Table of Contents

Apache Ant 1.8.1 Manual

This is the manual for version 1.8.1 of Apache Ant. If your version of Ant (as verified with `ant -version`) is older or newer than this version then this is not the correct manual set. Please use the documentation appropriate to your current version. Also, if you are using a version older than the most recent release, we recommend an upgrade to fix bugs as well as provide new functionality.

Introduction

Apache Ant is a Java-based build tool. In theory, it is kind of like *make*, without *make*'s wrinkles.

Why?

Why another build tool when there is already *make*, *gnumake*, *nmake*, *jam*, and others? Because all those tools have limitations that Ant's original author couldn't live with when developing software across multiple platforms. Make-like tools are inherently shell-based: they evaluate a set of dependencies, then execute commands not unlike what you would issue on a shell. This means that you can easily extend these tools by using or writing any program for the OS that you are working on; however, this also means that you limit yourself to the OS, or at least the OS type, such as Unix, that you are working on.

Makefiles are inherently evil as well. Anybody who has worked on them for any time has run into the dreaded tab problem. "Is my command not executing because I have a space in front of my tab?!!" said the original author of Ant way too many times. Tools like Jam took care of this to a great degree, but still have yet another format to use and remember.

Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface.

Granted, this removes some of the expressive power that is inherent in being able to construct a shell command such as `` `find . -name foo -exec rm {}` ``, but it gives you the ability to be cross-platform--to work anywhere and everywhere. And hey, if you really need to execute a shell command, Ant has an `<exec>` task that allows different commands to be executed based on the OS it is executing on.

Table of Contents

Installing Ant

Table of Contents

Using Ant

Table of Contents

Running Ant

Table of Contents

Ant Tasks

Table of Contents

Custom Components

Listeners & Loggers

Overview

Ant has two related features to allow the build process to be monitored: listeners and loggers.

Listeners

A listener is alerted of the following events:

- build started
- build finished
- target started
- target finished
- task started
- task finished
- message logged

These are used internally for various recording and housekeeping operations, however new listeners may registered on the command line through the `-listener` argument.

Loggers

Loggers extend the capabilities of listeners and add the following features:

- Receives a handle to the standard output and error print streams and therefore can log information to the console or the `-logfile` specified file.
- Logging level (-quiet, -verbose, -debug) aware
- Emacs-mode aware

Built-in Listeners/Loggers

Classname	Description	Type
`org.apache.tools.ant.DefaultLogger`	The logger used implicitly unless overridden with the `-logger` command-line switch.	BuildLogger
`org.apache.tools.ant.NoBannerLogger`	This logger omits output of empty target output.	BuildLogger
`org.apache.tools.ant.listener.MailLogger`	Extends DefaultLogger such that output is still generated the same, and when the build is finished an e-mail can be sent.	BuildLogger
`org.apache.tools.ant.listener.AnsiColorLogger`	Colorifies the build output.	BuildLogger
`org.apache.tools.ant.listener.Log4jListener`	Passes events to Log4j for highly customizable logging.	BuildListener

org.apache.tools.ant.XmlLogger	Writes the build information to an XML file.	BuildLogger
org.apache.tools.ant.TimestampedLogger	Prints the time that a build finished	BuildLogger
org.apache.tools.ant.listener.BigProjectLogger	Prints the project name every target	BuildLogger
org.apache.tools.ant.listener.ProfileLogger	The default logger, with start times, end times and durations added for each task and target.	BuildLogger

DefaultLogger

Simply run Ant normally, or:

```
ant -logger org.apache.tools.ant.DefaultLogger
```

NoBannerLogger

Removes output of empty target output.

```
ant -logger org.apache.tools.ant.NoBannerLogger
```

MailLogger

The MailLogger captures all output logged through DefaultLogger (standard Ant output) and will send success and failure messages to unique e-mail lists, with control for turning off success or failure messages individually.

Properties controlling the operation of MailLogger:

Property	Description	Required
MailLogger.mailhost	Mail server to use	No, default "localhost"
MailLogger.port	SMTP Port for the Mail server	No, default "25"
MailLogger.user	user name for SMTP auth	Yes, if SMTP auth is required on your SMTP server the email message will be then sent using Mime and requires JavaMail
MailLogger.password	password for SMTP auth	Yes, if SMTP auth is required on your SMTP server the email message will be then sent using Mime and requires JavaMail
MailLogger.ssl	on or true if ssl is needed This feature requires JavaMail	no
MailLogger.from	Mail "from" address	Yes, if mail needs to be sent
MailLogger.replyto	Mail "replyto" address(es), comma-separated	No
MailLogger.failure.notify	Send build failure e-mails?	No, default "true"
MailLogger.success.notify	Send build success e-mails?	No, default "true"

MailLogger.failure.to	Address(es) to send failure messages to, comma-separated	Yes, if failure mail is to be sent
MailLogger.success.to	Address(es) to send success messages to, comma-separated	Yes, if success mail is to be sent
MailLogger.failure.subject	Subject of failed build	No, default "Build Failure"
MailLogger.success.subject	Subject of successful build	No, default "Build Success"
MailLogger.failure.body	Fixed body of the email for a failed build. *Since Ant 1.8.0*	No, default is to send the full log output.
MailLogger.success.body	Fixed body of the email for a successful build. *Since Ant 1.8.0*	No, default is to send the full log output.
MailLogger.mimeType	MIME-Type of the message. *Since Ant 1.8.0*	No, default is text/plain
MailLogger.charset	Character set of the message. *Since Ant 1.8.0*	No
MailLogger.starttls.enable	on or true if STARTTLS should be supported (requires JavaMail). *Since Ant 1.8.0*	No, default is false
MailLogger.properties.file	Filename of properties file that will override other values.	No

```
ant -logger org.apache.tools.ant.listener.MailLogger
```

AnsiColorLogger

The AnsiColorLogger adds color to the standard Ant output by prefixing and suffixing ANSI color code escape sequences to it. It is just an extension of DefaultLogger and hence provides all features that DefaultLogger does.

AnsiColorLogger differentiates the output by assigning different colors depending upon the type of the message.

If used with the -logfile option, the output file will contain all the necessary escape codes to display the text in colorized mode when displayed in the console using applications like cat, more, etc.

This is designed to work on terminals that support ANSI color codes. It works on XTerm, ETerm, Win9x Console (with ANSI.SYS loaded.), etc.

NOTE: It doesn't work on WinNT and successors, even when a COMMAND.COM console loaded with ANSI.SYS is used.

If the user wishes to override the default colors with custom ones, a file containing zero or more of the custom color key-value pairs must be created. The recognized keys and their default values are shown below:

```
AnsiColorLogger.ERROR_COLOR=2;31
AnsiColorLogger.WARNING_COLOR=2;35
AnsiColorLogger.INFO_COLOR=2;36
AnsiColorLogger.VERBOSE_COLOR=2;32
AnsiColorLogger.DEBUG_COLOR=2;34
```

Each key takes as value a color combination defined as **Attribute;Foreground;Background**. In the above example, background value has not been used.

This file must be specfied as the value of a system variable named ant.logger.defaults and passed as an argument using the -D option to the **java** command that invokes the Ant application. An easy way to achieve this is to add -Dant.logger.defaults= */path/to/your/file* to the ANT_OPTS environment variable. Ant's launching script recognizes this flag and will pass it to the java command appropriately.

Format:

```
AnsiColorLogger.*=Attribute;Foreground;Background

Attribute is one of the following:
0 -> Reset All Attributes (return to normal mode)
1 -> Bright (Usually turns on BOLD)
2 -> Dim
3 -> Underline
5 -> link
7 -> Reverse
8 -> Hidden

Foreground is one of the following:
30 -> Black
31 -> Red
32 -> Green
33 -> Yellow
34 -> Blue
35 -> Magenta
36 -> Cyan
37 -> White

Background is one of the following:
40 -> Black
41 -> Red
42 -> Green
43 -> Yellow
44 -> Blue
45 -> Magenta
46 -> Cyan
47 -> White
```

```
ant -logger org.apache.tools.ant.listener.AnsiColorLogger
```

Log4jListener

Passes build events to Log4j, using the full classname's of the generator of each build event as the category:

- build started / build finished - org.apache.tools.ant.Project
- target started / target finished - org.apache.tools.ant.Target
- task started / task finished - the fully qualified classname of the task
- message logged - the classname of one of the above, so if a task logs a message, its classname is the category used, and so on.

All start events are logged as INFO. Finish events are either logged as INFO or ERROR depending on whether the build failed during that stage. Message events are logged according to their Ant logging level, mapping directly to a corresponding Log4j level.

```
ant -listener org.apache.tools.ant.listener.Log4jListener
```

To use Log4j you will need the Log4j JAR file and a 'log4j.properties' configuration file. Both should be placed somewhere in your Ant classpath. If the log4j.properties is in your project root folder you can add this with *-lib* option:

```
ant -listener org.apache.tools.ant.listener.Log4jListener -lib .
```

If, for example, you wanted to capture the same information output to the console by the DefaultLogger and send it to a file named 'build.log', you could use the following configuration:

```
log4j.rootLogger=ERROR, LogFile
log4j.logger.org.apache.tools.ant.Project=INFO
log4j.logger.org.apache.tools.ant.Target=INFO
log4j.logger.org.apache.tools.ant.taskdefs=INFO
log4j.logger.org.apache.tools.ant.taskdefs.Echo=WARN

log4j.appender.LogFile=org.apache.log4j.FileAppender
log4j.appender.LogFile.layout=org.apache.log4j.PatternLayout
log4j.appender.LogFile.layout.ConversionPattern=[%6r] %8c{1} : %m%n
log4j.appender.LogFile.file=build.log
```

For more information about configuring Log4J see its documentation page.

XmlLogger

Writes all build information out to an XML file named log.xml, or the value of the `XmlLogger.file` property if present, when used as a listener. When used as a logger, it writes all output to either the console or to the value of `-logfile`. Whether used as a listener or logger, the output is not generated until the build is complete, as it buffers the information in order to provide timing information for task, targets, and the project.

By default the XML file creates a reference to an XSLT file "log.xsl" in the current directory; look in ANT_HOME/etc for one of these. You can set the property `ant.XmlLogger.stylesheet.uri` to provide a uri to a style sheet. this can be a relative or absolute file path, or an http URL. If you set the property to the empty string, "", no XSLT transform is declared at all.

```
ant -listener org.apache.tools.ant.XmlLogger
ant -logger org.apache.tools.ant.XmlLogger -verbose -logfile build_log.xml
```

TimestampedLogger

Acts like the default logger, except that the final success/failure message also includes the time that the build completed. For example:

```
  BUILD SUCCESSFUL - at 16/08/05 16:24
```

To use this listener, use the command:

```
ant -logger org.apache.tools.ant.listener.TimestampedLogger
```

BigProjectLogger

This logger is designed to make examining the logs of a big build easier, especially those run under continuous integration tools. It

1. When entering a child project, prints its name and directory
2. When exiting a child project, prints its name
3. Includes the name of the project when printing a target
4. Omits logging the names of all targets that have no direct task output
5. Includes the build finished timestamp of the TimeStamp logger

This is useful when using <subant> to build a large project from many smaller projects -the output shows which particular project is building. Here is an example in which "clean" is being called on all a number of child projects, only some of which perform work:

```
=======================================================================
Entering project "xunit"
In /home/ant/components/xunit
=======================================================================

xunit.clean:
   [delete] Deleting directory /home/ant/components/xunit/build
   [delete] Deleting directory /home/ant/components/xunit/dist

=======================================================================
Exiting project "xunit"
=======================================================================

=======================================================================
Entering project "junit"
In /home/ant/components/junit
```

```
=======================================================================

=======================================================================
Exiting project "junit"
=======================================================================
```

The entry and exit messages are very verbose in this example, but in a big project compiling or testing many child components, the messages are reduced to becoming clear delimiters of where different projects are in charge -or more importantly, which project is failing.

To use this listener, use the command:

```
ant -logger org.apache.tools.ant.listener.BigProjectLogger
```

ProfileLogger

This logger stores the time needed for executing a task, target and the whole build and prints these information. The output contains a timestamp when entering the build, target or task and a timestamp and the needed time when exiting.

since Ant 1.8.0

Example

Having that buildfile

```
<project>
    <target name="aTarget">
        <echo>echo-task</echo>
        <zip destfile="my.zip">
            <fileset dir="${ant.home}"/>
        </zip>
    </target>
    <target name="anotherTarget" depends="aTarget">
        <echo>another-echo-task</echo>
    </target>
</project>
```

and executing with `ant -logger org.apache.tools.ant.listener.ProfileLogger anotherTarget` gives that output (with other timestamps and duration of course ;) :

```
Buildfile: ...\build.xml

Target aTarget: started Thu Jan 22 09:01:00 CET 2009

echo: started Thu Jan 22 09:01:00 CET 2009
     [echo] echo-task

echo: finishedThu Jan 22 09:01:00 CET 2009 (250ms)

zip: started Thu Jan 22 09:01:00 CET 2009
      [zip] Building zip: ...\my.zip

zip: finishedThu Jan 22 09:01:01 CET 2009 (1313ms)

Target aTarget: finishedThu Jan 22 09:01:01 CET 2009 (1719ms)

Target anotherTarget: started Thu Jan 22 09:01:01 CET 2009

echo: started Thu Jan 22 09:01:01 CET 2009
     [echo] another-echo-task

echo: finishedThu Jan 22 09:01:01 CET 2009 (0ms)

Target anotherTarget: finishedThu Jan 22 09:01:01 CET 2009 (0ms)

BUILD SUCCESSFUL
Total time: 2 seconds
```

Writing your own

See the Build Events section for developers.

Notes:

- A listener or logger should not write to standard output or error in the `messageLogged() method`; Ant captures these internally and it will trigger an infinite loop.
- Logging is synchronous; all listeners and loggers are called one after the other, with the build blocking until the output is processed. Slow logging means a slow build.
- When a build is started, and `BuildListener.buildStarted(BuildEvent event)` is called, the project is not fully functional. The build has started, yes, and the event.getProject() method call returns the Project instance, but that project is initialized with JVM and ant properties, nor has it parsed the build file yet. You cannot call `Project.getProperty()` for property lookup, or `Project.getName()` to get the project name (it will return null).
- Classes that implement `org.apache.tools.ant.SubBuildListener` receive notifications when child projects start and stop.

Table of Contents

IDE Integration

All the modern Java IDEs support Ant almost out of the box.

- AntRunner For JBuilder (unbundled)
- AntWork Plugin for the Jext Java Text Editor (unbundled)
- JDEE (Java Development Environment for Emacs) has built-in text ANT integration: selection of target through text field, execution, hyperlink to compilation errors. Installation: built-in JDEE 2.2.8 or later. Configuration: through customize menu "Jde Build Function"
- IDEA has built-in GUI ANT integration: GUI selection of targets, execution, hyperlink to compilation errors
- NetBeans NetBeans IDE uses Ant as the basis for its project system starting with the 4.0 release.
- jEdit jEdit is an open source java IDE with some great plugins for Java dev, a good XML editor and the Antfarm plugin to execute targets in a build file.
- Eclipse Eclipse is IBM's counterpoint to NetBeans; an open source IDE with Java and Ant support.
- Virtual Ant GUI allows you to work on a Virtual File System without dealing with the XML. Plugs into Eclipse, Netbeans & Intellij.
- WebSphere Studio Application Developer
- JBuilder 9 Personal JBuilder supports Ant with the following features. Add Ant nodes to projects and execute Ant targets from within JBuilder. Add custom Ant-based build tasks with custom Ant libraries to run Ant from within JBuilder. Rapid navigation from Ant build error messages to source files. Customize build menu and toolbar with custom build targets.

Table of Contents

Developing with Ant

Tutorials

Table of Contents

Tutorials

Apache Ant API has not been generated

If you see this page online at ant.apache.org, it is not a bug, but on purpose. We do not provide an online version of the API docs, they are included with all our distributions.

```
/*
 *                                 Apache License
 *                           Version 2.0, January 2004
 *                        http://www.apache.org/licenses/
 *
 *   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
 *
 *   1. Definitions.
 *
 *      "License" shall mean the terms and conditions for use, reproduction,
 *      and distribution as defined by Sections 1 through 9 of this document.
 *
 *      "Licensor" shall mean the copyright owner or entity authorized by
 *      the copyright owner that is granting the License.
 *
 *      "Legal Entity" shall mean the union of the acting entity and all
 *      other entities that control, are controlled by, or are under common
 *      control with that entity. For the purposes of this definition,
 *      "control" means (i) the power, direct or indirect, to cause the
 *      direction or management of such entity, whether by contract or
 *      otherwise, or (ii) ownership of fifty percent (50%) or more of the
 *      outstanding shares, or (iii) beneficial ownership of such entity.
 *
 *      "You" (or "Your") shall mean an individual or Legal Entity
 *      exercising permissions granted by this License.
 *
 *      "Source" form shall mean the preferred form for making modifications,
 *      including but not limited to software source code, documentation
 *      source, and configuration files.
 *
 *      "Object" form shall mean any form resulting from mechanical
 *      transformation or translation of a Source form, including but
 *      not limited to compiled object code, generated documentation,
 *      and conversions to other media types.
 *
 *      "Work" shall mean the work of authorship, whether in Source or
 *      Object form, made available under the License, as indicated by a
 *      copyright notice that is included in or attached to the work
 *      (an example is provided in the Appendix below).
 *
 *      "Derivative Works" shall mean any work, whether in Source or Object
 *      form, that is based on (or derived from) the Work and for which the
 *      editorial revisions, annotations, elaborations, or other modifications
 *      represent, as a whole, an original work of authorship. For the purposes
 *      of this License, Derivative Works shall not include works that remain
 *      separable from, or merely link (or bind by name) to the interfaces of,
 *      the Work and Derivative Works thereof.
 *
 *      "Contribution" shall mean any work of authorship, including
 *      the original version of the Work and any modifications or additions
 *      to that Work or Derivative Works thereof, that is intentionally
 *      submitted to Licensor for inclusion in the Work by the copyright owner
 *      or by an individual or Legal Entity authorized to submit on behalf of
```

* the copyright owner. For the purposes of this definition, "submitted"
* means any form of electronic, verbal, or written communication sent
* to the Licensor or its representatives, including but not limited to
* communication on electronic mailing lists, source code control systems,
* and issue tracking systems that are managed by, or on behalf of, the
* Licensor for the purpose of discussing and improving the Work, but
* excluding communication that is conspicuously marked or otherwise
* designated in writing by the copyright owner as "Not a Contribution."
*
* "Contributor" shall mean Licensor and any individual or Legal Entity
* on behalf of whom a Contribution has been received by Licensor and
* subsequently incorporated within the Work.
*
* 2. Grant of Copyright License. Subject to the terms and conditions of
* this License, each Contributor hereby grants to You a perpetual,
* worldwide, non-exclusive, no-charge, royalty-free, irrevocable
* copyright license to reproduce, prepare Derivative Works of,
* publicly display, publicly perform, sublicense, and distribute the
* Work and such Derivative Works in Source or Object form.
*
* 3. Grant of Patent License. Subject to the terms and conditions of
* this License, each Contributor hereby grants to You a perpetual,
* worldwide, non-exclusive, no-charge, royalty-free, irrevocable
* (except as stated in this section) patent license to make, have made,
* use, offer to sell, sell, import, and otherwise transfer the Work,
* where such license applies only to those patent claims licensable
* by such Contributor that are necessarily infringed by their
* Contribution(s) alone or by combination of their Contribution(s)
* with the Work to which such Contribution(s) was submitted. If You
* institute patent litigation against any entity (including a
* cross-claim or counterclaim in a lawsuit) alleging that the Work
* or a Contribution incorporated within the Work constitutes direct
* or contributory patent infringement, then any patent licenses
* granted to You under this License for that Work shall terminate
* as of the date such litigation is filed.
*
* 4. Redistribution. You may reproduce and distribute copies of the
* Work or Derivative Works thereof in any medium, with or without
* modifications, and in Source or Object form, provided that You
* meet the following conditions:
*
* (a) You must give any other recipients of the Work or
* Derivative Works a copy of this License; and
*
* (b) You must cause any modified files to carry prominent notices
* stating that You changed the files; and
*
* (c) You must retain, in the Source form of any Derivative Works
* that You distribute, all copyright, patent, trademark, and
* attribution notices from the Source form of the Work,
* excluding those notices that do not pertain to any part of
* the Derivative Works; and
*
* (d) If the Work includes a "NOTICE" text file as part of its

