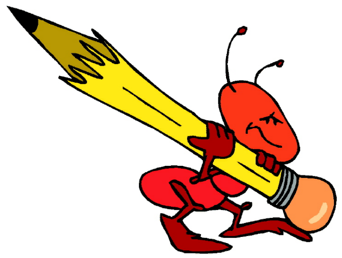
Accelerating and Automating the Build Process with IBM® Rational ClearCase® and Ant

Kevin Lee
IBM Rational Software
kevin.lee@uk.ibm.com

Rational software

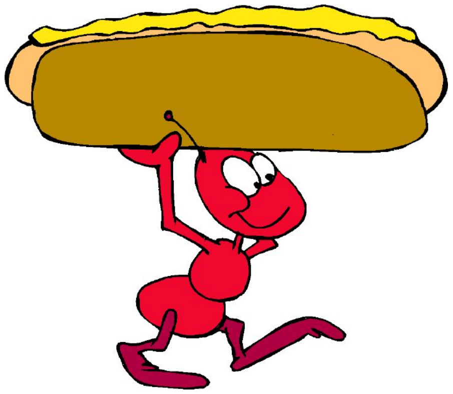


RATIONAL SOFTWARE DEVELOPMENT USER CONFERENCE

software runs the world 04

Agenda

- Overview of Ant
  - What is Ant
  - The Ant build file
  - Typical Ant sequence
- Overview of the Ant integration with ClearCase
- ClearCase Ant Patterns
- Demo



What is Ant?

- What is Ant?
  - Java-based build tool
  - De-facto standard for building Java projects
- Why use Ant?
  - Cross-platform
  - Java domain smart
  - Fast, extensible, integrated

The Ant build file

- XML format
- Default name: *build.xml*
- Typically in project root directory

- Defines a single project
- A project contains targets
- Targets contain tasks

Typical Ant sequence

project

targets

tasks

```
<?xml version="1.0" ?>
<project name="RationalDemo" default="compile">
    …

    <target name="init">…</target>

    <target name="clean" description="remove generated files">
      …
    </target>

    <target name="compile"
            depends="init" description="compile source code">
            <javac … />
    </target>

    <target name="dist"
            depends="compile" description="create distribution jar file">
            <jar … />
    </target>

</project>
```

ClearCase Ant Tasks

- Ant has a number of tasks for integration with ClearCase
- These tasks interface with "cleartool" command
- Current commands (in Ant 1.6.1):
  - `cccheckin`
  - `cccheckout`
  - `cclock`
  - `ccmkattr`
  - `ccmkdir`
  - `ccmkelem`
  - `ccmkbl`
  - `ccmklabel`
  - `ccmklbtype`
  - `ccrmtype`
  - `ccuncheckout`
  - `ccunlock`
  - `ccupdate`

ClearCase Ant Tasks cont…

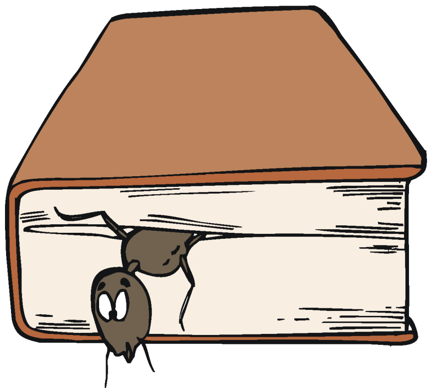
- These tasks do not cover all the actions you might want to carry out as part of the build, particularly if using UCM.
- However, it is easy to extend and create new tasks.
- For example:
  - `ccchbl`
  - `ccdiffbl`
  - `ccmkactivity`
  - `ccsetactivity`
  - `clearauditant`
  - `clearauditjarcr`

Example Ant ClearCase sequence

```
<target name="clearcase-pre"
          depends="init"
          description="execute ClearCase pre compile commands">

          <!-- update snapshot view -->
          <ccupdate viewpath="${user.dir}\.." graphical="false"
                    overwrite="true" currenttime="true" rename="false"/>

          <!-- lock the build branch -->
          <cclock objsel="brtype:project_int"
                    replace="true" nusers="ccadm"/>

          <!-- checkout files to be updated -->
          <cccheckout viewpath="src\com\ratlbank\model\Bank.java"
                    reserved="true" notco="false" />
</target>
```

