks
  - for Clearcase version control
- Depend
  - determines which classes are out of date and removes class files of other classes that depend on them
- EJB Tasks
  - for Enterprise Java Beans
  - tasks include
    - ddcreator
      - compiles deployment descriptors
    - ejbc
      - generates support classes needed to deploy a bean
    - wlrun
      - starts a WebLogic server
    - wlstop
      - stops a WebLogic server
    - ejbjar
      - creates an EJB1.1-compliant JAR file

Ant 1.3+ Optional Tasks (Cont'd)

- FTP
  - lists, gets, puts and deletes files on an FTP server
  - requires NetComponents.jar from http://www.oroinc.com/software/NetComponents.html
- JavaCC
  - CC stands for Compiler Compiler
  - reads a grammar specification and creates a Java application that can recognize matches to the grammar
- Javah
  - generates JNI header files
- JJTree
  - preprocessor for JavaCC
- Jlink
  - builds jar/zip files by merging entries from multiple jar/zip files
- JUnit
  - runs JUnit tests
  - requires junit.jar from http://junit.org
- JUnitReport
  - merges XML results from Junit test cases so an XSLT stylesheet can be applied to produce a single report
- MParse
  - for working with the Metamata Development environment

Ant 1.3+ Optional Tasks (Cont'd)

- Native2Ascii
  - converts files from native encodings to ASCII with escaped Unicode
- NetRexxC
  - compiles NetRexx source files
- Perforce Tasks
  - for Perforce version control
- PropertyFile
  - for editing Java property files
- RenameExtensions
  - changes the file extension on a set of files
- Script
  - executes a script written in a Bean Scripting Framework (BSF) language
  - includes JavaScript, PerlScript, VBScript, JPython and others
- Sound
  - plays a sound file at end of build
  - one for success and one for fail
- Stylebook
  - runs the Apache Stylebook documentation generator
- Telnet
  - automates a telnet session

Ant 1.3+ Optional Tasks (Cont’d)

- Test
  - executes a unit test in the org. apache.testlet framework
- Visual Age for Java Tasks
  - integrates VAJ repository contents into the Ant build process
- VssGet
  - gets files from a Microsoft Visual Source Safe repository
- VssLabel
  - assigns a label to a file or project in VSS

CVS Example

```
<target name="properties">
  <property name="tramp.home" value="." />
  <property name="tramp.src" value="${tramp.home}/src" />
  <property name="tramp.classes" value="${tramp.home}/classes" />
  <property name="tramp.lib" value="${tramp.home}/lib" />
  <property name="tramp.docs" value="${tramp.home}/docs" />
  <property name="tramp.cvsroot" value=":pserver:user@cvsbruegge.in.tum.de:
    /cvs/tramp" />
</target>

<target name="checkout" depends="properties">
  <echo>Logging in: the password for user anonymous is empty;
        just press enter when asked for it</echo>
  <cvs cvsroot="${itext.cvsroot}" command="login" />
  <cvs cvsroot="${itext.cvsroot}" package="src" dest="${itext.home}" />
</target>
```

Creating Custom Tasks

- Steps
  - create a Java class that
    - extends org.apache.tools.ant.Task
    - has a no-arg constructor
  - plan the attributes, text and child elements that your task element will use
  - for each attribute, add a set method

    ```
    public void set*AttrName*(*type attrName*)
    ```
    - *type* can be String or any Java primitive type
    - see Ant documentation for extra information on using enumerated attributes
  - for text, add an addText method

    ```
    public void addText(String text)
    ```

continued on next page

Creating Custom Tasks (Cont'd)

- Steps (cont'd)
  - for each child element, add a create or add method

    `public ChildTask createChildTask()`
    - for empty child task elements

    `public void addChildTask(ChildTask child)`
    - for non-empty child task elements
  - add the method that implements the tasks

    `public void execute()`
  - compile the class
  - insure that it can be found using the CLASSPATH environment variable
- For more information
  - see "Writing Your Own Task" under "Developing with Ant" in the included HTML-based Ant manual

ChildTask must be the name of a class that also follows these steps

Custom Task Example

This task accepts a single attribute called “file”.
It does not use text or child elements.

```
package tramp.ant;

import java.io.File;
import java.util.Date;
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class FileStats extends Task {
  private File file;

  public void execute() throws BuildException {
    System.out.println("    file: " + file.getAbsolutePath());
    System.out.println("  length: " + file.length() + " bytes");
    System.out.println("readable: " + file.canRead());
    System.out.println("writable: " + file.canWrite());
    System.out.println("modified: " + new Date(file.lastModified()));
  }

  public void setFile(String fileName) {
    file = new File(fileName);
  }
}
```

Custom Task Example (Cont'd)

- Target using the custom task

```
<target name="stats" description="displays file statistics">
  <taskdef name="fileStats" classname="tramp.ant.FileStats"/>
  <fileStats file="Test.java"/>
</target>
```

This can be avoided by registering the custom task in defaults.properties in the org.apache.tools.ant.taskdefs package along with the built-in tasks. Extract it from ant.jar, modify it and either put it back in ant.jar or place it so that it will be found within CLASSPATH before ant.jar

- Output of the target

```
Searching for build.xml ...
Buildfile: C:\XMLProgLabs\Framework\build.xml

stats:
    file: C:\XMLProgLabs\Framework\Test.java
  length: 5388 bytes
readable: true
writable: true
modified: Sat Nov 25 10:49:52 CST 2000

BUILD SUCCESSFUL

Total time: 1 second
```