Apache Ant: Another Neat Tool

Martin Bravenboer
mbravenb@cs.uu.nl

Institute of Information and Computing Sciences
University Utrecht
The Netherlands
Outline

- history and current status
- (dis)advantages
- basic build-configuration structure
- some examples
- task overview
- implementing your own tasks
- Ant tasks for XT
- features of Ant 2.0
History

• created as part of Tomcat
• donated to the Apache Software Foundation
• separated in January 2000
• official 1.1 release 19 July 2000
• current version: 1.4.1
• major re-design: Ant 2.0
Current status

- tremendously popular in Java community
- active user-group developing tasks
- used together with XP, unit-testing (Java Tools for Extreme Programming)
- integration with many IDE’s: NetBeans, Forte, JBuilder, Jext, jEdit, IntelliJ IDEA, Codeguide
- often mentioned in discussions on XML pipeline languages
Main advantages

- platform-independent
- clear syntax (yet verbose)
- model independent of syntax
- Tasks create attractive and consistent view on tools.
- somewhat declarative
- easy to develop new tasks
- (Java only) tools can be executed in one JVM
Some disadvantages

- poor integration, loss of great platform-specific features
- no knowledge of other tasks, Ant makes no assumptions about for example ‘results’
- task hides the features of the wrapped tool
- some tasks are simply not platform-independent
Build configuration: structure

Project has targets with possibly dependencies between them

- project: list(target) → project
- target: list(task-invocation) → target
- Tasks of the specified target(s) are *always* executed.
- Dependencies specify the order in which targets are executed.
- Targets are only executed *one* time.
Basic Java example: structure

```xml
<project name="basic-example" default="compile" basedir="..">
    <target name="clearbuild">
    <target name="compile">
    <target name="collect-jarcontent" depends="compile">
    <target name="jar" depends="collect-jarcontent">
    <target name="api-doc">
</project>
```
Basic Java example: properties

```xml
<project name="basic-example" default="compile" basedir="..">

    <property name="app.name" value="basic-example"/>
    <property name="build" value="build"/>
    <property name="src" value="src"/>

    <path id="project.class.path">
        <pathelement path="${classpath}" />
        <fileset dir="src/lib">
            <include name="**/*.jar"/>
        </fileset>
    </path>

    ...

</project>
```
Basic Java example: clearbuild

```xml
<target name="clearbuild">
    <delete>
        <fileset dir="${build}" includes="**"/>
    </delete>
</target>
```
Basic Java example: compile

```xml
<target name="compile">
  <mkdir dir="${build.root}/classes"/>

  <javac srcdir="${src.root}/java" 
    destdir="${build.root}/classes" 
    deprecation="on" 
    includes="**/*.java" 
    classpathref="project.class.path"/>

</target>
```
<target name="collect-jarcontent"
    depends="compile">
    <mkdir dir="${build}/jar-content"/>
    <copy todir="${build}/jar-content">
        <fileset dir="${build}/classes"/>
    </copy>
    <copy todir="${build}/jar-content">
        <fileset dir="${src}/jar-content"/>
    </copy>
</target>
Basic Java example: jar

```xml
<target name="jar" depends="collect-jarcontent">
  <mkdir dir="${build.root}/lib"/>

  <jar jarfile="${build}/lib/${app.name}.jar" basedir="${build}/jar-content"
       includes="**"
       manifest="${src.root}/Manifest.mf"/>
</target>
```
Basic Java example: api-doc

```xml
<target name="api-doc">
  <javadoc
      packagenames="org.pandoramix.*"
      sourcepath="${src.root}/java"
      destdir="${build.root}/api-docs"
      windowtitle="Ant Example API documentation"
      doctitle="Ant Example API documentation"
      classpathref="project.class.path"/>
</target>
```
Available tasks

- Java: standard tools, unit-testing, parser-generators, AspectJ, Javadoc, rmic
- File operations: copy, delete, zip, cab, tar, gzip, touch, chmod
- FTP, Get, Mail, Telnet, CVS, Microsoft Visual SourceSafe, Continuus Source Manager
- XML validating, XSL Transformations
- .NET: csharp compiler, IL assembler etc
Website management: problem

Website:
- XML, XHTML, Software
- XSL Transformations with shell scripts
- FTP by-hand

Problem:
- not platform-independent
- obscure and verbose shell-scripts
- what’s new?
Website management: solution

<project name="website-example" default="build" basedir="..">

  <property name="build" value="build"/>
  <property name="src" value="src"/>
  <property name="distr" value="distr/version_${version}/"/>