* distribution, then any Derivative Works that You distribute must
* include a readable copy of the attribution notices contained
* within such NOTICE file, excluding those notices that do not
* pertain to any part of the Derivative Works, in at least one
* of the following places: within a NOTICE text file distributed
* as part of the Derivative Works; within the Source form or
* documentation, if provided along with the Derivative Works; or,
* within a display generated by the Derivative Works, if and
* wherever such third-party notices normally appear. The contents
* of the NOTICE file are for informational purposes only and
* do not modify the License. You may add Your own attribution
* notices within Derivative Works that You distribute, alongside
* or as an addendum to the NOTICE text from the Work, provided
* that such additional attribution notices cannot be construed
* as modifying the License.
*
* You may add Your own copyright statement to Your modifications and
* may provide additional or different license terms and conditions
* for use, reproduction, or distribution of Your modifications, or
* for any such Derivative Works as a whole, provided Your use,
* reproduction, and distribution of the Work otherwise complies with
* the conditions stated in this License.
*
* 5. Submission of Contributions. Unless You explicitly state otherwise,
* any Contribution intentionally submitted for inclusion in the Work
* by You to the Licensor shall be under the terms and conditions of
* this License, without any additional terms or conditions.
* Notwithstanding the above, nothing herein shall supersede or modify
* the terms of any separate license agreement you may have executed
* with Licensor regarding such Contributions.
*
* 6. Trademarks. This License does not grant permission to use the trade
* names, trademarks, service marks, or product names of the Licensor,
* except as required for reasonable and customary use in describing the
* origin of the Work and reproducing the content of the NOTICE file.
*
* 7. Disclaimer of Warranty. Unless required by applicable law or
* agreed to in writing, Licensor provides the Work (and each
* Contributor provides its Contributions) on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
* implied, including, without limitation, any warranties or conditions
* of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
* PARTICULAR PURPOSE. You are solely responsible for determining the
* appropriateness of using or redistributing the Work and assume any
* risks associated with Your exercise of permissions under this License.
*
* 8. Limitation of Liability. In no event and under no legal theory,
* whether in tort (including negligence), contract, or otherwise,
* unless required by applicable law (such as deliberate and grossly
* negligent acts) or agreed to in writing, shall any Contributor be
* liable to You for damages, including any direct, indirect, special,
* incidental, or consequential damages of any character arising as a
* result of this License or out of the use or inability to use the
* Work (including but not limited to damages for loss of goodwill,

```
*     work stoppage, computer failure or malfunction, or any and all
*     other commercial damages or losses), even if such Contributor
*     has been advised of the possibility of such damages.
*
*   9. Accepting Warranty or Additional Liability. While redistributing
*     the Work or Derivative Works thereof, You may choose to offer,
*     and charge a fee for, acceptance of support, warranty, indemnity,
*     or other liability obligations and/or rights consistent with this
*     License. However, in accepting such obligations, You may act only
*     on Your own behalf and on Your sole responsibility, not on behalf
*     of any other Contributor, and only if You agree to indemnify,
*     defend, and hold each Contributor harmless for any liability
*     incurred by, or claims asserted against, such Contributor by reason
*     of your accepting any such warranty or additional liability.
*
*   END OF TERMS AND CONDITIONS
*
*   APPENDIX: How to apply the Apache License to your work.
*
*     To apply the Apache License to your work, attach the following
*     boilerplate notice, with the fields enclosed by brackets "[]"
*     replaced with your own identifying information. (Don't include
*     the brackets!)  The text should be enclosed in the appropriate
*     comment syntax for the file format. We also recommend that a
*     file or class name and description of purpose be included on the
*     same "printed page" as the copyright notice for easier
*     identification within third-party archives.
*
*   Copyright [yyyy] [name of copyright owner]
*
*   Licensed under the Apache License, Version 2.0 (the "License");
*   you may not use this file except in compliance with the License.
*   You may obtain a copy of the License at
*
*       http://www.apache.org/licenses/LICENSE-2.0
*
*   Unless required by applicable law or agreed to in writing, software
*   distributed under the License is distributed on an "AS IS" BASIS,
*   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
*   See the License for the specific language governing permissions and
*   limitations under the License.
*/
```

Feedback and Troubleshooting

If things do not work, especially simple things like `ant -version`, then something is wrong with your configuration. Before filing bug reports and emailing all the ant mailing lists

1. Check your environment variables. Are ANT_HOME and JAVA_HOME correct? If they have quotes or trailing slashes, remove them.
2. Unset CLASSPATH; if that is wrong things go horribly wrong. Ant does not need the CLASSPATH variable defined to anything to work.
3. Make sure there are no versions of crimson.jar or other XML parsers in JRE/ext
4. Is your path correct? is Ant on it? What about JDK/bin? have you tested this? If you are using Jikes, is it on the path? A createProcess error (especially with ID=2 on windows) usually means executable not found on the path.
5. Which version of ant are you running? Other applications distribute a copy -it may be being picked up by accident.
6. If a task is failing to run is optional.jar in ANT_HOME/lib? Are there any libraries which it depends on missing?
7. If a task doesn't do what you expect, run `ant -verbose` or `ant -debug` to see what is happening

If you can't fix your problem, start with the Ant User Mailing List . These are other ant users who will help you learn to use ant. If they cannot fix it then someone may suggest filing a bug report, which will escalate the issue. Remember of course, that support, like all open source development tasks, is voluntary. If you haven't invested time in helping yourself by following the steps above, it is unlikely that anyone will invest the time in helping you.

Also, if you don't understand something, the Ant User Mailing List is the place to ask questions. Not the developer list, nor the individuals whose names appears in the source and documentation. If they answered all such emails, nobody would have any time to improve ant.

To provide feedback on this software, please subscribe to the Ant User Mailing List

If you want to contribute to Ant or stay current with the latest development, join the Ant Development Mailing List

A searchable archive can be found at http://marc.theaimsgroup.com. Other archives will be documented online at Mailing Lists Archives

Apache Ant User Manual

by

- Stephane Bailliez (sbailliez@imediation.com)
- Nicola Ken Barozzi (nicolaken@apache.org)
- Jacques Bergeron (jacques.bergeron@dogico.com)
- Stefan Bodewig (stefan.bodewig@freenet.de)
- Patrick Chanezon (chanezon@netscape.com)
- James Duncan Davidson (duncan@x180.com)
- Tom Dimock (tad1@cornell.edu)
- Peter Donald (donaldp@apache.org)
- dIon Gillard (dion@apache.org)
- Erik Hatcher (ehatcher@apache.org)
- Diane Holt (holtdl@yahoo.com)
- Bill Kelly (bill.kelly@softwired-inc.com)
- Martijn Kruithof
- Arnout J. Kuiper (ajkuiper@wxs.nl)
- Antoine Lévy-Lambert
- Conor MacNeill
- Jan Matèrne
- Stefano Mazzocchi (stefano@apache.org)
- Erik Meade (emeade@geekfarm.org)
- Sam Ruby (rubys@us.ibm.com)
- Nico Seessle (nico@seessle.de)
- Jon S. Stevens (jon@latchkey.com)
- Wolf Siberski
- Magesh Umasankar
- Roger Vaughn (rvaughn@seaconinc.com)
- Dave Walend (dwalend@cs.tufts.edu)
- Phillip Wells (philwells@rocketmail.com)
- Christoph Wilhelms
- Craeg Strong (cstrong@arielpartners.com)

Version: 1.8.1

Table of Contents

Installing Ant

Getting Ant

Download Area Layout

Filename or Path	Description
KEYS	PGP-Keysfile. It contains the PGP-keys of Ant developers so you can 'trust' the distribution.
RELEASE-NOTES-{version}.html	Release notes of the given version in HTML format. When upgrading your Ant installation you should have a look at the *Changes that could break older environments* section.
ant-current-bin.zip	ZIP-Archive containing the compiled version of Ant in the last released version. This is the file most users will want to download.
ant-current-src.zip	ZIP-Archive containing the sources of Ant. If you have this you could compile Ant itself. If you do not have the *required* dependencies, the classes depeding on them are just not build.
ant-current-*.asc	Security file for checking the correctness of the zip file. This one is the PGP key.
ant-current-*.md5	Security file for checking the correctness of the zip file. This one is the MD5 key.
ant-current-*.sha1	Security file for checking the correctness of the zip file. This one is the SHA1 key.
antlibs/	This directory holds the Antlibs that are made of available by the Apache Ant project. Antlibs are bundles of Ant tasks that are not delivered as part of the Ant core but are available as optional downloads.
binaries/	The binaries directory holds specific Ant releases bundled in both ZIP and tar.gz compression formats. The named releases are in contrast to the ant-current-bin.zip file in the parent directory, which is always guaranteed to be the most current release of Ant.
common/	The common directory holds various files, such as the Apache License file that Ant is licensed under, that people may wish to examine without having to download the whole Ant distribution.
source/	The source directory holds the source code for specific Ant releases bundled in both ZIP and tar.gz compression formats. The named releases are in contrast to the ant-current-src.zip file in the parent directory, which is always guaranteed to hold the source code for the most current release of Ant.

Binary Edition

The latest stable version of Ant is available from the Ant web page http://ant.apache.org/.

As a binary in an RPM Package

Consult the jpackage section below.

Bundled in IDEs

All the main Java IDEs ship with Ant, products such as Eclipse, NetBeans and IntelliJ IDEA. If you install Ant this

way you usually get the most recent release of Ant at the time the IDE was released. Some of the IDEs (Eclipse and NetBeans in particular) ship with extra tasks that only work if IDE-specific tools are on Ant's path. To use these on command-line versions of Ant, the relevant JARs need to be added to the command-line Ant as extra libraries/tasks. Note that if it is an IDE task or extension that is not behaving, the Ant team is unable to field bug reports. Try the IDE mailing lists first, who will cross-file bugs if appropriate.

IDE's can invariably be pointed at different Ant installations. This lets developers upgrade to a new release of Ant, and eliminate inconsistencies between command-line and IDE Ant.

Bundled in Java applications

Many Java applications, most particularly application servers, ship with a version of Ant. These are primarily for internal use by the application, using the Java APIs to delegate tasks such as JSP page compilation to the Ant runtime. Such distributions are usually unsupported by everyone. Particularly troublesome are those products that non only ship with their own Ant release, they add their own version of ANT.BAT or ant.sh to the PATH. If Ant starts behaving wierdly after installing something, try the diagnostics advice.

Source Edition

If you prefer the source edition, you can download the source for the latest Ant release from http://ant.apache.org/srcdownload.cgi. If you prefer the leading-edge code, you can access the code as it is being developed via SVN. The Ant website has details on accessing SVN. All bug fixes will go in against the HEAD of the source tree, and the first response to many bugreps will be "have you tried the latest version". Don't be afraid to download and build a prererelease edition, as everything other than new features are usually stable.

See the section Building Ant on how to build Ant from the source code. You can also access the Ant SVN repository on-line.

System Requirements

Ant has been used successfully on many platforms, including Linux, commercial flavours of Unix such as Solaris and HP-UX, Windows NT-platforms, OS/2 Warp, Novell Netware 6, OpenVMS and MacOS X. The platforms used most for development are, in no particular order, Linux, MacOS X, Windows XP and Unix; these are therefore that platforms that tend to work best. As of Ant1.7, Windows 9x is no longer supported.

For the current version of Ant, you will also need a JDK installed on your system, version 1.4 or later required, 1.5 or later strongly recommended. The later the version of Java , the more Ant tasks you get.

Note: If a JDK is not present, only the JRE runtime, then many tasks will not work.

Note: Ant 1.8.* works with jdk1.4 and higher, Ant 1.7.* works with jdk1.3 and higher, Ant 1.6.* works with jdk 1.2 and higher, Ant 1.2 to Ant 1.5.* work with jdk 1.1 and higher.

Open Source Java Runtimes

The Ant team strongly supports users running Ant on Kaffe and other open source Java runtimes, and so strives to have a product that works well on those platforms. What appears to work well is Kaffe with Gnu Classpath and the Xerces and Xalan libraries.

Installing Ant

The binary distribution of Ant consists of the following directory layout:

```
ant
 +--- README, LICENSE, fetch.xml, other text files. //basic information
 +--- bin  // contains launcher scripts
 |
 +--- lib  // contains Ant jars plus necessary dependencies
 |
 +--- docs // contains documentation
 |       |
 |       +--- images  // various logos for html documentation
 |       |
 |       +--- manual  // Ant documentation (a must read ;-)
 |
 +--- etc // contains xsl goodies to:
           //   - create an enhanced report from xml output of various tasks.
           //   - migrate your build files and get rid of 'deprecated' warning
           //   - ... and more ;-)
```

Only the `bin` and `lib` directories are required to run Ant. To install Ant, choose a directory and copy the distribution files there. This directory will be known as ANT_HOME.

Windows 95, Windows 98 & Windows ME Note:
*On these systems, the script used to launch Ant will have problems if ANT_HOME is a long filename (i.e. a filename which is not of the format known as "8.3"). This is due to limitations in the OS's handling of the `"for"` batch-file statement. It is recommended, therefore, that Ant be installed in a **short**, 8.3 path, such as C:\Ant.*
On these systems you will also need to configure more environment space to cater for the environment variables used in the Ant lauch script. To do this, you will need to add or update the following line in the `config.sys` file `shell=c:\command.com c:\ /p /e:32768`

Setup

Before you can run Ant there is some additional set up you will need to do unless you are installing the RPM version from jpackage.org:

- Add the `bin` directory to your path.
- Set the `ANT_HOME` environment variable to the directory where you installed Ant. On some operating systems, Ant's startup scripts can guess `ANT_HOME` (Unix dialects and Windows NT/2000), but it is better to not rely on this behavior.
- Optionally, set the `JAVA_HOME` environment variable (see the Advanced section below). This should be set to the directory where your JDK is installed.

Note: Do not install Ant's ant.jar file into the lib/ext directory of the JDK/JRE. Ant is an application, whilst the extension directory is intended for JDK extensions. In particular there are security restrictions on the classes which may be loaded by an extension.

Windows Note:
The ant.bat script makes use of three environment variables - ANT_HOME, CLASSPATH and JAVA_HOME. **Ensure** that ANT_HOME and JAVA_HOME variables are set, and that they do **not** have quotes (either ' or ") and they do **not** end with \ or with /. CLASSPATH should be unset or empty.

Check Installation

You can check the basic installation with opening a new shell and typing `ant`. You should get a message like this

```
Buildfile: build.xml does not exist!
Build failed
```

So Ant works. This message is there because you need to write an individual buildfile for your project. With a `ant -version` you should get an output like

```
Apache Ant version 1.7.1 compiled on June 27 2008
```

If this does not work ensure your environment variables are set right. They must resolve to:

- required: %ANT_HOME%\bin\ant.bat
- optional: %JAVA_HOME%\bin\java.exe
- required: %PATH%=...*maybe-other-entries*...;%ANT_HOME%\bin;...*maybe-other-entries*...

ANT_HOME is used by the launcher script for finding the libraries. **JAVA_HOME** is used by the launcher for finding the JDK/JRE to use. (JDK is recommended as some tasks require the java tools.) If not set, the launcher tries to find one via the %PATH% environment variable. **PATH** is set for user convinience. With that set you can just start *ant* instead of always typing *the/complete/path/to/your/ant/installation/bin/ant*.

Optional Tasks

Ant supports a number of optional tasks. An optional task is a task which typically requires an external library to function. The optional tasks are packaged together with the core Ant tasks.

The external libraries required by each of the optional tasks is detailed in the Library Dependencies section. These external libraries must be added to Ant's classpath, in any of the following ways:

- In *`ANT_HOME`*`/lib`. This makes the JAR files available to all Ant users and builds.
- In `${user.home}/.ant/lib` (as of Ant 1.6). This allows different users to add new libraries to Ant. All JAR files added to this directory are available to command-line Ant.
- On the command line with a `-lib` parameter. This lets you add new JAR files on a case-by-case basis.
- In the `CLASSPATH` environment variable. Avoid this; it makes the JAR files visible to *all* Java applications, and causes no end of support calls. See below for details.
- In some `<classpath>` accepted by the task itself. For example, as of Ant 1.7.0 you can run the `<junit>` task without `junit.jar` in Ant's own classpath, so long as it is included (along with your program and tests) in the classpath passed when running the task.

  Where possible, this option is generally to be preferred, as the Ant script itself can determine the best path to load the library from: via relative path from the basedir (if you keep the library under version control with your project), according to Ant properties, environment variables, Ivy downloads, whatever you like.

IDEs have different ways of adding external JAR files and third-party tasks to Ant. Usually it is done by some configuration dialog. Sometimes JAR files added to a project are automatically added to ant's classpath.

The `CLASSPATH` environment variable

The CLASSPATH environment variable is a source of many Ant support queries. As the round trip time for diagnosis on the Ant user mailing list can be slow, and because filing bug reports complaining about 'ant.bat' not working will be rejected by the developers as WORKSFORME "this is a configuration problem, not a bug", you can save yourself a lot of time and frustration by following some simple steps.

1. Do not ever set CLASSPATH. Ant does not need it, it only causes confusion and breaks things.
2. If you ignore the previous rule, do not ever, ever, put quotes in the CLASSPATH, even if there is a space in a directory. This will break Ant, and it is not needed.
3. If you ignore the first rule, do not ever, ever, have a trailing backslash in a CLASSPATH, as it breaks Ant's ability to quote the string. Again, this is not needed for the correct operation of the CLASSPATH environment variable, even if a DOS directory is to be added to the path.
4. You can stop Ant using the CLASSPATH environment variable by setting the -noclasspath option on the command line. This is an easy way to test for classpath-related problems.

The usual symptom of CLASSPATH problems is that ant will not run with some error about not being able to find org.apache.tools.ant.launch.Launcher, or, if you have got the quotes/backslashes wrong, some very weird Java startup error. To see if this is the case, run ant -noclasspath or unset the CLASSPATH environment variable.

You can also make your Ant script reject this environment variable just by placing the following at the top of the script (or in an init target):

```
<property environment="env."/>
<property name="env.CLASSPATH" value=""/>
<fail message="Unset $CLASSPATH / %CLASSPATH% before running Ant!">
    <condition>
        <not>
            <equals arg1="${env.CLASSPATH}" arg2=""/>
        </not>
    </condition>
</fail>
```

Proxy Configuration

Many Ant built-in and third-party tasks use network connections to retrieve files from HTTP servers. If you are behind a firewall with a proxy server, then Ant needs to be configured with the proxy. Here are the different ways to do this.

