AntOddities

- Expanding wildcards in <exec> arguments
- <fileset>'s strange behaviour
- Changing default Locale
- Using your own classes inside <script>
  - How to import other (non java.*) classes, e.g. Ant´s own classes?
  - How to import classes which are '"not"' on Ant´s classpath?
- Writing a "Task" for getting the dependency list for a target
- Using Ant to download files and check their integrity
- Windows XP exec task : use os="Windows XP" instead of "os="Windows NT" despite docs
- Implementing a PreProcessor
- Compounding a property name from the instantiations of multiple previously instanced properties
  - The Problem
  - Solution Macrodef
- Why does <javac> require you to set source="1.4" for Java1.4 features, "1.5" for Java1.5?

Expanding wildcards in <exec> arguments

On Unix-like systems, wildcards are understood by shell intepreters, not by individual binary executables as /usr/bin/ls or shell scripts.

{{{<target name="list">

<exec executable="sh">

<arg value="-c"/> <arg value="ls /path/to/some/xml/files/*.xml"/>

</exec>

</target>}}}

Under Windows, instead, you can expect wildcards to be understood even without the need of invoking the cmd interpreter.

{{{<target name="compile-groovy-scripts">

<exec executable="groovy.bat">

<arg value="C:/path/to/scripts/*.groovy"/> <arg line="-d C:/path/to/classes/destination"/>

</exec>

</target>}}}

Giulio Piancastelli

<fileset>'s strange behaviour

Here is an oddity whose solution was discovered by Jan Matèrne. In Ant, what is the simplest way to get a <fileset> that contains only files that do **not** have an extension in a directory tree.

You might think you could just specify that the included files end in a period, like so:

```
<fileset includes="**/*." />
```

but that will only select files which literally end in a period. The actual answer is a bit counterintuitive. You select all files (implicit) and then exclude those that have an extension:

```
<fileset excludes="*.*" />
```

This probably only looks odd to people who think of file extensions as being something special. For Unix people, Ant's behavior isn't counterintuitive at all, as the dot in the first includes pattern is a literal dot and nothing else.

Changing default Locale

When I worked with CheckStyle I realized that there are localized message. So far so fine. But now I want to generate an international (English) site on my (German) machine. But how to realize that?

CheckStyle - like many other programs - uses the java.util.ResourceBundle.getBundle() method which returns the appropriate bundle for the **default** Locale . So I will set the default Locale to the US value. Before that I store the actual one (or the 'key') as property and restore that after invoking CheckStyle. Because I need (simple) access to the Java API, I write that inside <script> tasks:

{{{ <script language="javascript"> <![CDATA[

    importClass(java.util.Locale); actualDefault = Locale.getDefault(); project.setProperty("---actual-default-locale---", actualDefault); Locale.setDefault (Locale.US);

    ]]></script>

    <ant .../>

    <script language="javascript"> <![CDATA[

        importClass(java.util.Locale); actualDefault = project.getProperty("---actual-default-locale---"); Locale.setDefault(new Locale (actualDefault));

    ]]></script> }}}

Jan Matèrne

P.S. For another thing I needed to change the Locale by setting parameters on VM startup. I found a solution on Eclipse-Bug-Database. Kevin Barnes offered the possibility to set two VM args: -Duser.country=EN -Duser.language=US (haven´t found them in the JDK docs). Haven´t tested that for this context - so just for your info.

Using your own classes inside <script>

When you use <script language="javascript"> you are using the Java API. Ok so far. It´s simple to use java.io.File for getting information about a particular file or creating complex strings with java.util.StringBuffer. But there are two problems:

- How to import other (non java.*) classes, e.g. Ant´s own classes?
- How to import classes which are **not** on Ant´s classpath?

How to import other (non java.*) classes, e.g. Ant´s own classes?