  <target name="clearbuild">
  <target name="build">
  <target name="validate-build" depends="build">
  <target name="distribute" depends="build">
  <target name="deploy" depends="distribute">
</project>
Website management: solution

```xml
<target name="clearbuild">
  <delete>
    <fileset dir="${build}" includes="**" />
  </delete>
</target>
```
Website management: solution

```xml
<target name="build">
  <mkdir dir="${build}/content"/>
  <mkdir dir="${build}/wwwroot"/>

  <copy todir="${build}/content/">
    <fileset dir="${src}/content"/>
  </copy>

  <style basedir="${build}/content/"
    includes="**/*.xhtml"
    destdir="${build}/wwwroot/"
    style="${src}/transform/table-navigation.xsl"
    extension=".xhtml"/>

</target>
```
Website management: solution

```xml
<target name="validate-build" depends="build">
  <xmlvalidate lenient="no"
              warn="yes">
    <fileset dir="${build}/wwwroot"
             includes="**/*.xhtml"/>
  </xmlvalidate>
</target>
```
Website management: solution

```xml
<target name="distribute" depends="build">
  <mkdir dir="${distr}/wwwroot"/>

  <copy todir="${distr}/wwwroot/">
    <fileset dir="${build}/wwwroot"/>
    <fileset dir="${src}/wwwroot"/>
  </copy>
</target>
```
Website management: solution

```xml
<target name="deploy" depends="distribute">
  <ftp server="****" remotedir="****" userid="****" password="****"
       depends="yes" verbose="yes">
    <fileset dir="${distr}/wwwroot"/>
  </ftp>
</target>
```
Unit-testing

- XP: incremental testing and continuous builds and integration
- requires automated builds
- JUnit tasks for Ant: test in every build
- JUnitReport: attractive report generation
<junit haltonfailure="yes">
  <classpath>
    <pathelement location="${build}/classes"/>
    <pathelement location="${build}/tests"/>
  </classpath>

  <formatter type="plain" usefile="no"/>

  <batchtest>
    <fileset dir="${build.root}/tests">
      <include name="**/*Test.class" />
    </fileset>
  </batchtest>
</junit>
JUnit application with reporting

```xml
<junit>
  <formatter type="xml"/>
  <batchtest todir="${build}/test-reports">
    <fileset dir="${build}/tests">
      <include name="**/*Test.class" />
    </fileset>
  </batchtest>
</junit>

<junitreport todir="${build}/test-report">
  <fileset dir="${build}/test-reports">
    <include name="TEST-*.xml"/>
  </fileset>
  <report format="frames" todir="${build}/test-report/html"/>
</junitreport>
```
Ant is very popular because:

- simple and clear
- platform-independent
- Tasks provide an attractive and consistent view.
- easy to separate results (and buildfiles) from source

conclusion: We (well, at least I) want tasks for XT.
Writing your own tasks

- Ant makes very heavy use of reflection
- attributes:
  - tasks must implement set methods for attributes
  - arguments: String, primitives, Class, File, constructable from String
- character content: addText(String)
- invocation: execute()
- sub-elements: add or create method
Writing your own tasks: example

```java
package org.pandoramix.ant.taskdefs;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class SystemPropertyTask extends Task {
    private String _id;

    public void execute() throws BuildException {
        System.out.println(_id + " " + System.getProperty(_id));
    }

    public void setProperty(String id) {
        _id = id;
    }
}
```
XT Tasks: SDF module to parse-table

- SDF Module to parse-table:

  ```xml
  <sdf-to-table
     module="${src}/grammar/Main.sdf"
     dest="${build}/share/Tiger.tbl"/>
  ```

- calls pack-sdf, asfix-yield, sdf2table (also available as separate tasks)
XT Tasks: SDF to parse-table

- SDF definition to parse-table:

  `<sdf-to-ptable
   def="${src}/grammar/Tiger.def"
   dest="${build}/share/Tiger.tbl"/>`

- calls asfix-yield, sdf2table
XT Tasks: Module paths

- specifying paths to look for modules:

```xml
<sdf-to-htable
    module="${src}/grammar/Main.sdf"
    dest="${build}/share/Tiger.tbl">
  <modulepath>
    <path refid="cst-modules"/>
  </modulepath>
</sdf-to-htable>
```
XT Tasks: Stratego compiler

```xml
<path id="example.modules">
    <pathelement location="modules"/>
</path>

<target name="compile">
    <stratego-compile
        module="OrderCalc"
        destdir="build">
        <modulepath refid="example.modules"/>
    </stratego-compile>
</target>
```
<stratego-compile
    module="${src}/cst-to-ast/Tiger-Desugar"
    main="Tiger-Desugar"
    destdir="${build}/bin">

<modulepath>
    <path refid="tiger-cst-modules"/>
    <path refid="tiger-ast-modules"/>
</modulepath>
</stratego-compile>
XT Tasks: To do

- implement tasks to compile the Tiger compiler
- lazyness in tasks, dependency checking
- implement pipes to make easy pipelining of tools possible
- tasks like JUnit/JUnitReport for SUnit
Ant has no core support for pipes

- causes:
  - Intermediate files
  - Tasks should provide many different views (not atomic)

- solution: Ant has a TaskContainer, which could be used to implement a pipe
Let pipe be a TaskContainer. The sub-task must implement some 'Pipeable' interface.

```xml
<pipe in="${src}/grammar/Main.sdf"
    out="${build}/share/Tiger.tbl">
    <pack-sdf/>
    <asfix-yield/>
    <sdf-to-table/>
</pipe>
```
Some features of Ant 2.0

- better design of tasks: extract functionality for reuse
- extraction of common attributes in AspectHandlers
- better support for container tasks
- lazy-instantiation of tasks
- tasks will provide DTDs/schema’s
- compose build files from multiple sources
- better integration of new tasks
- a logo!
That’s it