- **With Java1.5**
  When you run Ant on Java1.5, you could try to use the automatic proxy setup mechanism with -autoproxy.
- **With explicit JVM properties.**
  These are documented by Sun, and control the proxy behaviour of the entire JVM. To set them in Ant, declare them in the ANT_OPTS environment variable. This is the best option for a non-mobile system. For a laptop, you have to change these settings as you roam.
- **In the build file itself**
  If you are writing an build file that is always to be used behind the firewall, the <setproxy> task lets you configure the proxy (which it does by setting the JVM properties). If you do this, we strongly recommend using ant properties to define the proxy host, port, etc, so that individuals can override the defaults.

The Ant team acknowledges that this is unsatisfactory. Until the JVM automatic proxy setup works properly everywhere, explicit JVM options via ANT_ARGS are probably the best solution. Setting properties on Ant's command line do not work, because those are *Ant properties* being set, not JVM options. This means the following does not set up the command line:

```
ant -Dhttp.proxyHost=proxy -Dhttp.proxyPort=81
```

All it does is set up two Ant properties.

One other troublespot with proxies is with authenticating proxies. Ant cannot go beyond what the JVM does here, and

as it is very hard to remotely diagnose, test and fix proxy-related problems, users who work behind a secure proxy will have to spend much time configuring the JVM properties until they are happy.

Windows and OS/2

Assume Ant is installed in `c:\ant\`. The following sets up the environment:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk-1.5.0.05
set PATH=%PATH%;%ANT_HOME%\bin
```

Linux/Unix (bash)

Assume Ant is installed in `/usr/local/ant`. The following sets up the environment:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.5.0.05
export PATH=${PATH}:${ANT_HOME}/bin
```

Linux/Unix (csh)

```
setenv ANT_HOME /usr/local/ant
setenv JAVA_HOME /usr/local/jdk/jdk-1.5.0.05
set path=( $path $ANT_HOME/bin )
```

Having a symbolic link set up to point to the JVM/JDK version makes updates more seamless.

RPM version from jpackage.org

The JPackage project distributes an RPM version of Ant. With this version, it is not necessary to set `JAVA_HOME` or `ANT_HOME` environment variables and the RPM installer will correctly place the Ant executable on your path.

NOTE: *Since Ant 1.7.0*, if the `ANT_HOME` environment variable is set, the jpackage distribution will be ignored.

Optional jars for the JPackage version are handled in two ways. The easiest, and best way is to get these external libraries from JPackage if JPackage has them available. (Note: for each such library, you will have to get both the external package itself (e.g. `oro-2.0.8-2jpp.noarch.rpm`) and the small library that links ant and the external package (e.g. `ant-apache-oro-1.6.2-3jpp.noarch.rpm`).

However, JPackage does not package proprietary software, and since some of the optional packages depend on proprietary jars, they must be handled as follows. This may violate the spirit of JPackage, but it is necessary if you need these proprietary packages. For example, suppose you want to install support for starteam, which jpackage does not support:

1. Decide where you want to deploy the extra jars. One option is in `$ANT_HOME/lib`, which, for JPackage is usually `/usr/share/ant/lib`. Another, less messy option is to create an `.ant/lib` subdirectory of your home directory and place your non-jpackage ant jars there, thereby avoiding mixing jpackage libraries with non-jpacakge stuff in the same folder. More information on where Ant finds its libraries is available here
2. Download a non-jpackage binary distribution from the regular Apache Ant site
3. Unzip or untar the distribution into a temporary directory
4. Copy the linking jar, in this case `ant-starteam.jar`, into the library directory you chose in step 1 above.
5. Copy the proprietary jar itself into the same directory.

Finally, if for some reason you are running on a system with both the JPackage and Apache versions of Ant available,

if you should want to run the Apache version (which will have to be specified with an absolute file name, not found on the path), you should use Ant's `--noconfig` command-line switch to avoid JPackage's classpath mechanism.

Advanced

There are lots of variants that can be used to run Ant. What you need is at least the following:

- The classpath for Ant must contain `ant.jar` and any jars/classes needed for your chosen JAXP-compliant XML parser.
- When you need JDK functionality (such as for the javac task or the rmic task), then `tools.jar` must be added. The scripts supplied with Ant, in the `bin` directory, will add the required JDK classes automatically, if the `JAVA_HOME` environment variable is set.
- When you are executing platform-specific applications, such as the exec task or the cvs task, the property `ant.home` must be set to the directory containing where you installed Ant. Again this is set by the Ant scripts to the value of the ANT_HOME environment variable.

The supplied ant shell scripts all support an `ANT_OPTS` environment variable which can be used to supply extra options to ant. Some of the scripts also read in an extra script stored in the users home directory, which can be used to set such options. Look at the source for your platform's invocation script for details.

Building Ant

To build Ant from source, you can either install the Ant source distribution or checkout the ant module from SVN.

Once you have installed the source, change into the installation directory.

Set the `JAVA_HOME` environment variable to the directory where the JDK is installed. See Installing Ant for examples on how to do this for your operating system.

Note: The bootstrap process of Ant requires a greedy compiler like Sun's javac or jikes. It does not work with gcj or kjc.

Make sure you have downloaded any auxiliary jars required to build tasks you are interested in. These should be added to the `lib/optional` directory of the source tree. See Library Dependencies for a list of JAR requirements for various features. Note that this will make the auxiliary JAR available for the building of Ant only. For running Ant you will still need to make the JARs available as described under Installing Ant.

As of version 1.7.0 Ant has a hard dependency on JUnit and you must install it manually into `lib/optional` (download it from JUnit.org) if you are using a source distribution of Ant.

Your are now ready to build Ant:

```
build -Ddist.dir=<directory_to_contain_Ant_distribution> dist   (Windows)

sh build.sh -Ddist.dir=<directory_to_contain_Ant_distribution> dist   (Unix)
```

This will create a binary distribution of Ant in the directory you specified.

The above action does the following:

- If necessary it will bootstrap the Ant code. Bootstrapping involves the manual compilation of enough Ant code to be able to run Ant. The bootstrapped Ant is used for the remainder of the build steps.

- Invokes the bootstrapped Ant with the parameters passed to the build script. In this case, these parameters define an Ant property value and specify the "dist" target in Ant's own `build.xml` file.
- Create the ant.jar and ant-launcher.jar JAR files
- Create optional JARs for which the build had the relevant libraries. If a particular library is missing from ANT_HOME/lib/optional, then the matching ant- JAR file will not be created. For example, ant-junit.jar is only built if there is a junit.jar in the optional directory.

On most occasions you will not need to explicitly bootstrap Ant since the build scripts do that for you. If however, the build file you are using makes use of features not yet compiled into the bootstrapped Ant, you will need to manually bootstrap. Run `bootstrap.bat` (Windows) or `bootstrap.sh` (UNIX) to build a new bootstrap version of Ant.

If you wish to install the build into the current `ANT_HOME` directory, you can use:

```
build install  (Windows)

sh build.sh install  (Unix)
```

You can avoid the lengthy Javadoc step, if desired, with:

```
build install-lite  (Windows)

sh build.sh install-lite  (Unix)
```

This will only install the `bin` and `lib` directories.

Both the `install` and `install-lite` targets will overwrite the current Ant version in `ANT_HOME`.

Ant's build script will try to set executable flags for its shell scripts on Unix-like systems. There are various reasons why the chmod-task might fail (like when you are running the build script as a different user than the one who installed Ant initially). In this case you can set the Ant property `chmod.fail` to false when starting the build like in

```
sh build.sh install -Dchmod.fail=false
```

and any error to change permission will not result in a build failure.

Library Dependencies

The following libraries are needed in Ant's classpath if you are using the indicated feature. Note that only one of the regexp libraries is needed for use with the mappers (and Java includes a regexp implementation which Ant will find automatically). You will also need to install the particular Ant optional jar containing the task definitions to make these tasks available. Please refer to the Installing Ant / Optional Tasks section above.

Jar Name	Needed For	Available At
jakarta-regexp-1.3.jar	regexp type with mappers (if you do not wish to use java.util.regex)	http://jakarta.apache.org/regexp/
jakarta-oro-2.0.8.jar	regexp type with mappers (if you do not wish to use java.util.regex) and the Perforce tasks To use the FTP task, you need jakarta-oro 2.0.8 or later, and commons-net	http://jakarta.apache.org/oro/

junit.jar	<junit> task. May be in classpath passed to task rather than Ant's classpath.	http://www.junit.org/
xalan.jar	junitreport task	http://xml.apache.org/xalan-j/
antlr.jar	antlr task	http://www.antlr.org/
bsf.jar	script task **Note**: Ant 1.6 and later require Apache BSF, not the IBM version. I.e. you need BSF 2.3.0-rc1 or later. **Note**: BSF 2.4.0 is needed to use a post 1.5R3 version of rhino's javascript. **Note**: BSF 2.4.0 uses jakarata-commons-logging so it needs the commons-logging.jar.	http://jakarta.apache.org/bsf/
Groovy jars	Groovy with script and scriptdef tasks You need to get the groovy jar and two asm jars from a groovy installation. The jars are groovy-[version].jar, asm-[vesion].jar and asm-util-[version].jar and antlr-[version].jar. As of groovy version 1.0-JSR-06, the jars are groovy-1.0-JSR-06.jar, antlr-2.7.5.jar, asm-2.2.jar and asm-util-2.2.jar. Alternatively one may use the embedded groovy jar file. This is located in the embedded directory of the groovy distribution. This bundles all the needed jar files into one jar file. It is called groovy-all-[version].jar.	http://groovy.codehaus.org/ The asm jars are also available from the creators of asm - http://asm.objectweb.org/
netrexx.jar	netrexx task, Rexx with the script task	http://www.ibm.com/software/awdtools/netrexx/download.html
js.jar	Javascript with script task If you use Apache BSF 2.3.0-rc1, you must use rhino 1.5R3 (later versions of BSF (e.g. version 2.4.0) work with 1.5R4 and higher).	http://www.mozilla.org/rhino/
jython.jar	Python with script task Warning : jython.jar also contains classes from jakarta-oro. Remove these classes if you are also using jakarta-oro.	http://jython.sourceforge.net/
jpython.jar	Python with script task **deprecated, jython is the prefered engine**	http://www.jpython.org/
jacl.jar and tcljava.jar	TCL with script task	http://www.scriptics.com/software/java/
BeanShell JAR(s)	BeanShell with script task. **Note**: Ant requires BeanShell version 1.3 or later	http://www.beanshell.org/
jruby.jar	Ruby with script task	http://jruby.sourceforge.net/

judo.jar	Judoscript with script task	http://www.judoscript.com/index.html
commons-logging.jar	CommonsLoggingListener	http://jakarta.apache.org/commons/logging/index.html
log4j.jar	Log4jListener	http://jakarta.apache.org/log4j/docs/index.html
commons-net.jar	ftp, rexec and telnet tasks jakarta-oro 2.0.8 or later is required together with commons-net 1.4.0. For all users, a minimum version of commons-net of 1.4.0 is recommended. Earlier versions did not support the full range of configuration options, and 1.4.0 is needed to compile Ant.	http://jakarta.apache.org/commons/net/index.html
bcel.jar	classfileset data type, JavaClassHelper used by the ClassConstants filter reader and optionally used by ejbjar for dependency determination	http://jakarta.apache.org/bcel/
mail.jar	Mail task with Mime encoding, and the MimeMail task	http://java.sun.com/products/javamail/
jsse.jar	Support for SMTP over TLS/SSL in the Mail task Already included Java 1.4+	http://java.sun.com/products/jsse/
activation.jar	Mail task with Mime encoding, and the MimeMail task	http://java.sun.com/products/javabeans/glasgow/jaf.html
jdepend.jar	jdepend task	http://www.clarkware.com/software/JDepend.html
resolver.jar **1.1beta or later**	xmlcatalog datatype *only if support for external catalog files is desired*	http://xml.apache.org/commons/.
jsch.jar **0.1.42 or later**	sshexec and scp tasks	http://www.jcraft.com/jsch/index.html
JAI - Java Advanced Imaging	image task	https://jai.dev.java.net/
Starteam SDK	Starteam version management tasks	http://www.borland.com/downloads/download_starteam.html

Troubleshooting

Diagnostics

Ant has a built in diagnostics feature. If you run `ant -diagnostics` ant will look at its internal state and print it out. This code will check and print the following things.

- Where Ant is running from. Sometimes you can be surprised.
- The version of ant.jar and of the ant-*.jar containing the optional tasks - and whether they match
- Which JAR files are in ANT_HOME/lib

- Which optional tasks are available. If a task is not listed as being available, either it is not present, or libraries that it depends on are absent.
- XML Parser information
- JVM system properties
- The status of the temp directory. If this is not writeable, or its clock is horribly wrong (possible if it is on a network drive), a lot of tasks will fail with obscure error messages.
- The current time zone as Java sees it. If this is not what it should be for your location, then dependency logic may get confused.

Running `ant -diagnostics` is a good way to check that ant is installed. It is also a first step towards self-diagnosis of any problem. Any configuration problem reported to the user mailing list will probably result ins someone asking you to run the command and show the results, so save time by using it yourself.

For under-IDE diagostics, use the <diagnostics> task to run the same tests as an ant task. This can be added to a diagnostics target in a build file to see what tasks are available under the IDE, what the XML parser and classpath is, etc.

user mailing list

If you cannot get Ant installed or working, the Ant user mailing list is the best place to start with any problem. Please do your homework first, make sure that it is not a CLASSPATH problem, and run a diagnostics check to see what Ant thinks of its own state. Why the user list, and not the developer list? Because there are more users than developers, so more people who can help you.

Please only file a bug report against Ant for a configuration/startup problem if there really is a fixable bug in Ant related to configuration, such as it not working on a particular platform, with a certain JVM version, etc, or if you are advised to do it by the user mailing list.

Platform Issues

Java versions

Java 1.5

You may need a bigger stack than default, especially if you are using the built in XSLT engine. We recommend you use Apache Xalan; indeed, some tasks (JUnit report in XML, for example) may not work against the shipping XSL engine.

Java 1.2

You may sometimes see messages like

```
A nonfatal internal JIT (3.10.107(x)) error 'chgTarg: Conditional' has
occurred in :
  'org/apache/tools/ant/Project.addReference
(Ljava/lang/String;Ljava/lang/Object;)V': Interpreting method.
  Please report this error in detail to
  http://java.sun.com/cgi-bin/bugreport.cgi
```

These are caused by subtle incompatibilities between the Java1.4+ compiled release; bugs in the Java runtime that Sun won't fix. Possible solutions:-

1. Upgrade your JVM.
2. Recompile Ant on a Java1.2 machine
3. Add `-Djava.compiler=NONE` to the value of your ANT_OPTS environment variable. This turns the JIT off.
4. Ignore the messages.

Unix and Linux

- You should use a GNU version of `tar` to untar the Apache Ant source tree, if you have downloaded this as a tar file. If you get wierd errors about missing files, this is the problem.
- Ant does not preserve file permissions when a file is copied, moved or archived, because Java does not let it read or write the permissions. Use `<chmod>` to set permissions, and when creating a tar archive, use the `mode` attribute of `<tarfileset>` to set the permissions in the tar file, or `<apply>` the real tar program.
- Ant is not symbolic link aware in moves, deletes and when recursing down a tree of directories to build up a list of files. Unexpected things can happen.
- Linux on IA-64: apparently you need a larger heap than the default one (64M) to compile big projects. If you get out of heap errors, either increase the heap or use a forking javac. Better yet, use jikes for extra compilation speed.

Microsoft Windows

Windows 9x (win95, win98, win98SE and winME) are not supported in Ant1.7,

The Ant team has retired support for these products because they are outdated and can expose customers to security risks. We recommend that customers who are still running Windows 98 or Windows Me upgrade to a newer, more

secure operating system, as soon as possible.

Customers who upgrade to Linux report improved security, richer functionality, and increased productivity.

Microsoft Windows 2K, XP and Server 2K03

Windows 9x (win95, win98, win98SE and winME) has a batch file system which does not work fully with long file names, so we recommend that ant and the JDK are installed into directories without spaces, and with 8.3 filenames. The Perl and Python launcher scripts do not suffer from this limitation.

All versions of windows are usually case insensitive, although mounted file systems (Unix drives, Clearcase views) can be case sensitive underneath, confusing patternsets.

Ant can often not delete a directory which is open in an Explorer window. There is nothing we can do about this short of spawning a program to kill the shell before deleting directories. Nor can files that are in use be overwritten.

Finally, if any Ant task fails with an IOError=2, it means that whatever native program Ant is trying to run, it is not on the path.

Microsoft Windows Vista

There are reports of problems with Windows Vista security bringing up dialog boxes asking if the user wants to run an untrusted executable during an ant run, such as when the <signjar> task runs the jarsigner.exe program. This is beyond Ant's control, and stems from the OS trying to provide some illusion of security by being reluctant to run unsigned native executables. The latest Java versions appear to resolve this problem by having signed binaries.

Cygwin

Cygwin is not an operating system; rather it is an application suite running under Windows and providing some UNIX like functionality. Sun has not created any specific Java Development Kit or Java Runtime Environment for cygwin. See this link : http://www.inonit.com/cygwin/faq/ . Only Windows path names are supported by JDK and JRE tools under Windows or cygwin. Relative path names such as "src/org/apache/tools" are supported, but Java tools do not understand /cygdrive/c to mean c:\.

The utility cygpath (used industrially in the ant script to support cygwin) can convert cygwin path names to Windows. You can use the <exec> task in ant to convert cygwin paths to Windows path, for instance like that :

```
<property name="some.cygwin.path" value="/cygdrive/h/somepath"/>
<exec executable="cygpath" outputproperty="windows.pathname">
   <arg value="--windows"/>
   <arg value="${some.cygwin.path}"/>
</exec>
<echo message="${windows.pathname}"/>
```

We get lots of support calls from Cygwin users. Either it is incredibly popular, or it is trouble. If you do use it, remember that Java is a Windows application, so Ant is running in a Windows process, not a Cygwin one. This will save us having to mark your bug reports as invalid.

Apple MacOS X

MacOS X is the first of the Apple platforms that Ant supports completely; it is treated like any other Unix.

Novell Netware

To give the same level of sophisticated control as Ant's startup scripts on other platforms, it was decided to make the main ant startup on NetWare be via a Perl Script, "runant.pl". This is found in the bin directory (for instance - bootstrap\bin or dist\bin).

One important item of note is that you need to set up the following to run ant:

- `CLASSPATH` - put ant.jar and any other needed jars on the system classpath.
- `ANT_OPTS` - On NetWare, `ANT_OPTS` needs to include a parameter of the form, "-envCWD=`ANT_HOME`", with `ANT_HOME` being the fully expanded location of Ant, **not** an environment variable. This is due to the fact that the NetWare System Console has no notion of a current working directory.