The answer to this is described on the homepage of the javascript interpreter, but very hidden (I think). Following the links "Documentation" and "Scripting Java" you´ll get some examples. Inside them it is written: *"If you wish to load classes from JavaScript that aren't in the java package, you'll need to prefix the package name with "*Packages*". For example ..."*. Transfered to you script task that would be:

```
importClass(Packages.org.apache.tools.ant.types.Path);
```

I'm confused. I was told that Java (static typing, compiled to bytecodes, etc.) is a totally different language than JavaScript (dynamic typing, interpreted, etc.). Are we using Java for scripting here ? Or are we actually using JavaScript ? Or both ?

AFAICT, the scripting language is EmcaScript (aka JavaScript), but a java binding layer was added so that the java script can create and manuipulate java objects.

How to import classes which are "'not"' on Ant´s classpath?

Ok, now we can use Ant´s classes and the whole javax.*-stuff. But if I need some custom classes? For example I have to create a reverse sorted list of files. Getting the list of files is no problem. Sorting can be done by java.util.TreeSet. But I need a custom Comparator which does the specific comparison. So I implement one and store it in ${basedir}/ext.

And now the trick: Ant´s Project class provides methods for getting classloader. And there is one which includes specified paths. So we

- get the *ext* directory
- create a <path> object
- get the classloader
- load the class
- instantiate that

{{{ <script language="javascript"> <![CDATA[

    importClass(java.util.TreeSet); importClass(Packages.org.apache.tools.ant.types.Path); loaderpath = new Path(project, project.getProperty("ext. dir")); classloader = project.createClassLoader(loaderpath);

    comparatorClass = classloader.loadClass("ReportDateComparator"); comparator = comparatorClass.newInstance();

    list = new TreeSet(comparator);

    ]]></script> }}}

Jan Matèrne

Writing a "Task" for getting the dependency list for a target

That was the question I was asked recently on jGuru. That was a nice question blocked URL The final result is

{{{ <macrodef name="dep">

<attribute name="root"/> <attribute name="file" default="@{root}.dep"/> <sequential> <script language="javascript"> <![CDATA[

// attribute expansion from macrodef (script can not reach the values) root = "@{root}"; filename = "@{file}"; // for collecting the informations lineSep = project.getProperty("line.separator");

list = new java.lang.StringBuffer(); file = new java.io.File(filename); // get all targets of the current project targets = project.getTargets(); // get the dependencies - including not executing targets sorted = project.topoSort(root, targets); // create the list and break if we have "executed" our root target // this code is adapted from Project.executeTarget(String) for(it=sorted.iterator(); it.hasNext();) {

curTarget = it.next(); list.append(curTarget).append(lineSep); if (curTarget.getName().equals(root)) {
break;
}
} // create the file echo = project.createTask("echo"); echo.setMessage(list.toString()); echo.setFile(file); echo.perform();

]]></script> </sequential>

</macrodef>

<target name="dep">

<dep root="build"/> <dep root="dist-lite" file="dist_lite.txt"/>

</target> }}}

Jan Matèrne

Using Ant to download files and check their integrity

While downloading the Milestone 4 of Eclipse I got an idea: why should I download all the files without knowing if there are corrupt? Ok, the following scenario:

- define a list with all names of the files to be downloaded
- download the file and its MD5 check file
- compute the MD5 checksum for the downloaded file
- compare that value with the one stored in the MD5-file

And for faster handling:

- don´t download files if there are already downloaded and valid
- don´t check files multiple times if they are valid

So I defined some properties in check-downloads.properties:

- *download.zip.dir*: remote directory containing the zip files (ftp://download2.eclipse.org/S-3.0M4-200310101454)
- *download.md5.dir*: remote directory containing the MD5 files (http://download2.eclipse.org/downloads/drops/S-3.0M4-200310101454/checksum)
- *dest.dir*: local directory for storing the files (.)
- *file.list*: comma separated list of zip files to download (eclipse-Automated-Tests-3.0M4.zip,eclipse-examples-3.0M4-win32.zip,...)
- *proxy.host*: proxy settings
- *proxy.port*: proxy settings

And the final buildfile is: {{{ <project default="main">

<taskdef resource="net/sf/antcontrib/antcontrib.properties"/>

<property name="result.file" value="check-downloads-results.properties"/> <property file="check-downloads.properties"/> <property file="${result.file}"/>