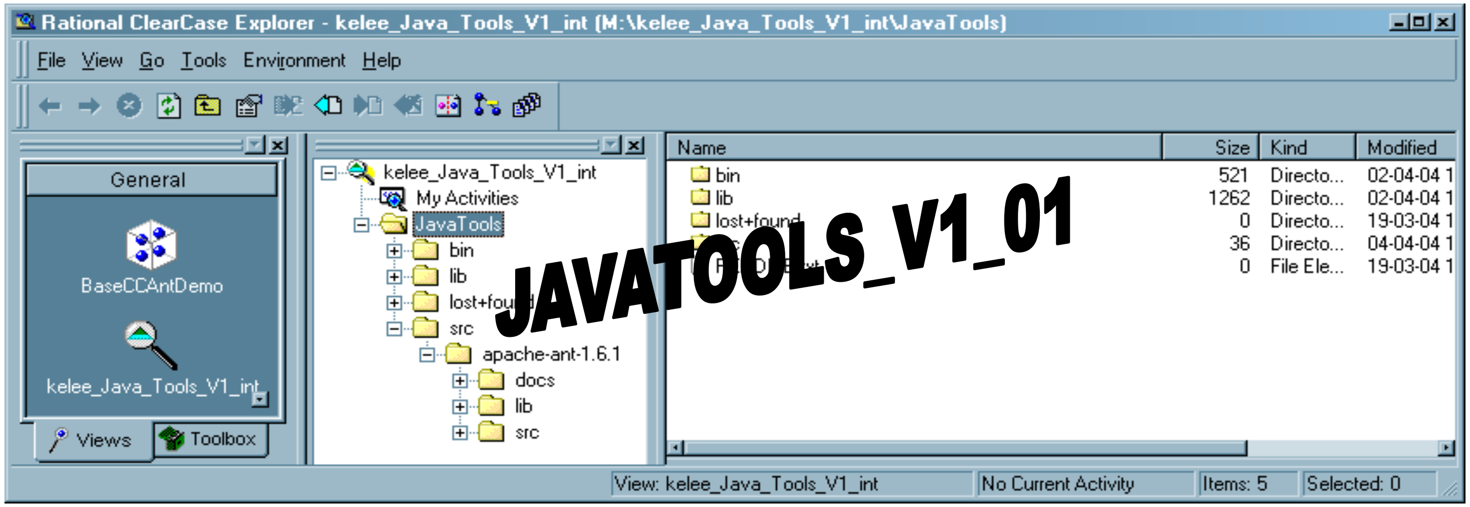
ClearCase Ant Patterns

Some examples and proven scenarios of how to use the features of Ant and ClearCase to get accelerate the build process…



1. Baseline Java tools as a single unit

- Problem:
  - Users are unsure of which versions of Java tools to use, i.e. which version of junit, log4j, checkstyle, ant itself should I be using? Not strictly Ant specific…
- Solution:
  - Place all the tools under version control and baseline them as a unit
  - In UCM create a component for it



Rational ClearCase Explorer - kelee_Java_Tools_V1_int (M:\kelee_Java_Tools_V1_int\JavaTools)

```
kelee_Java_Tools_V1_int
    My Activities
    JavaTools
        bin
        lib
        lost+found
        src
            apache-ant-1.6.1
                docs
                lib
                src
```

Name	Size	Kind	Modified
bin	521	Directo...	02-04-04 1
lib	1262	Directo...	02-04-04 1
lost+found	0	Directo...	19-03-04 1
[illegible]	36	Directo...	04-04-04 1
[illegible]	0	File Ele...	19-03-04 1

View: kelee_Java_Tools_V1_int | No Current Activity | Items: 5 | Selected: 0

2. Referencing Java tools libraries

- Problem:
  - We want to define a (multi-platform) classpath which includes our Java tools libraries
- Solution:
  - Define a path id and convert to O/S specific format
- Implementation:
  - Define path id based on relative location:

```
<path id="project.classpath">
          <pathelement location="${dir.build}"/>
          <!-- include java tools -->
          <fileset dir="${user.dir}\..\JavaTools\lib">
                    <include name="*.jar"/>
          </fileset>
</path>
```