It is suggested that you create up an ant.ncf that sets up these parameters, and calls `perl ANT_HOME/dist/bin/runant.pl`

The following is an example of such an NCF file(assuming ant is installed in 'sys:/apache-ant/'):

```
envset CLASSPATH=SYS:/apache-ant/bootstrap/lib/ant.jar
envset CLASSPATH=$CLASSPATH;SYS:/apache-ant/lib/optional/junit.jar
envset CLASSPATH=$CLASSPATH;SYS:/apache-ant/bootstrap/lib/optional.jar

setenv ANT_OPTS=-envCWD=sys:/apache-ant
envset ANT_OPTS=-envCWD=sys:/apache-ant
setenv ANT_HOME=sys:/apache-ant/dist/lib
envset ANT_HOME=sys:/apache-ant/dist/lib

perl sys:/apache-ant/dist/bin/runant.pl
```

Ant works on JVM version 1.3 or higher. You may have some luck running it on JVM 1.2, but serious problems have been found running Ant on JVM 1.1.7B. These problems are caused by JVM bugs that will not be fixed.

JVM 1.3 is supported on Novell NetWare versions 5.1 and higher.

Other platforms

Support for other platforms is not guaranteed to be complete, as certain techniques to hide platform details from build files need to be written and tested on every particular platform. Contributions in this area are welcome.

Proxy Configuration

This page discussing proxy issues on command-line ant. Consult your IDE documentation for IDE-specific information upon proxy setup.

All tasks and threads running in Ant's JVM share the same HTTP/FTP/Socks proxy configuration.

When any task tries to retrieve content from an HTTP page, including the `<get>` task, any automated URL retrieval in an XML/XSL task, or any third-party task that uses the `java.net.URL` classes, the proxy settings may make the difference between success and failure.

Anyone authoring a build file behind a blocking firewall will immediately appreciate the problems and may want to write a build file to deal with the problem, but users of third party build build files may find that the build file itself does not work behind the firewall.

This is a long standing problem with Java and Ant. The only way to fix it is to explictly configure Ant with the proxy settings, either by passing down the proxy details as JVM properties, or to tell Ant on a Java1.5+ system to have the JVM work it out for itself.

Java1.5+ proxy support (new for Ant1.7)

When Ant starts up, if the `-autoproxy` command is supplied, Ant sets the `java.net.useSystemProxies` system property. This tells a Java1.5+ JVM to use the current set of property settings of the host environment. Other JVMs, such as the Kaffe and Apache Harmony runtimes, may also use this property in future. It is ignored on the Java1.4 and earlier runtimes.

This property maybe enough to give command-line Ant builds network access, although in practise the results are inconsistent.

It is has also been reported a breaking the IBM Java 5 JRE on AIX, and does not always work on Linux (presumably due to missing gconf settings) Other odd things can go wrong, like Oracle JDBC drivers or pure Java SVN clients.

To make the `-autoproxy` option the default, add it to the environment variable `ANT_ARGS`, which contains a list of arguments to pass to Ant on every command line run.

How Autoproxy works

We are grateful for some input from Sun as to how the proxy code works under Java 1.5 and up. The `java.net.useSystemProxies` is checked only once, at startup time, the other checks (registry, gconf, system properties) are done dynamically whenever needed (socket connection, URL connection etc..).

Windows

The JVM goes straight to the registry, bypassing WinInet, as it is not present/consistent on all supported Windows platforms (it is part of IE, really). Java 7 may use the Windows APIs on the platforms when it is present.

Linux

The JVM uses the gconf library to look at specific entries. The GConf-2 settings used are:

```
 - /system/http_proxy/use_http_proxy             boolean
 - /system/http_proxy/use_authentication         boolean
```

```
- /system/http_proxy/host                          string
- /system/http_proxy/authentication_user          string
- /system/http_proxy/authentication_password      string
- /system/http_proxy/port                          int
- /system/proxy/socks_host                         string
- /system/proxy/mode                               string
- /system/proxy/ftp_host                           string
- /system/proxy/secure_host                        string
- /system/proxy/socks_port                         int
- /system/proxy/ftp_port                           int
- /system/proxy/secure_port                        int
- /system/proxy/no_proxy_for                       list
- /system/proxy/gopher_host                        string
- /system/proxy/gopher_port                        int
```

If you are using KDE or another GUI than Gnome, you can still use the `gconf-editor` tool to add these entries.

Manual JVM options

Any JVM can have its proxy options explicitly configured by passing the appropriate `-D` system property options to the runtime. Ant can be configured through all its shell scripts via the `ANT_OPTS` environment variable, which is a list of options to supply to Ant's JVM:

For bash:

```
    export ANT_OPTS="-Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080"
```

For csh/tcsh:

```
    setenv ANT_OPTS "-Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080"
```

If you insert this line into the Ant shell script itself, it gets picked up by all continuous integration tools running on the system that call Ant via the command line.

For Windows, set the `ANT_OPTS` environment variable in the appropriate "My Computer" properties dialog box (winXP), "Computer" properties (Vista)

This mechanism works across Java versions, is cross-platform and reliable. Once set, all build files run via the command line will automatically have their proxy setup correctly, without needing any build file changes. It also apparently overrides Ant's automatic proxy settings options.

It is limited in the following ways:

1. Does not work under IDEs. These need their own proxy settings changed
2. Not dynamic enough to deal with laptop configuration changes.

SetProxy Task

The setproxy task can be used to explicitly set a proxy in a build file. This manipulates the many proxy configuration properties of a JVM, and controls the proxy settings for all network operations in the same JVM from that moment.

If you have a build file that is only to be used in-house, behind a firewall, on an older JVM, *and you cannot change Ant's JVM proxy settings*, then this is your best option. It is ugly and brittle, because the build file now contains system configuration information. It is also hard to get this right across the many possible proxy options of different users (none, HTTP, SOCKS).

Note that proxy configurations set with this task will probably override any set by other mechanisms. It can also be used with fancy tricks to only set a proxy if the proxy is considered reachable:

```
  <target name="probe-proxy" depends="init">
```

```
    <condition property="proxy.enabled">
      <and>
        <isset property="proxy.host"/>
        <isreachable host="${proxy.host}"/>
      </and>
    </condition>
  </target>

  <target name="proxy" depends="probe-proxy" if="proxy.enabled">
    <property name="proxy.port" value="80"/>
    <property name="proxy.user" value=""/>
    <property name="proxy.pass" value=""/>
    <setproxy proxyhost="${proxy.host}" proxyport="${proxy.port}"
      proxyuser="${proxy.user}" proxypassword="${proxy.pass}"/>
  </target>
```

Custom ProxySelector implementations

As Java lets developers write their own ProxySelector implementations, it is theoretically possible for someone to write their own proxy selector class that uses different policies to determine proxy settings. There is no explicit support for this in Ant, and it has not, to the team's knowledge, been attempted.

This could be the most flexible of solutions, as one could easily imagine an Ant-specific proxy selector that was driven off ant properties, rather than system properties. Developers could set proxy options in their custom build.properties files, and have this propagate.

One issue here is with concurrency: the default proxy selector is per-JVM, not per-thread, and so the proxy settings will apply to all sockets opened on all threads; we also have the problem of how to propagate options from one build to the JVM-wide selector.

Configuring the Proxy settings of Java programs under Ant

Any program that is executed with `<java>` without setting `fork="true"` will pick up the Ant's settings. If you need different values, set `fork="false"` and provide the values in `<sysproperty>` elements.

If you wish to have a forked process pick up the Ant's settings, use the `<syspropertyset>` element to propagate the normal proxy settings. The following propertyset is a datatype which can be referenced in a `<java>` task to pass down the current values.

```
<propertyset id="proxy.properties">
  <propertyref prefix="java.net.useSystemProxies"/>
  <propertyref prefix="http."/>
  <propertyref prefix="https."/>
  <propertyref prefix="ftp."/>
  <propertyref prefix="socksProxy"/>
</propertyset>
```

Summary and conclusions

There are four ways to set up proxies in Ant.

1. With Ant1.7 and Java 1.5+ using the `-autoproxy` parameter.
2. Via JVM system properties -set these in the ANT_ARGS environment variable.
3. Via the <setproxy> task.
4. Custom ProxySelector implementations

Proxy settings are automatically shared with Java programs started under Ant *that are not forked*; to pass proxy settings down to subsidiary programs, use a propertyset.

Over time, we expect the Java 5+ proxy features to stabilize, and for Java code to adapt to them. However, given the

fact that it currently does break some builds, it will be some time before Ant enables the automatic proxy feature by default. Until then, you have to enable the `-autoproxy` option or use one of the alternate mechanisms to configure the JVM.

Further reading

- Java Networking Properties. Notice how not all proxy settings are documented there.
- Proxies

Using Ant

Writing a Simple Buildfile

Ant's buildfiles are written in XML. Each buildfile contains one project and at least one (default) target. Targets contain task elements. Each task element of the buildfile can have an `id` attribute and can later be referred to by the value supplied to this. The value has to be unique. (For additional information, see the Tasks section below.)

Projects

A *project* has three attributes:

Attribute	Description	Required
name	the name of the project.	No
default	the default target to use when no target is supplied.	No; however, **since Ant 1.6.0**, every project includes an implicit target that contains any and all top-level tasks and/or types. This target will always be executed as part of the project's initialization, even when Ant is run with the -projecthelp option.
basedir	the base directory from which all path calculations are done. This attribute might be overridden by setting the "basedir" property beforehand. When this is done, it must be omitted in the project tag. If neither the attribute nor the property have been set, the parent directory of the buildfile will be used.	No

Optionally, a description for the project can be provided as a top-level `<description>` element (see the description type).

Each project defines one or more *targets*. A target is a set of *tasks* you want to be executed. When starting Ant, you can select which target(s) you want to have executed. When no target is given, the project's default is used.

Targets

A target can depend on other targets. You might have a target for compiling, for example, and a target for creating a distributable. You can only build a distributable when you have compiled first, so the distribute target *depends on* the compile target. Ant resolves these dependencies.

It should be noted, however, that Ant's `depends` attribute only specifies the *order* in which targets should be executed - it does not affect whether the target that specifies the dependency(s) gets executed if the dependent target(s) did not (need to) run.

More information can be found in the dedicated manual page.

Tasks

A task is a piece of code that can be executed.

A task can have multiple attributes (or arguments, if you prefer). The value of an attribute might contain references to a property. These references will be resolved before the task is executed.

Tasks have a common structure:

```
<name attribute1="value1" attribute2="value2" ... />
```

where *name* is the name of the task, *attributeN* is the attribute name, and *valueN* is the value for this attribute.

There is a set of built-in tasks, along with a number of optional tasks, but it is also very easy to write your own.

All tasks share a task name attribute. The value of this attribute will be used in the logging messages generated by Ant.

Tasks can be assigned an `id` attribute:

```
<taskname id="taskID" ... />
```

where *taskname* is the name of the task, and *taskID* is a unique identifier for this task. You can refer to the corresponding task object in scripts or other tasks via this name. For example, in scripts you could do:

```
<script ... >
  task1.setFoo("bar");
</script>
```

to set the `foo` attribute of this particular task instance. In another task (written in Java), you can access the instance via `project.getReference("task1")`.

Note[1]: If "task1" has not been run yet, then it has not been configured (ie., no attributes have been set), and if it is going to be configured later, anything you've done to the instance may be overwritten.

Note[2]: Future versions of Ant will most likely *not* be backward-compatible with this behaviour, since there will likely be no task instances at all, only proxies.

Properties

Properties are an important way to customize a build process or to just provide shortcuts for strings that are used repeatedly inside a build file.

In its most simple form properties are defined in the build file (for example by the property task) or might be set outside Ant. A property has a name and a value; the name is case-sensitive. Properties may be used in the value of task attributes or in the nested text of tasks that support them. This is done by placing the property name between "`${`" and "`}`" in the attribute value. For example, if there is a "builddir" property with the value "build", then this could be used in an attribute like this: `${builddir}/classes`. This is resolved at run-time as `build/classes`.

With Ant 1.8.0 property expansion has become much more powerful than simple key value pairs, more details can be found in the concepts section of this manual.

Example Buildfile

```
<project name="MyProject" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
```

```
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
        description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Notice that we are declaring properties outside any target. As of Ant 1.6 all tasks can be declared outside targets (earlier version only allowed `<property>`,`<typedef>` and `<taskdef>`). When you do this they are evaluated before any targets are executed. Some tasks will generate build failures if they are used outside of targets as they may cause infinite loops otherwise (`<antcall>` for example).

We have given some targets descriptions; this causes the `projecthelp` invocation option to list them as public targets with the descriptions; the other target is internal and not listed.

Finally, for this target to work the source in the `src` subdirectory should be stored in a directory tree which matches the package names. Check the `<javac>` task for details.

Token Filters

A project can have a set of tokens that might be automatically expanded if found when a file is copied, when the filtering-copy behavior is selected in the tasks that support this. These might be set in the buildfile by the filter task.

Since this can potentially be a very harmful behavior, the tokens in the files **must** be of the form `@`*token*`@`, where *token* is the token name that is set in the `<filter>` task. This token syntax matches the syntax of other build systems that perform such filtering and remains sufficiently orthogonal to most programming and scripting languages, as well as with documentation systems.

Note: If a token with the format `@`*token*`@` is found in a file, but no filter is associated with that token, no changes take place; therefore, no escaping method is available - but as long as you choose appropriate names for your tokens, this should not cause problems.

Warning: If you copy binary files with filtering turned on, you can corrupt the files. This feature should be used with text files *only*.

Path-like Structures

You can specify `PATH`- and `CLASSPATH`-type references using both "`:`" and "`;`" as separator characters. Ant will convert

the separator to the correct character of the current operating system.

Wherever path-like values need to be specified, a nested element can be used. This takes the general form of:

```
    <classpath>
      <pathelement path="${classpath}"/>
      <pathelement location="lib/helper.jar"/>
    </classpath>
```

The `location` attribute specifies a single file or directory relative to the project's base directory (or an absolute filename), while the `path` attribute accepts colon- or semicolon-separated lists of locations. The `path` attribute is intended to be used with predefined paths - in any other case, multiple elements with `location` attributes should be preferred.

As a shortcut, the `<classpath>` tag supports `path` and `location` attributes of its own, so:

```
    <classpath>
      <pathelement path="${classpath}"/>
    </classpath>
```

can be abbreviated to:

```
    <classpath path="${classpath}"/>
```

In addition, one or more Resource Collections can be specified as nested elements (these must consist of file-type resources only). Additionally, it should be noted that although resource collections are processed in the order encountered, certain resource collection types such as fileset, dirset and files are undefined in terms of order.

```
    <classpath>
      <pathelement path="${classpath}"/>
      <fileset dir="lib">
        <include name="**/*.jar"/>
      </fileset>
      <pathelement location="classes"/>
      <dirset dir="${build.dir}">
        <include name="apps/**/classes"/>
        <exclude name="apps/**/*Test*"/>
      </dirset>
      <filelist refid="third-party_jars"/>
    </classpath>
```

This builds a path that holds the value of `${classpath}`, followed by all jar files in the `lib` directory, the `classes` directory, all directories named `classes` under the `apps` subdirectory of `${build.dir}`, except those that have the text `Test` in their name, and the files specified in the referenced FileList.

If you want to use the same path-like structure for several tasks, you can define them with a `<path>` element at the same level as *target*s, and reference them via their *id* attribute--see References for an example.

By default a path like structure will re-evaluate all nested resource collections whenever it is used, which may lead to unnecessary re-scanning of the filesystem. Since Ant 1.8.0 path has an optional *cache* attribute, if it is set to true, the path instance will only scan its nested resource collections once and assume it doesn't change during the build anymore (the default for *cache* still is *false*). Even if you are using the path only in a single task it may improve overall performance to set *cache* to *true* if you are using complex nested constructs.

A path-like structure can include a reference to another path-like structure (a path being itself a resource collection) via nested `<path>` elements:

```
    <path id="base.path">
      <pathelement path="${classpath}"/>
      <fileset dir="lib">
        <include name="**/*.jar"/>
      </fileset>
      <pathelement location="classes"/>
    </path>
```

```
    <path id="tests.path" cache="true">
      <path refid="base.path"/>
      <pathelement location="testclasses"/>
    </path>
```

The shortcuts previously mentioned for `<classpath>` are also valid for `<path>`.For example:

```
    <path id="base.path">
      <pathelement path="${classpath}"/>
    </path>
```

can be written as:

```
    <path id="base.path" path="${classpath}"/>
```

Path Shortcut

In Ant 1.6 a shortcut for converting paths to OS specific strings in properties has been added. One can use the expression ${toString:*pathreference*} to convert a path element reference to a string that can be used for a path argument. For example:

```
  <path id="lib.path.ref">
    <fileset dir="lib" includes="*.jar"/>
  </path>
  <javac srcdir="src" destdir="classes">
    <compilerarg arg="-Xbootclasspath/p:${toString:lib.path.ref}"/>
  </javac>
```

Command-line Arguments

Several tasks take arguments that will be passed to another process on the command line. To make it easier to specify arguments that contain space characters, nested `arg` elements can be used.

Attribute	Description	Required
value	a single command-line argument; can contain space characters.	Exactly one of these.
file	The name of a file as a single command-line argument; will be replaced with the absolute filename of the file.	
path	A string that will be treated as a path-like string as a single command-line argument; you can use `;` or `:` as path separators and Ant will convert it to the platform's local conventions.	
pathref	Reference to a path defined elsewhere. Ant will convert it to the platform's local conventions.	
line	a space-delimited list of command-line arguments.	
prefix	A fixed string to be placed in front of the argument. In the case of a line broken into parts, it will be placed in front of every part. *Since Ant 1.8.*	No
suffix	A fixed string to be placed immediately after the argument. In the case of a line broken into parts, it will be placed after every part. *Since Ant 1.8.*	No

It is highly recommended to avoid the `line` version when possible. Ant will try to split the command line in a way similar to what a (Unix) shell would do, but may create something that is very different from what you expect under some circumstances.

Examples

```
  <arg value="-l -a"/>
```

is a single command-line argument containing a space character, *not* separate commands "-l" and "-a".

```
  <arg line="-l -a"/>
```

This is a command line with two separate arguments, "-l" and "-a".

