

Software Build Techniques

a.k.a.

Stupid Ant Tricks

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii
Honolulu HI 96822

(1)

Objectives

Understand motivation for build and packaging technologies

Be able to use Ant for build/packaging of Java systems

Be able to structure a system for Ant-based build/packaging using CSDL best practices.

(2)

Motivation

Modern team-based, cross-platform development involves:

- Multiple developers and users.
- System is developed/used on multiple platforms:
 - Win2K, Linux, Win98, Solaris
- Developers/users have different environments:
 - c:\jdk1.3, d:\jdk1.3, c:\jdk1.2.2
- Differences between user and developer distributions:
 - Source distribution for developers
 - Binary distribution for users
- Developers may prefer different IDEs:
 - JBuilder, Emacs, RationalRose
- Users may not have any IDE at all.
- Frequent releases and distributions.

These issues are not unique to Java development!

(3)

Motivation (cont.)

Build/packaging techniques should allow:

- Users to build and test the system without IDE.
- Developers to build and test the system using their IDE of choice.
- Developers to build and test the way users build and test.
- Configuration to different platforms and different environments within a platform.
- Multiple release configurations
- Automated build to prevent errors
- Rapid release, deployment, testing

One (Java-specific) solution:

- Ant
- Build/packaging best practices from CSDL

(4)

Make

'Make' is the canonical build tool.

- Possibly the most useful software engineering tool ever invented.

Make features:

- Dependency analysis
- Unix-based (but ported to other platforms)
- Language independent (Java, C++, etc.)
- ASCII configuration file syntax
 - Tab characters required— Yikes!
- Supports invocation of any shell command.

(5)

Ant

Java-based build tool by Apache

- <http://jakarta.apache.org/ant/>

Features:

- Java-based
- Cross-platform
- Extensible
- XML configuration files
- Open source
- Defacto standard for Java projects

Online Documentation:

- <http://jakarta.apache.org/ant/manual/>

(6)

Ant vs. Make

Advantages of Make:

- More popular and widespread
- Supports any programming language.
- Provides 'glue' for invoking any shell command.

Advantages of Ant:

- Java-based, active development community
- Built-in support for Java and its tools.
- build.xml simpler and easier than Makefile.
- Simplifies cross-platform Java development
- Easier to extend in a portable manner.

Current recommendation:

- Always use Ant for Java-based development
- Use Make for other development

(7)

Ant concepts

Every project requires a build file

- named build.xml by default.

Each build file is composed of targets.

- Typical targets: compile, test, etc.

Each target is composed of tasks

- copy .java files to build/src, invoke javac...

Targets can have dependencies

- modified sources must be recompiled before testing.

(8)

Ant targets

Dependent targets are executed exactly once.

• Example:

- test depends upon compile
- deploy depends upon compile
- all depends upon test, deploy

- Compile will be executed only once.

Some targets are executed only when needed.

- Sources recompiled only if changed.

(9)

Example Ant target

```
<target name="compile"
  depends="prepare"
  description="Compiles code.">

  <javac srcdir="${build.src}"
    destdir="${build.dest}"
    />

</target>
```

(10)

Best practices

Ant provides a simple, powerful language for building Java-based systems.

How do we use this language effectively?

The next section describes my best approach so far on how to employ Ant to achieve the Three Prime Directives of Open Source Software Engineering.

The "stack" example application provides you with a template structure to get you off and running quickly.

(11)

Stack: An example Ant-based package

The stack system provides a simple ADT with push and pop.

- Not very interesting.

The stack package provides best practices for:

- The package directory structure
- README and other standard files
- Shell configuration-based setup (SetEnv)
- Ant extensions (junit, j2h, checkstyle, locc)

Note: we will augment this basic approach for web application development later in the semester.

- httpunit, Tomcat, etc.