3. Utilizing a Properties File

- Problem:
  - Project specific references mean *build.xml* file needs reworking for each new project
- Solution:
  - Maintain a *build.properties* file at the same level as the build.xml* file
  - Also means users can override them, if necessary
- Implementation:
  - Example *build.properties* file:

```
# build properties
name.project-vob     = RationalProjects
name.project         = RatlBankModel
name.build.prefix    = RATLBNK-MODEL
name.build.admin     = ccadm
name.build.branch    = RatlBankModel_Integration
file.main.class      = com.ratlbank.main.BankMain
```

3. Utilizing a Properties File cont…

- Implementation cont…
  - Loading the properties file:

```
…
<property file="build.properties" prefix="bp" />
…
```

  - Referencing the properties:

```
…
<cclock objsel="brtype:${bp.name.build.branch}"
          replace="true" nusers="${bp.name.build.admin}"/>
…
```

4. Generating build labels

- Problem:
  - Need to automatically generate a suitable ClearCase baseline/label and include it in the code or jar manifest file
- Solution:
  - Use ant’s *buildnumber* tag
- Implementation:
  - In *build.properties* file, make reference a *buildinfo* file (the file that will store the build number) and the source file which should be updated to include the build number:

```
…
file.build.info      = buildinfo.properties
file.build.referer   = src/com/ratlbank/model/Bank.java
…
```

4. Generating a BuildInfo file cont…

- Implementation cont…
  - Include string to replace in source file:

```
private final static String version = "@(#)<label> (on:<date>)@";
```

  - Generate the *buildinfo.properties* file with build number and date in:

```
<propertyfile file="${bp.file.build.info}"
          comment="Build Information File - DO NOT CHANGE" >
          <entry key="build.num" type="int default="0000"
                    operation="+" pattern="0000" />
          <entry key="build.date" type="date" value="now"
                    pattern="dd.MM.yyyy HH:mm" />
</propertyfile>
```

  - Update the specific source file with this version number:

```
<replaceregexp file="${bp.file.build.referer}"
          match="@\(#\).*@"   replace="@(#)$
{bp.name.build.prefix}-${build.num} (on: ${build.date})@" />
```

5. Generating “good” baselines

- Problem:
  - How can we make sure that the baseline we apply is “good” and suitable for further development
- Solution:
  - Generate the baseline before the build, use junit to run some basic acceptance tests and promote the baseline (UCM only) after the successful build.
- Implementation:
  - Lock down the integration stream, set into a build activity and apply the baseline:

```
<!-- lock the integration branch -->
<cclock objsel="brtype:${bp.name.build.branch}"
          replace="true" nusers="${bp.name.build.admin}"/>
<!- set into the build activity -->
<ccsetactivity activityselector="${bp.name.build.activity}"/>
<!- apply the baseline -->
<ccmkbl baselinerootname="${bp.name.build.prefix}-${build.num}"
    identical="yes" full="yes" viewpath=". " />
```

5. Generating “good” baselines cont…

- Implementation cont…
  - Run the junit tests:

```
...
<junit printsummary="on" fork="no" haltonfailure="false"
          failureproperty="tests.failed“ showoutput="true">
          <classpath refid="project.classpath"/>
          <formatter type="xml"/>
          <batchtest todir="${dir.build}">
                    <fileset dir="${dir.src}">
                              <include name="**/Test*.java"/>
                    </fileset>
          </batchtest>
</junit>
```

  - If the build succeeds, promote the baseline to “BUILT”:

```
<ccchbl baselineselector="${bp.name.build.prefix}-${build.num}"
          level="BUILT" nrecurse="true" />
```

6. System versus user build

- Problem:
  - We don’t want users to carry out a full system build (i.e. generating build numbers, applying baselines etc), but don’t want to maintain separate *build.xml* files.
- Solution:
  - Create a *system* build target which invokes the system build operations.
- Implementation:

```
<target name="system" description="generate system build">
         <antcall target="update-buildinfo" />
         <antcall target="junit-all" />
         <antcall target="baseline" />
         <antcall target="dist" />
         <antcall target="javadoc" />
</target>
```