```
  <arg path="/dir;/dir2:\dir3"/>
```

is a single command-line argument with the value `\dir;\dir2;\dir3` on DOS-based systems and `/dir:/dir2:/dir3` on Unix-like systems.

References

Any project element can be assigned an identifier using its `id` attribute. In most cases the element can subsequently be referenced by specifying the `refid` attribute on an element of the same type. This can be useful if you are going to replicate the same snippet of XML over and over again--using a `<classpath>` structure more than once, for example.

The following example:

```
<project ... >
  <target ... >
    <rmic ...>
      <classpath>
        <pathelement location="lib/"/>
        <pathelement path="${java.class.path}/"/>
        <pathelement path="${additional.path}"/>
      </classpath>
    </rmic>
  </target>

  <target ... >
    <javac ...>
      <classpath>
        <pathelement location="lib/"/>
        <pathelement path="${java.class.path}/"/>
        <pathelement path="${additional.path}"/>
      </classpath>
    </javac>
  </target>
</project>
```

could be rewritten as:

```
<project ... >
  <path id="project.class.path">
    <pathelement location="lib/"/>
    <pathelement path="${java.class.path}/"/>
    <pathelement path="${additional.path}"/>
  </path>

  <target ... >
    <rmic ...>
      <classpath refid="project.class.path"/>
    </rmic>
  </target>

  <target ... >
    <javac ...>
      <classpath refid="project.class.path"/>
    </javac>
  </target>
</project>
```

All tasks that use nested elements for PatternSets, FileSets, ZipFileSets or path-like structures accept references to these structures as shown in the examples. Using `refid` on a task will ordinarily have the same effect (referencing a task already declared), but the user should be aware that the interpretation of this attribute is dependent on the implementation of the element upon which it is specified. Some tasks (the property task is a handy example) deliberately assign a different meaning to `refid`.

Use of external tasks

Ant supports a plugin mechanism for using third party tasks. For using them you have to do two steps:

1. place their implementation somewhere where Ant can find them
2. declare them.

Don't add anything to the CLASSPATH environment variable - this is often the reason for very obscure errors. Use Ant's own mechanisms for adding libraries:

- via command line argument `-lib`
- adding to `${user.home}/.ant/lib`
- adding to `${ant.home}/lib`

For the declaration there are several ways:

- declare a single task per using instruction using `<taskdef name="taskname" classname="ImplementationClass"/>`
  `<taskdef name="for" classname="net.sf.antcontrib.logic.For" /> <for ... />`
- declare a bundle of tasks using a properties-file holding these taskname-ImplementationClass-pairs and `<taskdef>`
  `<taskdef resource="net/sf/antcontrib/antcontrib.properties" /> <for ... />`
- declare a bundle of tasks using a xml-file holding these taskname-ImplementationClass-pairs and `<taskdef>`
  `<taskdef resource="net/sf/antcontrib/antlib.xml" /> <for ... />`
- declare a bundle of tasks using a xml-file named antlib.xml, XML-namespace and `antlib:` protocoll handler
  `<project xmlns:ac="antlib:net.sf.antconrib"/> <ac:for ... />`

If you need a special function, you should

1. have a look at this manual, because Ant provides lot of tasks
2. have a look at the external task page in the manual (or better online)
3. have a look at the external task wiki page
4. ask on the Ant user list
5. implement (and share) your own

Targets

A target is a container of tasks that cooperate to reach a desired state during the build process.

Targets can depend on other targets and Ant ensures that these other targets have been executed before the current target. For example you might have a target for compiling and a target for creating a distributable. You can only build a distributable when you have compiled first, so the distribute target *depends on* the compile target.

Ant tries to execute the targets in the `depends` attribute in the order they appear (from left to right). Keep in mind that it is possible that a target can get executed earlier when an earlier target depends on it:

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Suppose we want to execute target D. From its `depends` attribute, you might think that first target C, then B and then A is executed. Wrong! C depends on B, and B depends on A, so first A is executed, then B, then C, and finally D.

```
Call-Graph:  A --> B --> C --> D
```

In a chain of dependencies stretching back from a given target such as D above, each target gets executed only once, even when more than one target depends on it. Thus, executing the D target will first result in C being called, which in turn will first call B, which in turn will first call A. After A, then B, then C have executed, execution returns to the dependency list of D, which will not call B and A, since they were already called in process of dependency resolution for C and B respectively as dependencies of D. Had no such dependencies been discovered in processing C and B, B and A would have been executed after C in processing D's dependency list.

A target also has the ability to perform its execution if (or unless) a property has been set. This allows, for example, better control on the building process depending on the state of the system (java version, OS, command-line property defines, etc.). To make a target *sense* this property, you should add the `if` (or `unless`) attribute with the name of the property that the target should react to. **Note:** In the most simple case Ant will only check whether the property has been set, the value doesn't matter, but using property expansions you can build more complex conditions. See the properties page for more details. For example:

```
<target name="build-module-A" if="module-A-present"/>
```

```
<target name="build-own-fake-module-A" unless="module-A-present"/>
```

In the first example, if the `module-A-present` property is set (to any value, e.g. *false*), the target will be run. In the second example, if the `module-A-present` property is set (again, to any value), the target will not be run.

Only one propertyname can be specified in the if/unless clause. If you want to check multiple conditions, you can use a dependend target for computing the result for the check:

```
<target name="myTarget" depends="myTarget.check" if="myTarget.run">
    <echo>Files foo.txt and bar.txt are present.</echo>
</target>

<target name="myTarget.check">
    <condition property="myTarget.run">
        <and>
            <available file="foo.txt"/>
            <available file="bar.txt"/>
        </and>
    </condition>
</target>
```

```
Call-Graph:  myTarget.check --> maybe(myTarget)
```

If no `if` and no `unless` attribute is present, the target will always be executed.

Important: the `if` and `unless` attributes only enable or disable the target to which they are attached. They do not control whether or not targets that a conditional target depends upon get executed. In fact, they do not even get evaluated until the target is about to be executed, and all its predecessors have already run.

The optional `description` attribute can be used to provide a one-line description of this target, which is printed by the `-projecthelp` command-line option. Targets without such a description are deemed internal and will not be listed, unless either the `-verbose` or `-debug` option is used.

It is a good practice to place your tstamp tasks in a so-called *initialization* target, on which all other targets depend. Make sure that target is always the first one in the depends list of the other targets. In this manual, most initialization targets have the name `"init"`.

```
<project>
    <target name="init">
        <tstamp/>
    </target>
    <target name="otherTarget" depends="init">
        ...
    </target>
</project>
```

Especially if you only have a few tasks you also could place these tasks directly under the project tag (since Ant 1.6.0):

```
<project>
    <tstamp/>
</project>
```

If the depends attribute and the if/unless attribute are set, the depends attribute is executed first.

A target has the following attributes:

Attribute	Description	Required
name	the name of the target.	Yes
depends	a comma-separated list of names of targets on which this target depends.	No
if	the name of the property that must be set in order for this target to execute, or something evaluating to true.	No
unless	the name of the property that must not be set in order for this target to execute, or something evaluating to false.	No
description	a short description of this target's function.	No
extensionOf	Adds the current target to the depends list of the named extension-point. *since Ant 1.8.0.*	No

A target name can be any alphanumeric string valid in the encoding of the XML file. The empty string "" is in this set, as is comma "," and space " ". Please avoid using these, as they will not be supported in future Ant versions because of all the confusion they cause on command line and IDE. IDE support of unusual target names, or any target name containing spaces, varies with the IDE.

Targets beginning with a hyphen such as `"-restart"` are valid, and can be used to name targets that should not be called directly from the command line.
For Ants main class every option starting with hyphen is an option for Ant itself and not a target. For that reason calling these target from command line is not possible. On the other hand IDEs usually don't use Ants main class as entry point and calling them from the IDE is usually possible.

Extension-Points

since Ant 1.8.0.

Extension-Points are similar to targets in that they have a name and a depends list and can be executed from the command line. Just like targets they represent a state during the build process.

Unlike targets they don't contain any tasks, their main purpose is to collect targets that contribute to the desired state in their depends list.

Targets can add themselves to an extension-points's depends list via their extensionOf attribute. The targets that add themselves will be added after the targets of the explicit depends-attribute of the extension-point, if multiple targets add themselves, their relative order is not defined.

The main purpose of an extension-point is to act as an extension point for build files designed to be imported. In the imported file an extension-point defines a state that must be reached and targets from other build files can join the depends list of said extension-point in order to contribute to that state.

For example your imported build file may need to compile code, it might look like:

```
<target name="create-directory-layout">
   ...
</target>
<extension-point name="ready-to-compile"
              depends="create-directory-layout"/>
<target name="compile" depends="ready-to-compile">
   ...
</target>

Call-Graph:  create-directory-layout --> 'empty slot' --> compile
```

And you need to generate some source before compilation, then in your main build file you may use something like

```
<target name="generate-sources"
        extensionOf="ready-to-compile">
   ...
</target>

Call-Graph:  create-directory-layout --> generate-sources  --> compile
```

This will ensure that the *generate-sources* target is executed before the *compile* target.

Don't rely on the order of the depends list, if *generate-sources* depends on *create-directory-layout* then it must explicitly depend on it via its own depends attribute.

Properties

Properties are key-value-pairs where Ant tries to expand `${key}` to `value` at runtime.

There are many tasks that can set properties, the most common one is the property task. In addition properties can be defined via command line arguments or similar mechanisms from outside of Ant.

Normally property values can not be changed, once a property is set, most tasks will not allow its value to be modified. In general properties are of global scope, i.e. once they have been defined they are available for any task or target invoked subsequently - it is not possible to set a property in a child build process created via the ant, antcall or subant tasks and make it available to the calling build process, though.

Starting with Ant 1.8.0 the local task can be used to create properties that are locally scoped to a target or a sequential element like the one of the macrodef task.

Built-in Properties

Ant provides access to all system properties as if they had been defined using a `<property>` task. For example, `${os.name}` expands to the name of the operating system.

For a list of system properties see the Javadoc of System.getProperties.

In addition, Ant has some built-in properties:

```
basedir              the absolute path of the project's basedir (as set
                     with the basedir attribute of <project>).
ant.file             the absolute path of the buildfile.
ant.version          the version of Ant
ant.project.name     the name of the project that is currently executing;
                     it is set in the name attribute of <project>.
ant.project.default-target
                     the name of the currently executing project's
                     default target;  it is set via the default
                     attribute of <project>.
ant.project.invoked-targets
                     a comma separated list of the targets that have
                     been specified on the command line (the IDE,
                     an <ant> task ...) when invoking the current
                     project.
ant.java.version     the JVM version Ant detected; currently it can hold
                     the values "1.2", "1.3",
                     "1.4",  "1.5" and "1.6".
ant.core.lib         the absolute path of the ant.jar file.
```

There is also another property, but this is set by the launcher script and therefore maybe not set inside IDEs:

```
ant.home             home directory of Ant
```

The following property is only set if Ant is started via the Launcher class (which means it may not be set inside IDEs either):

```
ant.library.dir      the directory that has been used to load Ant's
                     jars from.  In most cases this is ANT_HOME/lib.
```

PropertyHelpers

Ant's property handling is accomplished by an instance of `org.apache.tools.ant.PropertyHelper` associated with the current Project. You can learn more about this class by examining Ant's Java API. In Ant 1.8 the PropertyHelper class was much reworked and now itself employs a number of helper classes (actually instances of the `org.apache.tools.ant.PropertyHelper$Delegate` marker interface) to take care of discrete tasks such as property setting, retrieval, parsing, etc. This makes Ant's property handling highly extensible; also of interest is the new propertyhelper task used to manipulate the PropertyHelper and its delegates from the context of the Ant buildfile.

There are three sub-interfaces of `Delegate` that may be useful to implement.

- `org.apache.tools.ant.property.PropertyExpander` is responsible for finding the property name inside a string in the first place (the default extracts `foo` from `${foo}`).

  This is the interface you'd implement if you wanted to invent your own property syntax - or allow nested property expansions since the default implementation doesn't balance braces (see `NestedPropertyExpander` in the "props" Antlib for an example).

- `org.apache.tools.ant.PropertyHelper$PropertyEvaluator` is used to expand `${some-string}` into an `Object`.

  This is the interface you'd implement if you want to provide your own storage independent of Ant's project instance - the interface represents the reading end. An example for this would be `org.apache.tools.ant.property.LocalProperties` which implements storage for local properties.

  Another reason to implement this interface is if you wanted to provide your own "property protocol" like expanding `toString:foo` by looking up the project reference foo and invoking `toString()` on it (which is already implemented in Ant, see below).

- `org.apache.tools.ant.PropertyHelper$PropertySetter` is responsible for setting properties.

  This is the interface you'd implement if you want to provide your own storage independent of Ant's project instance - the interface represents the reading end. An example for this would be `org.apache.tools.ant.property.LocalProperties` which implements storage for local properties.

The default `PropertyExpander` looks similar to:

```
public class DefaultExpander implements PropertyExpander {
    public String parsePropertyName(String s, ParsePosition pos,
                                    ParseNextProperty notUsed) {
        int index = pos.getIndex();
        if (s.indexOf("${", index) == index) {
            int end = s.indexOf('}', index);
            if (end < 0) {
                throw new BuildException("Syntax error in property: " + s);
            }
            int start = index + 2;
            pos.setIndex(end + 1);
            return s.substring(start, end);
        }
        return null;
    }
}
```

The logic that replaces `${toString:some-id}` with the stringified representation of the object with id `some-id` inside the current build is contained in a PropertyEvaluator similar to the following code:

```
public class ToStringEvaluator implements PropertyHelper.PropertyEvaluator {
    private static final String prefix = "toString:";
    public Object evaluate(String property, PropertyHelper propertyHelper) {
        Object o = null;
        if (property.startsWith(prefix) && propertyHelper.getProject() != null) {
            o = propertyHelper.getProject().getReference(
                    property.substring(prefix.length()));
        }
```

```
        return o == null ? null : o.toString();
    }
}
```

Property Expansion

When Ant encounters a construct `${some-text}` the exact parsing semantics are subject to the configured property helper delegates.

$$ Expansion

In its default configuration Ant will expand the text `$$` to a single `$` and suppress the normal property expansion mechanism for the text immediately following it, i.e. `$${key}` expands to `${key}` and not `value` even though a property named `key` was defined and had the value `value`. This can be used to escape literal `$` characters and is useful in constructs that only look like property expansions or when you want to provide diagnostic output like in

```
  <echo>$${builddir}=${builddir}</echo>
```

which will echo this message:

```
  ${builddir}=build/classes
```

if the property `builddir` has the value `build/classes`.

In order to maintain backward compatibility with older Ant releases, a single '$' character encountered apart from a property-like construct (including a matched pair of french braces) will be interpreted literally; that is, as '$'. The "correct" way to specify this literal character, however, is by using the escaping mechanism unconditionally, so that "$$" is obtained by specifying "$$$$". Mixing the two approaches yields unpredictable results, as "$$$" results in "$$".

Nesting of Braces

In its default configuration Ant will not try to balance braces in property expansions, it will only consume the text up to the first closing brace when creating a property name. I.e. when expanding something like `${a${b}}` it will be translated into two parts:

1. the expansion of property `a${b` - likely nothing useful.
2. the literal text `}` resulting from the second closing brace

This means you can't use easily expand properties whose names are given by properties, but there are some workarounds for older versions of Ant. With Ant 1.8.0 and the the props Antlib you can configure Ant to use the `NestedPropertyExpander` defined there if you need such a feature.

Expanding a "Property Name"

In its most simple form `${key}` is supposed to look up a property named `key` and expand to the value of the property. Additional `PropertyEvaluators` may result in a different interpretation of `key`, though.

The props Antlib provides a few interesting evaluators but there are also a few built-in ones.

Getting the value of a Reference with ${toString:}

Any Ant type which has been declared with a reference can also its string value extracted by using the `${toString:}` operation, with the name of the reference listed after the `toString:` text. The `toString()` method of the Java class instance that is referenced is invoked -all built in types strive to produce useful and relevant output in such an instance.

For example, here is how to get a listing of the files in a fileset,

```
<fileset id="sourcefiles" dir="src" includes="**/*.java" />
<echo> sourcefiles = ${toString:sourcefiles} </echo>
```

There is no guarantee that external types provide meaningful information in such a situation

Getting the value of a Reference with ${ant.refid:}

Any Ant type which has been declared with a reference can also be used as a property by using the `${ant.refid:}` operation, with the name of the reference listed after the `ant.refid:` text. The difference between this operation and `${toString:}` is that `${ant.refid:}` will expand to the referenced object itself. In most circumstances the toString method will be invoked anyway, for example if the `${ant.refid:}` is surrounded by other text.

This syntax is most useful when using a task with attribute setters that accept objects other than String. For example if the setter accepts a Resource object as in

```
public void setAttr(Resource r) { ... }
```

then the syntax can be used to pass in resource subclasses previously defined as references like

```
  <url url="http://ant.apache.org/" id="anturl"/>
  <my:task attr="${ant.refid:anturl}"/>
```

If/Unless Attributes

The `<target>` element and various tasks (such as `<fail>`) and task elements (such as `<test>` in `<junit>`) support `if` and `unless` attributes which can be used to control whether the item is run or otherwise takes effect.

In Ant 1.7.1 and earlier, these attributes could only be property names. The item was enabled if a property with that name was defined - even to be the empty string or `false` - and disabled if the property was not defined. For example, the following works but there is no way to override the file existence check negatively (only positively):

```
<target name="-check-use-file">
    <available property="file.exists" file="some-file"/>
</target>
<target name="use-file" depends="-check-use-file" if="file.exists">
    <!-- do something requiring that file... -->
</target>
<target name="lots-of-stuff" depends="use-file,other-unconditional-stuff"/>
```

As of Ant 1.8.0, you may instead use property expansion; a value of `true` (or `on` or `yes`) will enable the item, while `false` (or `off` or `no`) will disable it. Other values are still assumed to be property names and so the item is enabled only if the named property is defined.

Compared to the older style, this gives you additional flexibility, because you can override the condition from the command line or parent scripts:

```
<target name="-check-use-file" unless="file.exists">
    <available property="file.exists" file="some-file"/>
</target>
<target name="use-file" depends="-check-use-file" if="${file.exists}">
    <!-- do something requiring that file... -->
</target>
<target name="lots-of-stuff" depends="use-file,other-unconditional-stuff"/>
```

Now `ant -Dfile.exists=false lots-of-stuff` will run `other-unconditional-stuff` but not `use-file`, as you might expect, and you can disable the condition from another script too:

```
<antcall target="lots-of-stuff">
    <param name="file.exists" value="false"/>
</antcall>
```

Similarly, an `unless` attribute disables the item if it is either the name of property which is defined, or if it evaluates to a `true`-like value. For example, the following allows you to define `skip.printing.message=true` in `my-prefs.properties` with the results you might expect:

```
<property file="my-prefs.properties"/>
<target name="print-message" unless="${skip.printing.message}">
    <echo>hello!</echo>
</target>
```

Running Ant

Command Line

If you've installed Ant as described in the Installing Ant section, running Ant from the command-line is simple: just type `ant`.

When no arguments are specified, Ant looks for a `build.xml` file in the current directory and, if found, uses that file as the build file and runs the target specified in the `default` attribute of the `<project>` tag. To make Ant use a build file other than `build.xml`, use the command-line option `-buildfile` *`file`*, where *file* is the name of the build file you want to use.

If you use the `-find` [*`file`*] option, Ant will search for a build file first in the current directory, then in the parent directory, and so on, until either a build file is found or the root of the filesystem has been reached. By default, it will look for a build file called `build.xml`. To have it search for a build file other than `build.xml`, specify a file argument. **Note:** If you include any other flags or arguments on the command line after the `-find` flag, you must include the file argument for the `-find` flag, even if the name of the build file you want to find is `build.xml`.

You can also set properties on the command line. This can be done with the `-D`*`property`*`=`*`value`* option, where *property* is the name of the property, and *value* is the value for that property. If you specify a property that is also set in the build file (see the property task), the value specified on the command line will override the value specified in the build file. Defining properties on the command line can also be used to pass in the value of environment variables; just pass `-DMYVAR=%MYVAR%` (Windows) or `-DMYVAR=$MYVAR` (Unix) to Ant. You can then access these variables inside your build file as `${MYVAR}`. You can also access environment variables using the property task's `environment` attribute.

Options that affect the amount of logging output by Ant are: `-quiet`, which instructs Ant to print less information to the console; `-verbose`, which causes Ant to print additional information to the console; and `-debug`, which causes Ant to print considerably more additional information.

It is also possible to specify one or more targets that should be executed. When omitted, the target that is specified in the `default` attribute of the `project` tag is used.

The `-projecthelp` option prints out a list of the build file's targets. Targets that include a `description` attribute are listed as "Main targets", those without a `description` are listed as "Other targets", then the "Default" target is listed ("Other targets" are only displayed if there are no main targets, or if Ant is invoked in -verbose or -debug mode).

Command-line Options Summary