<target name="main">

<setproxy proxyHost="${proxy.host}" proxyPort="${proxy.port}"/> <foreach list="${file.list}" param="file" target="checkFile"/>

</target>

<target name="checkFile" depends="check.download,check.md5-1,check.md5-2" if="file"/>

```
<target name="check.init">

    <property name="zip.file" value="${file}"/> <property name="md5.file" value="${file}.md5"/> <condition property="md5-ok"><isset
    property="${zip.file}.isValid"/></condition> <condition property="download-ok">

        <and>

            <available file="${dest.dir}/${zip.file}"/> <available file="${dest.dir}/${md5.file}"/>

        </and>

    </condition>

</target>

<target name="check.download" unless="download-ok" depends="check.init">

    <echo>Download ${md5.file}</echo> <get src="${download.md5.dir}/${md5.file}" dest="${dest.dir}/${md5.file}"/> <echo>Download ${zip.
    file}</echo> <get src="${download.zip.dir}/${zip.file}" dest="${dest.dir}/${zip.file}"/>

</target>

<target name="check.md5-1" if="md5-ok" depends="check.init">

    <echo>${zip.file}: just processed</echo>

</target>

<target name="check.md5-2" unless="md5-ok" depends="check.init">

    <trycatch><try>

        <!-- what is the valid md5 value specified in the md5 file --> <loadfile srcFile="${md5.file}" property="md5.valid">

            <filterchain>

                <striplinebreaks/> <tokenfilter>

                    <stringtokenizer/> <replaceregex pattern="${zip.file}" replace=/>

                </tokenfilter> <tokenfilter>

                    <trim/>

                </tokenfilter>

            </filterchain>

        </loadfile> <!-- what is the actual md5 value --> <checksum file="${zip.file}" property="md5.actual"/> <!-- compare them --> <con
        dition property="md5.isValid">

            <equals arg1="${md5.valid}" arg2="${md5.actual}"/>

        </condition> <property name="md5.isValid" value="false"/> <!-- print the result --> <if>

            <istrue value="${md5.isValid}"/> <then>

                <echo>${zip.file}: ok</echo> <echo file="${result.file}"

                    append="true"
                    message="${zip.file}.isValid=true${line.separator}"/>

            </then> <else>

                <echo>${zip.file}: Wrong MD5 checksum !!!</echo> <echo>- expected: ${md5.valid}</echo> <echo>- actual :
                ${md5.actual}</echo> <move file="${zip.file}" tofile="${zip.file}.wrong-checksum"/>

            </else>

        </if>

    </try><catch/></trycatch>

</target>

</project> }}}
```

I got very nice results when starting with -quiet mode.

Jan Matèrne

Windows XP exec task : use os="Windows XP" instead of "os="Windows NT" despite docs

using ant 1.6.1 with j2sdk1.4.2

Doc incorrectly says you should use os="Windows NT" for the exec task to lauch a batch file

If you try to launch a batch file with 'exec' task, you should try to use instead :

```
    <exec dir="bat" executable="cmd" os="Windows XP" failonerror="true">
        <arg line="/c somebatch.bat"/>
    </exec>
```

If you encounter such a problem, try a verbose execution:

ant - verbose build.xml

this should give you these results: (I've just tailored an adapted build.xml)

```
XP_as_NT:
     [echo] batch.bat creation with content : '/c dir ..\ >dirNT.txt'
     [echo] exec executable='cmd' os='Windows NT' failonerror='true'
     [exec] Current OS is Windows XP
     [exec] This OS, Windows XP was not found in the specified list of valid OSes: Windows NT
[available] Unable to find dirNT.txt to set property Success.XP_as_NT
Property ${Success.XP_as_NT} has not been set
     [echo] Success.XP_as_NT : ${Success.XP_as_NT}

BUILD SUCCESSFUL
Total time: 3 seconds
```