7. Automatically generating build reports

- Problem:
  - We want to automatically generate a build log, junit test log and also a ClearCase report of what went into the build, i.e. file versions or UCM activities.
- Solution:
  - Use the ant XML logger, the *junitreport* task and the *ccdiffbl* task.
- Implementation:
  - Start off the build directing the output to the XML logger:

```
C:\>ant -logger org.apache.tools.ant.XmlLogger -logfile build
\log.xml <target>
```

  - (You can specify the stylesheet to use on the command line or in the build.xml file itself)

7. Automatically generating build reports cont…

- Implementation cont…
  - Run the *junitreport* task to create a junit test log:

```
<junitreport todir="${dir.build}">
          <fileset dir="${dir.build}">
                    <include name="TEST-*.xml"/>
          </fileset>
          <report format="noframes" todir="${dir.build}"/>
</junitreport>
```

  - Run the *ccdiffbl* task to generate the ClearCase contents report

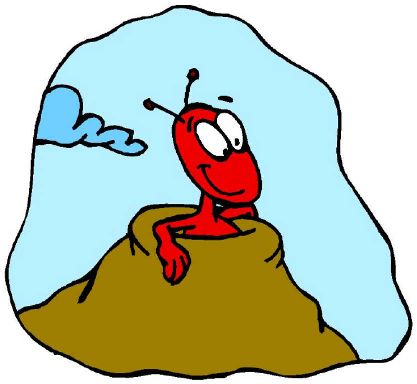
```
<record name="${dir.build}\clearcase.txt" action="start" />
<ccdiffbl baselineselector="${bp.name.build.prefix}-${build.num}"
          predecessor="true" versions="true" />
<record name="${dir.build}\clearcase.txt" action="stop" />
```

8. Performing a ClearCase audit

- Problem
- Solution
- Implementation

Other tips

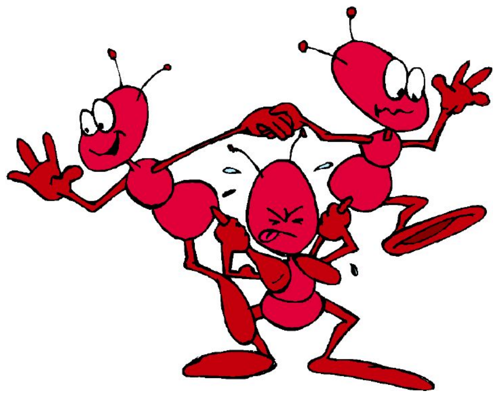
- Some other ideas…
  - Use the ClearCase scheduler for scheduling automating builds



Example reports

DEMO

The Easy Way



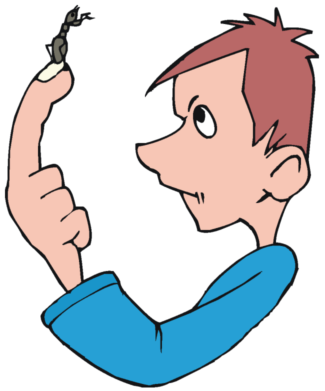
Apache Maven Project
http://maven.apache.org/

cruisecontrol
continuous integration toolkit

Anthill Build Server

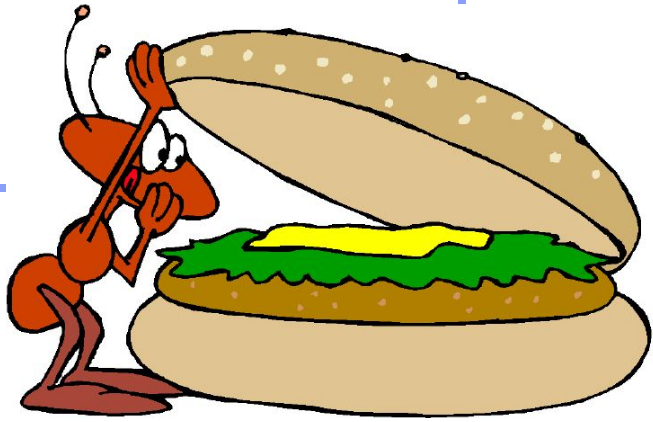
Summary

- Ant – very powerful tool
- Careful consideration to get the most out of ClearCae



References

QUESTIONS



Thank You

Kevin Lee
kevin.lee@uk.ibm.com