```
ant [options] [target [target2 [target3] ...]]
Options:
  -help, -h              print this message
  -projecthelp, -p       print project help information
  -version               print the version information and exit
  -diagnostics           print information that might be helpful to
                         diagnose or report problems.
  -quiet, -q             be extra quiet
  -verbose, -v           be extra verbose
  -debug, -d             print debugging information
  -emacs, -e             produce logging information without adornments
  -lib <path>            specifies a path to search for jars and classes
  -logfile <file>        use given file for log
    -l     <file>                ''
  -logger <classname>    the class which is to perform logging
  -listener <classname>  add an instance of class as a project listener
  -noinput               do not allow interactive input
  -buildfile <file>      use given buildfile
```

```
    -file    <file>              ''
    -f       <file>              ''
  -D<property>=<value>   use value for given property
  -keep-going, -k        execute all targets that do not depend
                         on failed target(s)
  -propertyfile <name>   load all properties from file with -D
                         properties taking precedence
  -inputhandler <class>  the class which will handle input requests
  -find <file>           (s)earch for buildfile towards the root of
    -s  <file>           the filesystem and use it
  -nice  number          A niceness value for the main thread:
                         1 (lowest) to 10 (highest); 5 is the default
  -nouserlib             Run ant without using the jar files from ${user.home}/.ant/lib
  -noclasspath           Run ant without using CLASSPATH
  -autoproxy             Java 1.5+ : use the OS proxies
  -main <class>          override Ant's normal entry point
```

For more information about `-logger` and `-listener` see Loggers & Listeners.

For more information about `-inputhandler` see InputHandler.

Easiest way of changing the exit-behaviour is subclassing the original main class:

```
public class CustomExitCode extends org.apache.tools.ant.Main {
    protected void exit(int exitCode) {
        // implement your own behaviour, e.g. NOT exiting the JVM
    }
}
```

and starting Ant with access (`-lib path-to-class`) to this class.

Library Directories

Prior to Ant 1.6, all jars in the ANT_HOME/lib would be added to the CLASSPATH used to run Ant. This was done in the scripts that started Ant. From Ant 1.6, two directories are scanned by default and more can be added as required. The default directories scanned are ANT_HOME/lib and a user specific directory, ${user.home}/.ant/lib. This arrangement allows the Ant installation to be shared by many users while still allowing each user to deploy additional jars. Such additional jars could be support jars for Ant's optional tasks or jars containing third-party tasks to be used in the build. It also allows the main Ant installation to be locked down which will please system adminstrators.

Additional directories to be searched may be added by using the -lib option. The -lib option specifies a search path. Any jars or classes in the directories of the path will be added to Ant's classloader. The order in which jars are added to the classpath is as follows:

- -lib jars in the order specified by the -lib elements on the command line
- jars from ${user.home}/.ant/lib (unless -nouserlib is set)
- jars from ANT_HOME/lib

Note that the CLASSPATH environment variable is passed to Ant using a -lib option. Ant itself is started with a very minimalistic classpath. Ant should work perfectly well with an empty CLASSPATH environment variable, something the the -noclasspath option actually enforces. We get many more support calls related to classpath problems (especially quoting problems) than we like.

The location of ${user.home}/.ant/lib is somewhat dependent on the JVM. On Unix systems ${user.home} maps to the user's home directory whilst on recent versions of Windows it will be somewhere such as C:\Documents and Settings\username\.ant\lib. You should consult your JVM documentation for more details.

Examples

```
ant
```

runs Ant using the `build.xml` file in the current directory, on the default target.

```
ant -buildfile test.xml
```

runs Ant using the `test.xml` file in the current directory, on the default target.

```
ant -buildfile test.xml dist
```

runs Ant using the `test.xml` file in the current directory, on the target called `dist`.

```
ant -buildfile test.xml -Dbuild=build/classes dist
```

runs Ant using the `test.xml` file in the current directory, on the target called `dist`, setting the `build` property to the value `build/classes`.

```
ant -lib /home/ant/extras
```

runs Ant picking up additional task and support jars from the /home/ant/extras location

```
ant -lib one.jar;another.jar
```

```
ant -lib one.jar -lib another.jar
```

adds two jars to Ants classpath.

Files

The Ant wrapper script for Unix will source (read and evaluate) the file `~/.antrc` before it does anything. On Windows, the Ant wrapper batch-file invokes `%HOME%\antrc_pre.bat` at the start and `%HOME%\antrc_post.bat` at the end. You can use these files, for example, to set/unset environment variables that should only be visible during the execution of Ant. See the next section for examples.

Environment Variables

The wrapper scripts use the following environment variables (if set):

- `JAVACMD` - full path of the Java executable. Use this to invoke a different JVM than `JAVA_HOME/bin/java(.exe)`.
- `ANT_OPTS` - command-line arguments that should be passed to the JVM. For example, you can define system properties or set the maximum Java heap size here.
- `ANT_ARGS` - Ant command-line arguments. For example, set `ANT_ARGS` to point to a different logger, include a listener, and to include the `-find` flag.
  Note: If you include `-find` in `ANT_ARGS`, you should include the name of the build file to find, even if the file is called `build.xml`.

Java System Properties

Some of Ant's core classes can be configured via system properties.

Here is the result of a search through the codebase. Because system properties are available via Project instance, I searched for them with a

```
    grep -r -n "getPropert" * > ..\grep.txt
```

command. After that I filtered out the often-used but not-so-important values (most of them read-only values): *path.separator, ant.home, basedir, user.dir, os.name, line.separator, java.home, java.version, java.version, user.home,*

java.class.path
And I filtered out the *getPropertyHelper* access.

property name	valid values /default value	description
`ant.build.javac.source`	Source-level version number	Default *source* value for <javac>/<javadoc>
`ant.build.javac.target`	Class-compatibility version number	Default *target* value for <javac>
`ant.executor.class`	classname; default is org. apache. tools. ant. helper. DefaultExecutor	**Since Ant 1.6.3** Ant will delegate Target invocation to the org.apache.tools.ant.Executor implementation specified here.
`ant.file`	read only: full filename of the build file	This is set to the name of the build file. In <import>-ed files, this is set to the containing build file.
`ant.file.*`	read only: full filename of the build file of Ant projects	This is set to the name of a file by project; this lets you determine the location of <import>-ed files,
`ant.input.properties`	filename (required)	Name of the file holding the values for the PropertyFileInputHandler.
`ant.logger.defaults`	filename (optional, default '/org/ apache/ tools/ ant/ listener/ defaults.properties')	Name of the file holding the color mappings for the AnsiColorLogger.
`ant.netrexxc.*`	several formats	Use specified values as defaults for netrexxc.
`ant.PropertyHelper`	ant-reference-name (optional)	Specify the PropertyHelper to use. The object must be of the type org.apache.tools.ant.PropertyHelper. If not defined an object of org.apache.tools.ant.PropertyHelper will be used as PropertyHelper.
`ant.regexp.regexpimpl`	classname	classname for a RegExp implementation; if not set Ant uses JDK 1.4's implementation; RegExp-Mapper "Choice of regular expression implementation"
`ant.reuse.loader`	boolean	allow to reuse classloaders used in org.apache.tools.ant.util.ClasspathUtil
`ant.XmlLogger.stylesheet.uri`	filename (default 'log.xsl')	Name for the stylesheet to include in the logfile by XmlLogger.
`build.compiler`	name	Specify the default compiler to use. see javac, EJB Tasks (compiler attribute), javah
`build.compiler.emacs`	boolean (default false)	Enable emacs-compatible error messages. see javac "Jikes Notes"
`build.compiler.fulldepend`	boolean (default false)	Enable full dependency checking see javac "Jikes Notes"
`build.compiler.jvc.extensions`	boolean (default true)	enable Microsoft extensions of their java compiler see javac "Jvc Notes"

build.compiler.pedantic	boolean (default false)	Enable pedantic warnings. see javac "Jikes Notes"
build.compiler.warnings	Deprecated flag	see javac "Jikes Notes"
build.rmic	name	control the rmic compiler
build.sysclasspath	see its dedicated page, no default value	see its dedicated page
file.encoding	name of a supported character set (e.g. UTF-8, ISO-8859-1, US-ASCII)	use as default character set of email messages; use as default for source-, dest- and bundleencoding in translate see JavaDoc of java.nio.charset.Charset for more information about character sets (not used in Ant, but has nice docs).
jikes.class.path	path	The specified path is added to the classpath if jikes is used as compiler.
MailLogger.properties.file, MailLogger.*	filename (optional, defaults derived from Project instance)	Name of the file holding properties for sending emails by the MailLogger. Override properties set inside the buildfile or via command line.
org.apache.tools.ant.ProjectHelper	classname (optional, default 'org.apache.tools.ant.ProjectHelper')	specifies the classname to use as ProjectHelper. The class must extend org.apache.tools.ant.ProjectHelper.
p4.port, p4.client, p4.user	several formats	Specify defaults for port-, client- and user-setting of the perforce tasks.
websphere.home	path	Points to home directory of websphere. see EJB Tasks
XmlLogger.file	filename (default 'log.xml')	Name for the logfile for MailLogger.
ant.project-helper-repo.debug	boolean (default 'false')	Set it to true to enable debuging with Ant's ProjectHelper internal repository.

If new properties get added (it happens), expect them to appear under the "ant." and "org.apache.tools.ant" prefixes, unless the developers have a very good reason to use another prefix. Accordingly, please avoid using properties that begin with these prefixes. This protects you from future Ant releases breaking your build file.

return code

the ant start up scripts (in their Windows and Unix version) return the return code of the java program. So a successful build returns 0, failed builds return other values.

Cygwin Users

The Unix launch script that come with Ant works correctly with Cygwin. You should not have any problems launching Ant from the Cygwin shell. It is important to note, however, that once Ant is running it is part of the JDK which operates as a native Windows application. The JDK is not a Cygwin executable, and it therefore has no knowledge of Cygwin paths, etc. In particular when using the <exec> task, executable names such as "/bin/sh" will not work, even though these work from the Cygwin shell from which Ant was launched. You can use an executable name such as "sh"

and rely on that command being available in the Windows path.

OS/2 Users

The OS/2 launch script was developed to perform complex tasks. It has two parts: `ant.cmd` which calls Ant and `antenv.cmd` which sets the environment for Ant. Most often you will just call `ant.cmd` using the same command line options as described above. The behaviour can be modified by a number of ways explained below.

Script `ant.cmd` first verifies whether the Ant environment is set correctly. The requirements are:

1. Environment variable `JAVA_HOME` is set.
2. Environment variable `ANT_HOME` is set.
3. Environment variable `CLASSPATH` is set and contains at least one element from `JAVA_HOME` and at least one element from `ANT_HOME`.

If any of these conditions is violated, script `antenv.cmd` is called. This script first invokes configuration scripts if there exist: the system-wide configuration `antconf.cmd` from the `%ETC%` directory and then the user configuration `antrc.cmd` from the `%HOME%` directory. At this moment both `JAVA_HOME` and `ANT_HOME` must be defined because `antenv.cmd` now adds `classes.zip` or `tools.jar` (depending on version of JVM) and everything from `%ANT_HOME%\lib` except `ant-*.jar` to `CLASSPATH`. Finally `ant.cmd` calls per-directory configuration `antrc.cmd`. All settings made by `ant.cmd` are local and are undone when the script ends. The settings made by `antenv.cmd` are persistent during the lifetime of the shell (of course unless called automatically from `ant.cmd`). It is thus possible to call `antenv.cmd` manually and modify some settings before calling `ant.cmd`.

Scripts `envset.cmd` and `runrc.cmd` perform auxiliary tasks. All scripts have some documentation inside.

Running Ant as a background process on Unix(-like) systems

If you start Ant as a background process (like in `ant &`) and the build process creates another process, Ant will immediately try to read from standard input, which in turn will most likely suspend the process. In order to avoid this, you must redirect Ant's standard input or explicitly provide input to each spawned process via the input related attributes of the corresponding tasks.

Tasks that create such new processes include `<exec>`, `<apply>` or `<java>` when the `fork` attribute is `true`.

Running Ant via Java

If you have installed Ant in the do-it-yourself way, Ant can be started from one of two entry points:

```
java -Dant.home=c:\ant org.apache.tools.ant.Main [options] [target]

java -Dant.home=c:\ant org.apache.tools.ant.launch.Launcher [options] [target]
```

The first method runs Ant's traditional entry point. The second method uses the Ant Launcher introduced in Ant 1.6. The former method does not support the -lib option and all required classes are loaded from the CLASSPATH. You must ensure that all required jars are available. At a minimum the CLASSPATH should include:

- `ant.jar` and `ant-launcher.jar`
- jars/classes for your XML parser
- the JDK's required jar/zip files

The latter method supports the -lib, -nouserlib, -noclasspath options and will load jars from the specified ANT_HOME. You should start the latter with the most minimal classpath possible, generally just the ant-launcher.jar.

Ant can be started in Ant via the `<java>` command. Here is an example:

```
<java
        classname="org.apache.tools.ant.launch.Launcher"
        fork="true"
        failonerror="true"
        dir="${sub.builddir}"
        timeout="4000000"
        taskname="startAnt"
>
    <classpath>
        <pathelement location="${ant.home}/lib/ant-launcher.jar"/>
    </classpath>
    <arg value="-buildfile"/>
    <arg file="${sub.buildfile}"/>
    <arg value="-Dthis=this"/>
    <arg value="-Dthat=that"/>
    <arg value="-Dbasedir=${sub.builddir}"/>
    <arg value="-Dthe.other=the.other"/>
    <arg value="${sub.target}"/>
</java>
```

Overview of Ant Tasks

Given the large number of tasks available with Ant, it may be difficult to get an overall view of what each task can do. The following tables provide a short description of each task and a link to the complete documentation.

Archive Tasks

[Back to top]

Task Name	Description
BUnzip2	Expands a file packed using GZip or BZip2.
BZip2	Packs a file using the GZip or BZip2 algorithm. This task does not do any dependency checking; the output file is always generated
Cab	Creates Microsoft CAB archive files. It is invoked similar to the Jar or Zip tasks. This task will work on Windows using the external *cabarc* tool (provided by Microsoft), which must be located in your executable path.
Ear	An extension of the Jar task with special treatment for files that should end up in an Enterprise Application archive.
GUnzip	Expands a GZip file.
GZip	GZips a set of files.
Jar	Jars a set of files.
Jlink	*Deprecated.* Use the `zipfileset` and `zipgroupfileset` attributes of the Jar or Zip tasks instead.
Manifest	Creates a manifest file.
Rpm	Invokes the *rpm* executable to build a Linux installation file. This task currently only works on Linux or other Unix platforms with RPM support.
SignJar	Signs a jar or zip file with the *javasign* command-line tool.
Tar	Creates a tar archive.

Unjar	Unzips a jarfile.
Untar	Untars a tarfile.
Unwar	Unzips a warfile.
Unzip	Unzips a zipfile.
War	An extension of the Jar task with special treatment for files that should end up in the `WEB-INF/lib`, `WEB-INF/classes`, or `WEB-INF` directories of the Web Application Archive.
Zip	Creates a zipfile.

Audit/Coverage Tasks

[Back to top]

Task Name	Description
JDepend	Invokes the JDepend parser. This parser "traverses a set of Java source-file directories and generates design-quality metrics for each Java package".

Compile Tasks

[Back to top]

Task Name	Description
Depend	Determines which classfiles are out-of-date with respect to their source, removing the classfiles of any other classes that depend on the out-of-date classes, forcing the re-compile of the removed classfiles. Typically used in conjunction with the Javac task.
Javac	Compiles the specified source file(s) within the running (Ant) VM, or in another VM if the `fork` attribute is specified.
Apt	Runs the annotation processor tool (apt), and then optionally compiles the original code, and any generated source code.
JspC	Runs the JSP compiler. It can be used to precompile JSP pages for fast initial invocation of JSP pages, deployment on a server without the full JDK installed, or simply to syntax-check the pages without deploying them. The Javac task can be used to compile the generated Java source. (For Weblogic JSP compiles, see the Wljspc task.)
NetRexxC	Compiles a NetRexx source tree within the running (Ant) VM.
Rmic	Runs the *rmic* compiler on the specified file(s).
Wljspc	Compiles JSP pages using Weblogic's JSP compiler, *weblogic.jspc*. (For non-Weblogic JSP compiles, see the JspC task.

Deployment Tasks

[Back to top]

Task Name	Description
ServerDeploy	Task to run a "hot" deployment tool for vendor-specific J2EE server.

Documentation Tasks

[Back to top]

Task Name	Description
Javadoc/*Javadoc2*	Generates code documentation using the *javadoc* tool. The Javadoc2 task is deprecated; use the Javadoc task instead.

EJB Tasks

[Back to top]

Task Name	Description
EJB Tasks	(See the documentation describing the EJB tasks.)

Execution Tasks

[Back to top]

Task Name	Description
Ant	Runs Ant on a supplied buildfile, optionally passing properties (with possibly new values). This task can be used to build sub-projects.
AntCall	Runs another target within the same buildfile, optionally passing properties (with possibly new values).
Apply/*ExecOn*	Executes a system command. When the `os` attribute is specified, the command is only executed when Ant is run on one of the specified operating systems.
Dependset	This task compares a set of source files with a set of target files. If any of the source files is newer than any of the target files, all the target files are removed.
Exec	Executes a system command. When the `os` attribute is specified, the command is only executed when Ant is run on one of the specified operating systems.
Java	Executes a Java class within the running (Ant) VM, or in another VM if the `fork` attribute is specified.
Parallel	A container task that can contain other Ant tasks. Each nested task specified within the `<parallel>` tag will be executed in its own thread.
Sequential	A container task that can contain other Ant tasks. The nested tasks are simply executed in sequence. Its primary use is to support the sequential execution of a subset of tasks within the `<parallel>` tag.
Sleep	A task for suspending execution for a specified period of time. Useful when a build or deployment process requires an interval between tasks.
Subant	Calls a given target for all defined sub-builds. This is an extension of ant for bulk project execution.
Waitfor	Blocks execution until a set of specified conditions become true. This task is intended to be used with the Parallel task to synchronize a set of processes.

File Tasks

[Back to top]

Task Name	Description
Attrib	Changes the permissions and/or attributes of a file or all files inside the specified directories. Currently, it has effect only under Windows.
Checksum	Generates a checksum for a file or set of files. This task can also be used to perform checksum verifications.
Chgrp	Changes the group ownership of a file or all files inside the specified directories. Currently, it has effect only under Unix.
Chmod	Changes the permissions of a file or all files inside the specified directories. Currently, it has effect only under Unix. The permissions are also UNIX style, like the arguments for the *chmod* command.
Chown	Changes the owner of a file or all files inside the specified directories. Currently, it has effect only under Unix.
Concat	Concatenates multiple files into a single one or to Ant's logging system.

Copy	Copies a file or Fileset to a new file or directory.
Copydir	*Deprecated.* Use the Copy task instead.
Copyfile	*Deprecated.* Use the Copy task instead.
Delete	Deletes either a single file, all files and sub-directories in a specified directory, or a set of files specified by one or more FileSets.
Deltree	*Deprecated.* Use the Delete task instead.
Filter	Sets a token filter for this project, or reads multiple token filters from a specified file and sets these as filters. Token filters are used by all tasks that perform file-copying operations.
FixCRLF	Modifies a file to add or remove tabs, carriage returns, linefeeds, and EOF characters.
Get	Gets a file from a URL.
Mkdir	Creates a directory. Non-existent parent directories are created, when necessary.
Move	Moves a file to a new file or directory, or a set(s) of file(s) to a new directory.
Patch	Applies a "diff" file to originals.
Rename	*Deprecated.* Use the Move task instead.
RenameExtensions	*Deprecated*. Use the Move task with a glob mapper instead.
Replace	Replace is a directory-based task for replacing the occurrence of a given string with another string in selected file.
ReplaceRegExp	Directory-based task for replacing the occurrence of a given regular expression with a substitution pattern in a file or set of files.
Sync	Synchronize two directory trees.
Tempfile	Generates a name for a new temporary file and sets the specified property to that name.
Touch	Changes the modification time of a file and possibly creates it at the same time.

Java2 Extensions Tasks

[Back to top]

Task Name	Description
Jarlib-available	Check whether an extension is present in a FileSet or an ExtensionSet. If the extension is present, the specified property is set.
Jarlib-display	Display the "Optional Package" and "Package Specification" information contained within the specified jars.
Jarlib-manifest	Task to generate a manifest that declares all the dependencies in manifest. The dependencies are determined by looking in the specified path and searching for Extension/"Optional Package" specifications in the manifests of the jars.
Jarlib-resolve	Try to locate a jar to satisfy an extension, and place the location of the jar into the specified property.

Logging Tasks

[Back to top]

Task Name	Description
Record	Runs a listener that records the logging output of the build-process events to a file. Several recorders can exist at the same time. Each recorder is associated with a file.

Mail Tasks

[Back to top]

Task Name	Description
Mail	A task to send SMTP email.
MimeMail	*Deprecated*. Use the Mail task instead.

Miscellaneous Tasks

[Back to top]

Task Name	Description
Defaultexcludes	Modify the list of default exclude patterns from within your build file.
Echo	Echoes text to `System.out` or to a file.
Fail	Exits the current build by throwing a BuildException, optionally printing additional information.
GenKey	Generates a key in keystore.
HostInfo	Sets properties related to the provided host, or to the host the process is run on.
Input	Allows user interaction during the build process by displaying a message and reading a line of input from the console.
Script	Executes a script in a Apache BSF-supported language.
Sound	Plays a sound file at the end of the build, according to whether the build failed or succeeded.
Splash	Displays a splash screen.
Sql	Executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file using the `src` attribute, or from between the enclosing SQL tags.
Taskdef	Adds a task definition to the current project, such that this new task can be used in the current project.
TStamp	Sets the `DSTAMP`, `TSTAMP`, and `TODAY` properties in the current project, based on the current date and time.
Typedef	Adds a data-type definition to the current project, such that this new type can be used in the current project.
XmlValidate	Checks that XML files are valid (or only well-formed). This task uses the XML parser that is currently used by Ant by default, but any SAX1/2 parser can be specified, if needed.

Pre-process Tasks

[Back to top]

Task Name	Description
ANTLR	Invokes the ANTLR Translator generator on a grammar file.
AntStructure	Generates a DTD for Ant buildfiles that contains information about all tasks currently known to Ant.
Import	Import another build file and potentially override targets in it with targets of your own.
Include	Include another build file.
JavaCC	Invokes the JavaCC compiler-compiler on a grammar file.
Javah	Generates JNI headers from a Java class.
JJDoc	Invokes the JJDoc documentation generator for the JavaCC compiler-compiler. JJDoc takes a JavaCC parser specification and produces documentation for the BNF grammar. It can operate in three modes,

	determined by command line options. This task only invokes JJDoc if the grammar file is newer than the generated BNF grammar documentation.
JJTree	Invokes the JJTree preprocessor for the JavaCC compiler-compiler. It inserts parse-tree building actions at various places in the JavaCC source that it generates. The output of JJTree is run through JavaCC to create the parser. This task only invokes JJTree if the grammar file is newer than the generated JavaCC file.
Macrodef	Define a new task as a macro built-up upon other tasks.
Native2Ascii	Converts files from native encodings to ASCII with escaped Unicode. A common usage is to convert source files maintained in a native operating system encoding to ASCII, prior to compilation.
Presetdef	Define a new task by instrumenting an existing task with default values for attributes or child elements.
Translate	Identifies keys in files, delimited by special tokens, and translates them with values read from resource bundles.
XSLT	Processes a set of documents via XSLT.

Property Tasks

[Back to top]

Task Name	Description
Available	Sets a property if a specified file, directory, class in the classpath, or JVM system resource is available at runtime.
Basename	Sets a property to the last element of a specified path.
BuildNumber	Task that can be used to track build numbers.
Condition	Sets a property if a certain condition holds true; this is a generalization of Available and Uptodate.
Dirname	Sets a property to the value of the specified file up to, but not including, the last path element.
Echoproperties	Lists the current properties.
LoadFile	Loads a file into a property.
LoadProperties	Load a file's contents as Ant properties. This task is equivalent to using `<property file="..."/>` except that it supports nested `<filterchain>` elements, and it cannot be specified outside a target.
MakeURL	Creates a URL (list) from a file/fileset or path
PathConvert	Converts a nested path, path reference, filelist reference, or fileset reference to the form usable on a specified platform and/or to a list of items separated by the specified separator and stores the result in the specified property.
Property	Sets a property (by name and value), or set of properties (from a file or resource) in the project.
PropertyFile	Creates or modifies property files. Useful when wanting to make unattended modifications to configuration files for application servers and applications. Typically used for things such as automatically generating a build number and saving it to a build properties file, or doing date manipulation.
Uptodate	Sets a property if a given target file is newer than a set of source files.
Whichresource	Find a class or resource.
XmlProperty	Loads property values from a well-formed XML file.

Remote Tasks

[Back to top]

Task Name	Description
FTP	Implements a basic FTP client that can send, receive, list, and delete files, and create directories.
Rexec	Task to automate a remote rexec session.
Scp	Copy files to or from a remote server using SSH.
setproxy	Sets Java's web proxy properties, so that tasks and code run in the same JVM can have through-the-firewall access to remote web sites.
Sshexec	Execute a command on a remote server using SSH.
Telnet	Task to automate a remote *telnet* session. This task uses nested `<read>` and `<write>` tags to indicate strings to wait for and specify text to send.

SCM Tasks

[Back to top]

Task Name	Description
Cvs	Handles packages/modules retrieved from a CVS repository.
CvsChangeLog	Generates an XML report of the changes recorded in a CVS repository.
CVSPass	Adds entries to a .cvspass file. Adding entries to this file has the same affect as a *cvs login* command.
CvsTagDiff	Generates an XML-formatted report file of the changes between two tags or dates recorded in a CVS repository.
ClearCase	Tasks to perform the ClearCase cleartool *checkin*, *checkout*, *uncheckout*, *update*, *lock*, *unlock*, *mklbtype*, *rmtype*, *mklabel*, *mkattr*, *mkdir*, *mkelem*, and *mkbl* commands.
Continuus/Synergy	Tasks to perform the Continuus *ccmcheckin*, *ccmcheckout*, *ccmcheckintask*, *ccmreconfigure*, and *ccmcreateTask* commands.
Microsoft Visual SourceSafe	Tasks to perform the Visual SourceSafe *vssget*, *vsslabel*, *vsshistory*, *vsscheckin*, *vsscheckout*, *vssadd*, *vsscp*, and *vsscreate* commands.
Perforce	Tasks to perform the Perforce *p4sync*, *p4change*, *p4edit*, *p4submit*, *p4have*, *p4label*, *p4counter*, *p4reopen*, *p4revert*, and *p4add* commands.
Pvcs	Allows the user extract the latest edition of the source code from a PVCS repository.
SourceOffSite	Tasks to perform the SourceOffSite *sosget*, *soslabel*, *soscheckin*, and *soscheckout* commands.
StarTeam	Tasks to perform the StarTeam *stcheckout*, *stcheckin*, *stlabel*, and *stlist* commands. The Starteam task is deprecated; use STCheckout instead.

Testing Tasks

[Back to top]

Task Name	Description
Junit	Runs tests from the Junit testing framework. This task has been tested with JUnit 3.0 up to JUnit 3.7; it won't work with versions prior to JUnit 3.0.
JunitReport	Merges the individual XML files generated by the Junit task and applies a stylesheet on the resulting merged document to provide a browsable report of the testcases results.

Table of Contents

Core Tasks

GZip
Import
Include
Input
Jar
Java
Javac
Javadoc/*Javadoc2*
Length
LoadFile
LoadProperties
LoadResource
Local
MakeURL
Mail
MacroDef
Manifest
ManifestClassPath
Mkdir
Move
Nice
Parallel
Patch
PathConvert
PreSetDef
Property
PropertyHelper
Record
Rename
Replace
ResourceCount
Retry
Rmic
Sequential
SignJar
Sleep
Sql
Subant
Sync
Tar
Taskdef
Tempfile
Touch
Truncate
TStamp
Typedef
Unjar
Untar
Unwar
Unzip
Uptodate
Waitfor
WhichResource
War

XmlProperty
XSLT/*Style*
Zip

Table of Contents

Optional Tasks

ant.build.clonevm

Since Ant 1.7

The value of the ant.build.clonevm system property controls how Ant instruments forked Java Virtual Machines. The java and junit tasks support clonevm attributes to control the VMs on a task-by-task basis while the system property applies to all forked Java VMs.

If the value of the property is true, then all system properties of the forked Java Virtual Machine will be the same as those of the Java VM running Ant. In addition, if you set ant.build.clonevm to true and build.sysclasspath has not been set, the bootclasspath of forked Java VMs gets constructed as if build.sysclasspath had the value "last".

Note that this has to be a system property, so it cannot be specified on the Ant command line. Use the ANT_OPTS environment variable instead.

build.sysclasspath

The value of the build.sysclasspath property controls how the system classpath, i.e. the classpath in effect when Ant is run, affects the behavior of classpaths in Ant. The default behavior varies from task to task.

The values and their meanings are:

value	meaning
only	Only the system classpath is used and classpaths specified in build files, etc are ignored. This situation could be considered as the person running the build file knows more about the environment than the person writing the build file.
ignore	The system classpath is ignored. This situation is the reverse of the above. The person running the build trusts the build file writer to get the build file right. This mode is recommended for portable scripts.
last	The classpath is concatenated to any specified classpaths at the end. This is a compromise, where the build file writer has priority.
first	Any specified classpaths are concatenated to the system classpath. This is the other form of compromise where the build runner has priority.

Since Ant 1.7 the value of this property also affects the bootclasspath settings--it combines the bootclasspath that has been specified for a task with the bootclasspath of the Java VM running Ant. If the property has not been set, it defaults to "ignore" in this case.

The source and target attributes of <javac> don't have any default values for historical reasons. Since the underlying javac compiler's default depends on the JDK you use, you may encounter build files that don't explicitly set those attributes and that will no longer compile using a newer JDK. If you cannot change the build file, Ant provides two properties that help you setting default values for these attributes. If the attributes have been set explicitly, the properties listed here will be ignored.

ant.build.javac.source

Since Ant 1.7

Provides a default value for <javac>'s and <javadoc>'s source attribute.

ant.build.javac.target

Since Ant 1.7

Provides a default value for <javac>'s target attribute.

Common Attributes of all Tasks

All tasks share the following attributes:

Attribute	Description	Required
id	Unique identifier for this task instance, can be used to reference this task in scripts.	No
taskname	A different name for this task instance - will show up in the logging output.	No
description	Room for your comments	No

Description

Description

Allows for a description of the project to be specified that will be included in the output of the `ant -projecthelp` command.

Parameters

(none)

Examples

```
<description>
This buildfile is used to build the Foo subproject within
the large, complex Bar project.
</description>
```

Directory-based Tasks

Some tasks use directory trees for the actions they perform. For example, the javac task, which compiles a directory tree with `.java` files into `.class` files, is one of these directory-based tasks. Because some of these tasks do so much work with a directory tree, the task itself can act as an implicit FileSet.

Whether the fileset is implicit or not, it can often be very useful to work on a subset of the directory tree. This section describes how you can select a subset of such a directory tree when using one of these directory-based tasks.

Ant gives you two ways to create a subset of files in a fileset, both of which can be used at the same time:

- Only include files and directories that match any `include` patterns and do not match any `exclude` patterns in a given PatternSet.
- Select files based on selection criteria defined by a collection of selector nested elements.

Patternset

We said that Directory-based tasks can sometimes act as an implicit `<fileset>`, but in addition to that, a FileSet acts as an implicit `<patternset>`.

The inclusion and exclusion elements of the implicit PatternSet can be specified inside the directory-based task (or explicit fileset) via either:

- the attributes `includes` and `excludes`.
- nested elements `<include>` and `<exclude>`.
- external files specified with the attributes `includesfile` and `excludesfile`.
- external files specified with the nested elements `<includesfile>` and `<excludesfile>`.

When dealing with an external file, each line of the file is taken as a pattern that is added to the list of include or exclude patterns.

When both inclusion and exclusion are used, only files/directories that match at least one of the include patterns and don't match any of the exclude patterns are used. If no include pattern is given, all files are assumed to match the include pattern (with the possible exception of the default excludes).

Patterns

As described earlier, patterns are used for the inclusion and exclusion of files. These patterns look very much like the patterns used in DOS and UNIX:

'*' matches zero or more characters, '?' matches one character.

In general, patterns are considered relative paths, relative to a task dependent base directory (the dir attribute in the case of `<fileset>`). Only files found below that base directory are considered. So while a pattern like `../foo.java` is possible, it will not match anything when applied since the base directory's parent is never scanned for files.

Examples:

`*.java` matches `.java`, `x.java` and `FooBar.java`, but not `FooBar.xml` (does not end with `.java`).

`?.java` matches `x.java`, `A.java`, but not `.java` or `xyz.java` (both don't have one character before `.java`).

Combinations of *'s and ?'s are allowed.

Matching is done per-directory. This means that first the first directory in the pattern is matched against the first directory in the path to match. Then the second directory is matched, and so on. For example, when we have the pattern /?abc/*/*.java and the path /xabc/foobar/test.java, the first ?abc is matched with xabc, then * is matched with foobar, and finally *.java is matched with test.java. They all match, so the path matches the pattern.

To make things a bit more flexible, we add one extra feature, which makes it possible to match multiple directory levels. This can be used to match a complete directory tree, or a file anywhere in the directory tree. To do this, ** must be used as the name of a directory. When ** is used as the name of a directory in the pattern, it matches zero or more directories. For example: /test/** matches all files/directories under /test/, such as /test/x.java, or /test/foo/bar/xyz.html, but not /xyz.xml.

There is one "shorthand": if a pattern ends with / or \, then ** is appended. For example, mypackage/test/ is interpreted as if it were mypackage/test/**.

Example patterns:

**/CVS/*	Matches all files in CVS directories that can be located anywhere in the directory tree.
	Matches:
	`CVS/Repository`
	`org/apache/CVS/Entries`
	`org/apache/jakarta/tools/ant/CVS/Entries`
	But not:
	`org/apache/CVS/foo/bar/Entries (foo/bar/ part does not match)`

org/apache/jakarta/**	Matches all files in the org/apache/jakarta directory tree.
	Matches:
	`org/apache/jakarta/tools/ant/docs/index.html`
	`org/apache/jakarta/test.xml`
	But not:
	`org/apache/xyz.java`
	(jakarta/ part is missing).

org/apache/**/CVS/*	Matches all files in CVS directories that are located anywhere in the directory tree under org/apache.
	Matches:
	`org/apache/CVS/Entries`
	`org/apache/jakarta/tools/ant/CVS/Entries`
	But not:
	`org/apache/CVS/foo/bar/Entries`
	(foo/bar/ part does not match)

/test/	Matches all files that have a test element in their path, including test as a filename.

When these patterns are used in inclusion and exclusion, you have a powerful way to select just the files you want.

Selectors

The <fileset>, whether implicit or explicit in the directory-based task, also acts as an <and> selector container. This can be used to create arbitrarily complicated selection criteria for the files the task should work with. See the Selector documentation for more information.

Standard Tasks/Filesets

Many of the standard tasks in ant take one or more filesets which follow the rules given here. This list, a subset of those, is a list of standard ant tasks that can act as an implicit fileset:

- <checksum>
- <copydir> (deprecated)
- <delete>
- <dependset>
- <fixcrlf>
- <javac>
- <replace>
- <rmic>
- <style> (aka <xslt>)
- <tar>
- <zip>
- <ddcreator>
- <ejbjar>
- <ejbc>
- <cab>
- <native2ascii>
- <netrexxc>
- <renameextensions>
- <depend>
- <translate>
- <image>
- <jlink> (deprecated)
- <jspc>
- <wljspc>

Examples

```
<copy todir="${dist}">
  <fileset dir="${src}"
           includes="**/images/*"
           excludes="**/*.gif"
  />
</copy>
```

This copies all files in directories called `images` that are located in the directory tree defined by `${src}` to the destination directory defined by `${dist}`, but excludes all `*.gif` files from the copy.

```
<copy todir="${dist}">
  <fileset dir="${src}">
    <include name="**/images/*"/>
    <exclude name="**/*.gif"/>
  </fileset>
</copy>
```

The same as the example above, but expressed using nested elements.

```
<delete dir="${dist}">
    <include name="**/images/*"/>
    <exclude name="**/*.gif"/>
</delete>
```

Deleting the original set of files, the `delete` task can act as an implicit fileset.

Default Excludes

There are a set of definitions that are excluded by default from all directory-based tasks. They are:

```
     **/*~
     **/#*#
     **/.#*
     **/%*%
     **/._*
     **/CVS
     **/CVS/**
     **/.cvsignore
     **/SCCS
     **/SCCS/**
     **/vssver.scc
     **/.svn
     **/.svn/**
     **/.DS_Store
```

If you do not want these default excludes applied, you may disable them with the `defaultexcludes="no"` attribute.

This is the default list; note that you can modify the list of default excludes by using the defaultexcludes task.

DirSet

A DirSet is a group of directories. These directories can be found in a directory tree starting in a base directory and are matched by patterns taken from a number of PatternSets and Selectors.

PatternSets can be specified as nested `<patternset>` elements. In addition, DirSet holds an implicit PatternSet and supports the nested `<include>`, `<includesfile>`, `<exclude>` and `<excludesfile>` elements of `<patternset>` directly, as well as `<patternset>`'s attributes.

Selectors are available as nested elements within the DirSet. If any of the selectors within the DirSet do not select the directory, it is not considered part of the DirSet. This makes a DirSet equivalent to an `<and>` selector container.

Attribute	Description	Required
dir	The root of the directory tree of this DirSet.	Yes
includes	A comma- or space-separated list of patterns of directories that must be included; all directories are included when omitted.	No
includesfile	The name of a file; each line of this file is taken to be an include pattern.	No
excludes	A comma- or space-separated list of patterns of directories that must be excluded; no directories are excluded when omitted.	No
excludesfile	The name of a file; each line of this file is taken to be an exclude pattern.	No
casesensitive	Specifies whether case-sensitivity should be applied (`true`|`yes`|`on` or `false`|`no`|`off`).	No; defaults to true.
followsymlinks	Shall symbolic links be followed? Defaults to true. See fileset's documentation.	No

Examples

```
<dirset dir="${build.dir}">
  <include name="apps/**/classes"/>
  <exclude name="apps/**/*Test*"/>
</dirset>
```

Groups all directories named `classes` found under the `apps` subdirectory of `${build.dir}`, except those that have the text `Test` in their name.