(12)

Stack top-level directory contents

- README.html
 - contains initial instructions
- setEnv.bat, setEnv.csh
 - configures the shell environment
- build.xml
 - build data for Ant.
- src/
 - the system source files
- lib/
 - library files and data
- doc/
 - documentation files.
- build/
 - artifacts of the build process

(13)

README.html

- Contents:
- A. Contents of the package
 - package substructure
 - B. Installation
 - installing Java and setting JAVA_HOME
 - invoking setenv to install Ant.
 - invoking ant to build/install
 - C. Invocation
 - how to use the system
 - D. User documentation
 - typically HTML in the doc/ directory
 - E. Developer documentation
 - also HTML in the doc/ directory
 - F. Contact information
 - website, email, mailing lists, CVS

(14)

SetEnv (bootstrapping)

Sets PATH and defines important env vars.
Requires JAVA_HOME to be defined.

Resulting impact on shell environment:
•'ant' and 'java' defined as shell commands
•CLASSPATH set to null to avoid conflicts in this shell.

Creates a shell configured for building system.
Will be extended for Tomcat and client/server.
Does NOT require separate Ant download.

(15)

Basic build.xml tasks

- | | |
|--|----------------------------|
| init | dist |
| •sets properties | •creates distribution .zip |
| prepare | j2h |
| •creates directories | •creates HTML source |
| compile | size |
| •compiles system | •computes size using LOCC |
| package | checkstyle |
| •creates jar file | •checks src format |
| test | javadoc |
| •runs checkstyle | •creates javadocs |
| •runs junit | clean |
| •runs junitreport | •deletes build/,backups |
| •requires Junit test class names to start with "Test". | |

(16)

The lib/ directory

- Contains data and executables:
- lib/Ant1.4.1/
 - contains the Ant build package.
 - lib/Ant1.4.1/lib
 - contains jar files for Ant and for system.
 - all jar files automatically on CLASSPATH during build!
 - lib/Jbuilder
 - contains project.jpx file
 - facilitates group-based development
 - lib/manifest
 - contains main-class specification
 - enables 'java -jar stack.jar'

(17)

The doc/ directory

- Contains documentation files:
- doc/LicenseInfo.html
 - Copies of licenses for ALL software included in release.
 - (Pointer to this file should be in every source file.)
 - doc/History.html
 - Provides a written record of the public releases and the changes made.
 - Helps users decide whether to upgrade and the pros/cons of the upgrade.
 - doc/Jbuilder.html
 - Provides instructions on using the .jpx file in the lib/Jbuilder directory.
- Other possible documentation:
- User guide, architecture overview, CVS, etc.

(18)

The build/ directory

Contains files derived from build process.
Can be deleted and reconstructed.
Not included in CVS or in distribution .zip.

Build subdirs and the targets that create them:

•build/api/	javadoc
•build/classes/	compile
•build/dist/	dist
•build/size_data/	size
•build/src/	compile
•build/src_html/	j2h
•build/test_output/	test

(19)

Compilation Notes

The build process makes a copy of the entire source tree in the build directory before starting compile.

- This allows Ant-based "token filtering".
- Example: Stack embeds the release number into the source code which is printed out with 'java -jar stack.jar'.

All .class files placed in build/classes.

It is better to keep build products out of src/
•Facilitates CVS management later.

(20)

Testing Notes

The "test" target performs two kinds of tests:

- Formatting tests using Checkstyle
 - Ensures your code obeys most (but not all) style guidelines.
- Unit test using JUnit
 - Ensures your code executes correctly (if you write good test cases.)

Encourages incremental testing and formatting.

Note: JRefactory "pretty printer" implements template JavaDocs and other style guidelines.

- Requires some tweaking (2 space indentation, etc.)
- Beware of turning in code with "Description of" comments.
 - I will search for these in your code.
 - Use JBuilder "Search in path" to find and remove.