Compared to this one :

```
XP_as_XP:
     [echo] Batch file creation with content : '/c dir ..\ >dirXP.txt'
     [echo] exec executable='cmd' os='Windows XP' failonerror='true'
     [exec] Current OS is Windows XP
     [exec] Executing 'cmd' with arguments:
     [exec] '/c'
     [exec] 'batchXP.bat'
     [exec]
     [exec] The ' characters around the executable and arguments are
     [exec] not part of the command.

     [exec] F:\trucking\ant\bat>dir ..\  1>dirXP.txt


[available] Found: dirXP.txt in F:\trucking\ant\bat
     [echo] Success.XP_as_XP : true

BUILD SUCCESSFUL
Total time: 3 seconds
```

Sadly enough, the env variable OS on Windows XP says : Windows_NT, and doesn't reflect what you see on these verbose execution.

The only simple way to determine os seems then with the windir env variable; which happens to be <somedrive>\WINNT on windows NT and <somedrive>\Windows on windows XP

You may test it that way:

```
<project name="YourProject">

<property environment ="env"/>

<target name="init">
    <condition property="osIsXP">
        <equals arg1="${env.HOMEDRIVE}\WINDOWS"
                arg2="${env.windir}"/>
    </condition>
</target>
```

Marc Persuy

A potential problem with this solution arises when Windows XP is not installed in the folder "WINDOWS". While it usually resides there, it can be installed in a folder with any name. Also, upgrading from Windows 2000 keeps the default installation directory that 2000 used, "WINNT". Thus, the folder name alone can't guarantee that you're working with XP.

Ryan Stinnett

Implementing a PreProcessor

On Bug-28738 is a question about preprocessor. You can do that without any external tasks (but I think the external tasks are more comfortable than this way blocked URL

The example java file is the common known HelloWorld:

```
public class HelloWorld {

    public static void main(String arg[]){
        //@debug start
        log("debug:let say this is a debug code or logging");
        //@debug end

        System.out.println("HELLO WORLD");
    }

    //@debug start
    private static void log(String msg) {
        System.out.println("LOG: " + msg);
    }
    //@debug end
}
```

We can use the <replaceregexp> for deleting the code between debug-start and -end statements:

```
    <replaceregexp match="//@debug start.*?//@debug end" replace="" flags="gs">
```

Important is the question mark in the match clause (.*?), so we´ve got minimal pattern matching. Otherwise all between the first debug-start mark and the very last debug-end will be selected. And therefore we will lose important code segments blocked URL The flag "s" is responsible that we will get the whole file at once and the "g" that we catch all debug-statements.

After running that command we´ll get that:

```
public class HelloWorld {

    public static void main(String arg[]){


        System.out.println("HELLO WORLD");
    }


}
```

Maybe you will run a code beautyfier on that so you´ll delete the empty lines ... But you can see that the expected code is generated. And the class file is beautified blocked URL

Here the buildfile for the example. Place that in a project root directory and the original java source in "src" subdirectory.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project>
    <echo> ************* Debug Mode ************* </echo>
    <delete dir="classes"/>
    <mkdir dir="classes"/>
    <javac srcdir="src" destdir="classes"/>
    <java classname="HelloWorld" classpath="classes"/>

    <echo> ************* LIVE Mode ************* </echo>
    <delete dir="classes"/>
    <delete dir="src-live"/>
    <mkdir dir="classes"/>
    <copy todir="src-live">
        <fileset dir="src"/>
    </copy>

    <!-- important is the .*? because .* matches too much -->
    <replaceregexp match="//@debug start.*?//@debug end" replace="" flags="gs">
        <fileset dir="src-live"/>
    </replaceregexp>

    <javac srcdir="src-live" destdir="classes"/>
    <java classname="HelloWorld" classpath="classes"/>
</project>
```

The output would be:

```
Buildfile: build.xml
     [echo]  ************* Debug Mode *************
   [delete] Deleting directory C:\tmp\anttests\preprocessor\classes
    [mkdir] Created dir: C:\tmp\anttests\preprocessor\classes
    [javac] Compiling 1 source file to C:\tmp\anttests\preprocessor\classes
     [java] LOG: debug:let say this is a debug code or logging
     [java] HELLO WORLD
     [echo]  ************* LIVE Mode *************
   [delete] Deleting directory C:\tmp\anttests\preprocessor\classes
   [delete] Deleting directory C:\tmp\anttests\preprocessor\src-live
    [mkdir] Created dir: C:\tmp\anttests\preprocessor\classes
     [copy] Copying 1 file to C:\tmp\anttests\preprocessor\src-live
    [javac] Compiling 1 source file to C:\tmp\anttests\preprocessor\classes
     [java] HELLO WORLD