(21)

Useful build/ subdir files

build/api/index.html

- JavaDoc design documentation output root

build/dist/stack-2.0122.zip

- The zip file, ready for distribution.

build/size_data/size.csv

- Excel spreadsheet with size data.

build/src_html/index.html

- J2H HTML source output root

build/test_output/index.html

- JUnit report output root

(22)

Distribution release numbering

Traditional approach (i.e. jakarta-tomcat-4.0.1.zip)

- MM.mm.bb, where:
 - MM = Major release number (major redesign or incompatible changes)
 - mm = Minor release number (minor enhancements, backward compatible)
 - bb = Bugfix release number (bug fix, no new functionality)
- All numbers maintained manually.

Advantage:

- Release number comparison indicates degree of change.

Disadvantage:

- Requires manual editing of release value every time which leads to errors (easy to forget to do this.)

(23)

Distribution release numbering

Our approach (i.e. stack-2.0122.zip)

- M.mmdd, where:
 - M = major release number
 - mm = month
 - dd = day
- Major release number maintained manually.
- mmdd generated automatically

Advantages:

- Automatic except when incrementing major release.
- Release number indicates date.

Disadvantages:

- Minor/bugfix info lost from release number.
- Can't have > 1 distribution per day.
- Must inc major release number at least once/year.

(24)

Useful extensions

Directory extensions:

- bin/ (contains additional batch files.)

SetEnv extensions

- Tomcat env vars and path additions to support client-server builds and tests.

Ant/lib extensions

- httpunit.jar (web UI testing),
- servlet.jar (servlet compilation)
- jdom.jar (XML processing)

Separate binary and source distributions

- Provide 'distSrc' and 'distBin' targets.
- May require different README.html for each dist.
- Not worth it until system gets both big and popular.

(25)

IDE integration

Important to maintain independence from IDE.

- All build functions should be possible without any IDE.

But it's fine to provide IDE-specific files to facilitate development.

- Example: stack.jpj with pre-built project to facilitate compilation/enhancement in Jbuilder.

Provide additional directories for other IDEs if desired.

(26)

Stack as a standard

All of your projects must be developed, packaged, and submitted using the standard build conventions embodied in the stack package.

In particular, AccountInfo1 must be turned in as a zip distribution file created by the "dist" task.

I will unzip and run your Ant targets as part of the evaluation process.

I will also load and check your .jpx file.

(27)

Stack as a template

How to create Ant-based build for your system:

- Download and unzip stack .zip distribution.
- Install and test all targets.
- Rename top-level directory to your project name.
- Delete interior of src/ directory.
- Copy your sources into src/
- Edit the five property values in the "System specific configuration" section of the init task for your system's situation.
- Replace lib/Jbuilder/stack.jpj file with the one for your own system.
- Edit: README.html, History.html, LicenseInfo.html, manifest, Jbuilder.html.
- Test and verify that all Ant targets are now working on your system.

(28)

Project Name Convention

I will be downloading and reviewing 40 projects.

They can't all be named "accountinfo1".

Please use the following template for your project names: <username>-<project>.

Thus:

- johnson-accountinfo1.zip
- tburns-accountinfo1.zip
- etc.

(29)

Limitations of stack example

The stack template provides a framework plus basic build functionality, but has limitations:

- No CVS integration
- No support for Tomcat and client-server issues.
- Many other targets needed for webapp development (war file generation, etc.)

We will augment stack's basic functionality with new targets during the semester.

(30)

Meta-level goal

Provide an efficient solution to satisfying both

- PD#2 (users can install system) and
- PD#3 (developers can understand and enhance system)

Other approaches exist:

- InstallAnywhere for PD#2 (more work)
- IDE-specific solution for PD#3 (limited)

With good installation documentation, I believe Ant-based build can satisfy both PD#2 and PD#3.

(31)

Ant competitors

GNU Make

- <http://www.gnu.org/software/make/make.html>

Jam (Competitor for Make)

- <http://freetype.sourceforge.net/jam/>

Nant: (Ant for .Net)

- <http://nant.sourceforge.net/>

(32)

What to do next

Readings (links at module page):

- Ant User Manual
- Articles on Ant

Port your project over to Stack template.

Begin using Ant-based build as daily component of your development.

(33)

In-class demo of Ant

(34)