BUILD SUCCESSFUL
Total time: 3 seconds
```

Compounding a property name from the instantiations of multiple previously instanced properties

The Problem

Although plain Ant sytax does not allow one to do so, a simple macrodef can derive property names by pasting together the instantiations of multiple previously instanced properties.

Given a set of resource-like properties such as:

```
driver.bsd="SomeBSDDriver"
driver.os2="A.Real.Old.Driver"
driver.windows="GPFGalore"

booter.bsd="boot"
booter.os2="boot.sys"
booter.windows="ntldr"
```

You might wish to pass to some target or task properties/parameters such as ${component} and ${targetOS} in the form

```
<do-something-with object="${${component}.${targetOS}}/>
```

so that if, for example, ${component} were valued as "driver" and ${targetOS} were valued as "os2", the value of ${${component}.${targetOS}} would be the expansion of ${driver.os2}, i.e., "A.Real.Old.Driver". However, Ant expansions of property instantiations are not recursive. So in this instance the expansion of ${${component}.${targetOS}} is not "A.Real.Old.Driver" but instead undefined and possibly varying by Ant version. (Ant 1.6.1 would yield a literal value of ${${component}.${targetOS}} while Ant 1.7 alpha currently yields ${${component}.os2})

Solution Macrodef

Define a macro allowing us to express the problem case as:

```
<macro.compose-property name="object" stem="${component} selector="${targetOS}"/>
<do-something-with object="${object}"/>
```

Here is the macro (along lines suggested by Peter Reilly with reference to http://ant.apache.org/faq.html#propertyvalue-as-name-for-property):

```
<!-- Allows you define a new property with a value of ${${a}.${b}} which can't be done by the Property task
alone.  -->
    <macrodef name="macro.compose-property">
        <attribute name="name"/>
        <attribute name="stem"/>
        <attribute name="selector"/>
        <sequential>
            <property name="@{name}" value="${@{stem}.@{selector}}"/>
        </sequential>
    </macrodef>
```

Why does <javac> require you to set source="1.4" for Java1.4 features, "1.5" for Java1.5?

The command line javac tool automatically selects the latest version of Java for the platform you build on. So when you build on Java1.4, it is as if -source 1.4 was passed in, while on Java1.5, you get the effect of -source 1.5

Yet on Ant, you have to explicitly say what version of the Java language you want to build against. If you have the assert keyword in source and omit the source="1.4" attribute, the compile will break. If you use fancy Java1.5 stuff and forget source="1.5", you get errors when you hit enums, attributes, generics or the new for stuff.

Q. Why is this the case? Why doesnt Ant "do the right thing?"

A. Because of a deliberate policy decision by the project.

Ant was written, first and foremost, to build open source projects. In such a project, you don't know who builds your project; you just give out your source and rely on it compiling everywhere, regardless of which platform or JVM the person at the far end used. We also assume that a source tarball will still compile, years into the future.

If <javac> automatically selected the latest version of Java, then any Java1.2 code that used "enum" as a variable name, or "assert". To build the old projects, you would have to edit every build file that didnt want to use the latest Java version and force in a source="1.3" attribute. Except if they were other people's projects, you would have to struggle to get that change committed, and things like ApacheGump would never work, because all old-JVM projects would never compile straight from the SCM repository.

That is why, in Ant, if you want the latest and greatest features of the Java language, you have to ask for them. By doing so you are placing a declaration in your build file what language version you want to use, both now and into the future.

An interesting follow-on question is then "why does javac on the command line risk breaking every makefile-driven Java project by automatically updating Java language versions unless told not to?" That is for Sun to answer, not the Ant team.