```
<dirset dir="${build.dir}">
  <patternset id="non.test.classes">
    <include name="apps/**/classes"/>
    <exclude name="apps/**/*Test*"/>
  </patternset>
</dirset>
```

Groups the same directories as the above example, but also establishes a PatternSet that can be referenced in other `<dirset>` elements, rooted at a different directory.

```
<dirset dir="${debug_build.dir}">
  <patternset refid="non.test.classes"/>
</dirset>
```

Groups all directories in directory `${debug_build.dir}`, using the same patterns as the above example.

```
<dirset id="dirset" dir="${workingdir}">
   <present targetdir="${workingdir}">
        <mapper type="glob" from="*" to="*/${markerfile}" />
   </present>
</dirset>
```

Selects all directories somewhere under `${workingdir}` which contain a `${markerfile}`.

FileList

FileLists are explicitly named lists of files. Whereas FileSets act as filters, returning only those files that exist in the file system and match specified patterns, FileLists are useful for specifying files that may or may not exist. Multiple files are specified as a list of files, relative to the specified directory, with no support for wildcard expansion (filenames with wildcards will be included in the list unchanged). FileLists can appear inside tasks that support this feature or as stand-alone types.

Attribute	Description	Required
dir	The base directory of this FileList.	Yes
files	The list of file names. This is a list of file name separated by whitespace, or by commas.	Yes, unless there is a nested file element

Nested Element: file

This represents a file name. The nested element allows filenames containing white space and commas.

Since Ant 1.6.2

Attribute	Description	Required
name	The name of the file.	Yes

Examples

```
<filelist
    id="docfiles"
    dir="${doc.src}"
    files="foo.xml,bar.xml"/>
```

The files `${doc.src}/foo.xml` and `${doc.src}/bar.xml`. Note that these files may not (yet) actually exist.

```
<filelist
    id="docfiles"
    dir="${doc.src}"
    files="foo.xml
           bar.xml"/>
```

Same files as the example above.

```
<filelist refid="docfiles"/>
```

Same files as the example above.

```
<filelist
    id="docfiles"
    dir="${doc.src}">
    <file name="foo.xml"/>
    <file name="bar.xml"/>
</filelist>
```

Same files as the example above.

FileSet

A FileSet is a group of files. These files can be found in a directory tree starting in a base directory and are matched by patterns taken from a number of PatternSets and Selectors.

PatternSets can be specified as nested `<patternset>` elements. In addition, FileSet holds an implicit PatternSet and supports the nested `<include>`, `<includesfile>`, `<exclude>` and `<excludesfile>` elements of PatternSet directly, as well as PatternSet's attributes.

Selectors are available as nested elements within the FileSet. If any of the selectors within the FileSet do not select the file, the file is not considered part of the FileSet. This makes a FileSet equivalent to an `<and>` selector container.

Attribute	Description	Required
dir	the root of the directory tree of this FileSet.	Either dir or file must be specified
file	shortcut for specifying a single-file fileset	
defaultexcludes	indicates whether default excludes should be used or not (`yes | no`); default excludes are used when omitted.	No
includes	comma- or space-separated list of patterns of files that must be included; all files are included when omitted.	No
includesfile	the name of a file; each line of this file is taken to be an include pattern.	No
excludes	comma- or space-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file; each line of this file is taken to be an exclude pattern.	No
casesensitive	Must the include and exclude patterns be treated in a case sensitive way? Defaults to true.	No
followsymlinks	Shall symbolic links be followed? Defaults to true. See the note below.	No
erroronmissingdir	Specify what happens if the base directory does not exist. If true a build error will happen, if false, the fileset will be ignored/empty. Defaults to true. *Since Ant 1.7.1 (default is true for backward compatibility reasons.)*	No

Note: All files/directories for which the canonical path is different from its path are considered symbolic links. On Unix systems this usually means the file really is a symbolic link but it may lead to false results on other platforms.

Examples

```
<fileset dir="${server.src}" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</fileset>
```

Groups all files in directory `${server.src}` that are Java source files and don't have the text `Test` in their name.

```
<fileset dir="${server.src}" casesensitive="yes">
  <patternset id="non.test.sources">
    <include name="**/*.java"/>
    <exclude name="**/*Test*"/>
  </patternset>
</fileset>
```

Groups the same files as the above example, but also establishes a PatternSet that can be referenced in other `<fileset>` elements, rooted at a different directory.

```
<fileset dir="${client.src}" >
  <patternset refid="non.test.sources"/>
</fileset>
```

Groups all files in directory `${client.src}`, using the same patterns as the above example.

```
<fileset dir="${server.src}" casesensitive="yes">
  <filename name="**/*.java"/>
  <filename name="**/*Test*" negate="true"/>
</fileset>
```

Groups the same files as the top example, but using the `<filename>` selector.

```
<fileset dir="${server.src}" casesensitive="yes">
  <filename name="**/*.java"/>
  <not>
    <filename name="**/*Test*"/>
  </not>
</fileset>
```

Groups the same files as the previous example using a combination of the `<filename>` selector and the `<not>` selector container.

```
<fileset dir="src" includes="main/" />
```

Selects all files in *src/main* (e.g. *src/main/Foo.java* or *src/main/application/Bar.java*).

Mapping File Names

Some tasks take source files and create target files. Depending on the task, it may be quite obvious which name a target file will have (using javac, you know there will be `.class` files for your `.java` files) - in other cases you may want to specify the target files, either to help Ant or to get an extra bit of functionality.

While source files are usually specified as filesets, you don't specify target files directly - instead, you tell Ant how to find the target file(s) for one source file. An instance of `org.apache.tools.ant.util.FileNameMapper` is responsible for this. It constructs target file names based on rules that can be parameterized with `from` and `to` attributes - the exact meaning of which is implementation-dependent.

These instances are defined in `<mapper>` elements with the following attributes:

Attribute	Description	Required
type	specifies one of the built-in implementations.	Exactly one of these
classname	specifies the implementation by class name.	
classpath	the classpath to use when looking up `classname`.	No
classpathref	the classpath to use, given as reference to a path defined elsewhere.	No
from	the `from` attribute for the given implementation.	Depends on implementation.
to	the `to` attribute for the given implementation.	Depends on implementation.

Note that Ant will not automatically convert / or \ characters in the `to` and `from` attributes to the correct directory separator of your current platform. If you need to specify this separator, use `${file.separator}` instead. For the regexpmapper, `${file.separator}` will not work, as on windows it is the '\' character, and this is an escape character for regular expressions, one should use the `handledirsep` attribute instead.

Parameters specified as nested elements

The classpath can be specified via a nested `<classpath>`, as well - that is, a path-like structure.

Since Ant 1.7.0, nested File Mappers can be supplied via either `<mapper>` elements or `<typedef>`'d implementations of `org.apache.tools.ant.util.FileNameMapper`. If nested File Mappers are specified by either means, the mapper will be implicitly configured as a composite mapper.

The built-in mapper types are:

All built-in mappers are case-sensitive.

As of Ant 1.7.0, each of the built-in mapper implementation types is directly accessible using a specific tagname. This makes it possible for filename mappers to support attributes in addition to the generally available *to* and *from*.
The `<mapper type|classname="...">` usage form remains valid for reasons of backward compatibility.

identity

The target file name is identical to the source file name. Both `to` and `from` will be ignored.

Examples:

```
<mapper type="identity"/>
<identitymapper/>
```

Source file name	Target file name
A.java	A.java
foo/bar/B.java	foo/bar/B.java
C.properties	C.properties
Classes/dir/dir2/A.properties	Classes/dir/dir2/A.properties

flatten

The target file name is identical to the source file name, with all leading directory information stripped off. Both `to` and `from` will be ignored.

Examples:

```
<mapper type="flatten"/>
<flattenmapper/>
```

Source file name	Target file name
A.java	A.java
foo/bar/B.java	B.java
C.properties	C.properties
Classes/dir/dir2/A.properties	A.properties

merge

The target file name will always be the same, as defined by `to` - `from` will be ignored.

Examples:

```
<mapper type="merge" to="archive.tar"/>
<mergemapper to="archive.tar"/>
```

Source file name	Target file name
A.java	archive.tar
foo/bar/B.java	archive.tar
C.properties	archive.tar
Classes/dir/dir2/A.properties	archive.tar

glob

Both `to` and `from` are required and define patterns that may contain at most one `*`. For each source file that matches the `from` pattern, a target file name will be constructed from the `to` pattern by substituting the `*` in the `to` pattern with the text that matches the `*` in the `from` pattern. Source file names that don't match the `from` pattern will be ignored.

Examples:

```
<mapper type="glob" from="*.java" to="*.java.bak"/>
<globmapper from="*.java" to="*.java.bak"/>
```

Source file name	Target file name
A.java	A.java.bak
foo/bar/B.java	foo/bar/B.java.bak

C.properties	ignored
Classes/dir/dir2/A.properties	ignored

```
<mapper type="glob" from="C*ies" to="Q*y"/>
<globmapper from="C*ies" to="Q*y"/>
```

Source file name	Target file name
A.java	ignored
foo/bar/B.java	ignored
C.properties	Q.property
Classes/dir/dir2/A.properties	Qlasses/dir/dir2/A.property

The globmapper mapper can take the following extra attributes.

Attribute	Description	Required
casesensitive	If this is false, the mapper will ignore case when matching the glob pattern. This attribute can be true or false, the default is true. *Since Ant 1.6.3.*	No
handledirsep	If this is specified, the mapper will ignore the difference between the normal directory separator characters - \ and /. This attribute can be true or false, the default is false. This attribute is useful for cross-platform build files. *Since Ant 1.6.3.*	No

An example:

```
<pathconvert property="x" targetos="unix">
  <path path="Aj.Java"/>
  <mapper>
  <chainedmapper>
    <flattenmapper/>
    <globmapper from="a*.java" to="*.java.bak" casesensitive="no"/>
  </chainedmapper>
  </mapper>
</pathconvert>
<echo>x is ${x}</echo>
```

will output "x is j.java.bak".

and

```
<pathconvert property="x" targetos="unix">
  <path path="d/e/f/j.java"/>
  <mapper>
    <globmapper from="${basedir}\d/e\*" to="*" ignoredirchar="yes"/>
  </mapper>
</pathconvert>
<echo>x is ${x}</echo>
```

will output "x is f/j.java".

regexp

Both `to` and `from` are required and define regular expressions. If the source file name matches the `from` pattern, the target file name will be constructed from the `to` pattern, using `\0` to `\9` as back-references for the full match (`\0`) or the matches of the subexpressions in parentheses. Source files not matching the `from` pattern will be ignored.

Note that you need to escape a dollar-sign (`$`) with another dollar-sign in Ant.

The regexp mapper needs a supporting library and an implementation of

`org.apache.tools.ant.util.regexp.RegexpMatcher` that hides the specifics of the library. Ant comes with implementations for the java.util.regex package of JDK 1.4 or higher, jakarta-regexp and jakarta-ORO. If you compile from sources and plan to use one of them, make sure the libraries are in your `CLASSPATH`. For information about using gnu.regexp or gnu.rex with Ant, see this article.

If you want to use one of the supported regular expression libraries you need to also use the corresponding `ant-[jakarta-oro, jakarta-regexp, apache-oro, apache-regexp}.jar` from the Ant release you are using. Make sure, both will be loaded from the same classpath, that is either put them into your `CLASSPATH`, `ANT_HOME/lib` directory or a nested `<classpath>` element of the mapper - you cannot have `ant-[jakarta-oro, jakarta-regexp, apache-oro, apache-regexp].jar` in `ANT_HOME/lib` and the library in a nested `<classpath>`.

Ant will choose the regular-expression library based on the following algorithm:

- If the system property `ant.regexp.matcherimpl` has been set, it is taken as the name of the class implementing `org.apache.tools.ant.util.regexp.RegexpMatcher` that should be used.
- If it has not been set, uses the JDK 1.4 classes.

Examples:

```
<mapper type="regexp" from="^(.*)\.java$$" to="\1.java.bak"/>
<regexpmapper from="^(.*)\.java$$" to="\1.java.bak"/>
```

Source file name	Target file name
A.java	A.java.bak
foo/bar/B.java	foo/bar/B.java.bak
C.properties	ignored
Classes/dir/dir2/A.properties	ignored

```
<mapper type="regexp" from="^(.*)/([^/]+)/([^/]*)$$" to="\1/\2/\2-\3"/>
<regexpmapper from="^(.*)/([^/]+)/([^/]*)$$" to="\1/\2/\2-\3"/>
```

Source file name	Target file name
A.java	ignored
foo/bar/B.java	foo/bar/bar-B.java
C.properties	ignored
Classes/dir/dir2/A.properties	Classes/dir/dir2/dir2-A.properties

```
<mapper type="regexp" from="^(.*)\.(.*)$$" to="\2.\1"/>
<regexpmapper from="^(.*)\.(.*)$$&" to="\2.\1"/>
```

Source file name	Target file name
A.java	java.A
foo/bar/B.java	java.foo/bar/B
C.properties	properties.C
Classes/dir/dir2/A.properties	properties.Classes/dir/dir2/A

```
<mapper type="regexp" from="^(.*?)(\$$[^/\\\.]*)?\.class$$" to="\1.java"/>
<regexpmapper from="^(.*?)(\$$[^/\\\.]*)?\.class$$" to="\1.java"/>
```

Source file name	Target file name
ClassLoader.class	ClassLoader.java
java/lang/ClassLoader.class	java/lang/ClassLoader.java
java\lang\ClassLoader$1.class	java\lang\ClassLoader.java
java/lang/ClassLoaderfoo1.class	java/lang/ClassLoader.java

The regexpmapper mapper can take the following extra attributes.

Attribute	Description	Required
casesensitive	If this is false, the mapper will ignore case when matching the pattern. This attribute can be true or false, the default is true. *Since Ant 1.6.3.*	No
handledirsep	If this is specified, the mapper will treat a \ character in a filename as a / for the purposes of matching. This attribute can be true or false, the default is false. This attribute is useful for cross-platform build files. *Since Ant 1.6.3.*	No

An example:

```
    <pathconvert property="x" targetos="unix">
      <path path="Aj.Java"/>
      <chainedmapper>
        <flattenmapper/>
        <regexpmapper from="a(.*)\.java" to="\1.java.bak" casesensitive="no"/>
      </chainedmapper>
    </pathconvert>
    <echo>x is ${x}</echo>
```

will output "x is j.java.bak".

and

```
  <pathconvert property="hd.prop" targetos="windows">
    <path path="d\e/f\j.java"/>
    <chainedmapper>
      <regexpmapper from="${basedir}/d/e/(.*)" to="\1" handledirsep="yes"/>
    </chainedmapper>
  </pathconvert>
```

will set `hd.prop` to "f\j.java".

package

Sharing the same syntax as the glob mapper, the package mapper replaces directory separators found in the matched source pattern with dots in the target pattern placeholder. This mapper is particularly useful in combination with `<uptodate>` and `<junit>` output.

The to and from attributes are both required.

Example:

```
<mapper type="package" from="*Test.java" to="TEST-*Test.xml"/>
<packagemapper from="*Test.java" to="TEST-*Test.xml"/>
```

Source file name	Target file name
`org/apache/tools/ant/util/PackageMapperTest.java`	`TEST-org.apache.tools.ant.util.PackageMapperTest.xml`
`org/apache/tools/ant/util/Helper.java`	ignored

unpackage (since Ant 1.6.0)

This mapper is the inverse of the package mapper. It replaces the dots in a package name with directory separators. This is useful for matching XML formatter results against their JUnit test test cases. The mapper shares the sample syntax as the glob mapper.

The to and from attributes are both required.

Example:

```
<mapper type="unpackage" from="TEST-*Test.xml" to="${test.src.dir}/*Test.java">
<unpackagemapper from="TEST-*Test.xml" to="${test.src.dir}/*Test.java">
```

Source file name	Target file name
TEST-org.acme.AcmeTest.xml	${test.src.dir}/org/acme/AcmeTest.java

composite (since Ant 1.7.0)

This mapper implementation can contain multiple nested mappers. File mapping is performed by passing the source filename to each nested <mapper> in turn, returning all results. The *to* and *from* attributes are ignored.

Starting with Ant 1.8.0 the order of the mapped results is the same as the order of the nested mappers; prior to Ant 1.8.0 the order has been undefined.

Examples:

```
<compositemapper>
  <identitymapper/>
  <packagemapper from="*.java" to="*"/>
</compositemapper>
```

Source file name	Target file names
foo/bar/A.java	foo/bar/A.java
	foo.bar.A

The composite mapper has no corresponding <mapper **type**> attribute.

chained (since Ant 1.7.0)

This mapper implementation can contain multiple nested mappers. File mapping is performed by passing the source filename to the first nested mapper, its results to the second, and so on. The target filenames generated by the last nested mapper comprise the ultimate results of the mapping operation. The *to* and *from* attributes are ignored.

Examples:

```
<chainedmapper>
  <flattenmapper/>
  <globmapper from="*" to="new/path/*"/>
  <mapper>
    <globmapper from="*" to="*1"/>
    <globmapper from="*" to="*2"/>
  </mapper>
</chainedmapper>
```

Source file name	Target file names
foo/bar/A.java	new/path/A.java1
	new/path/A.java2
boo/far/B.java	new/path/B.java1
	new/path/B.java2

The chained mapper has no corresponding <mapper **type**> attribute.

filtermapper (since Ant 1.6.3)

This mapper implementation applies a filterchain to the source file name.

Examples:

```
<filtermapper>
  <replacestring from="\" to="/"/>
</filtermapper>
```

Source file name	Target file names
foo\bar\A.java	foo/bar/A.java

```
<filtermapper>
  <scriptfilter language="beanshell">
    self.setToken(self.getToken().toUpperCase());
  </scriptfilter>
</filtermapper>
```

Source file name	Target file names
foo\bar\A.java	FOO\BAR\A.JAVA

The filtermapper has no corresponding `<mapper type>` attribute.

scriptmapper (since Ant 1.7)

This mapper executes a script written in Apache BSF or JSR 223 supported language, once per file to map.

The script can be declared inline or in a specified file.

See the Script task for an explanation of scripts and dependencies.

Attribute	Description	Required
language	Scripting language	Yes
manager	The script engine manager to use. See the script task for using this attribute.	No - default is "auto"
src	File containing the script	No
setbeans	whether to have all properties, references and targets as global variables in the script. *since Ant 1.8.0*	No, default is "true".
classpath	The classpath to pass into the script.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No

This filename mapper can take a nested <classpath> element. See the script task on how to use this element.

Example:

```
<scriptmapper language="javascript">
  self.addMappedName(source.toUpperCase());
  self.addMappedName(source.toLowerCase());
</scriptmapper>
```

Source file name	Target file names
foo\bar\A.java	FOO\BAR\A.JAVA
	foo\bar\a.java

To use this mapper, the scripts need access to the source file, and the ability to return multiple mappings. Here are the relevant beans and their methods. The script is called once for every source file, with the list of mapped names reset

after every invocation.

Script bean	Description
`source: String`	The file/path to map
`self`	the scriptmapper itself
`self.addMappedName(String name)`	Add a new mapping
`self.clear()`	Reset the list of files.

The scriptmapper has no corresponding `<mapper type>` attribute.

firstmatchmapper (since Ant 1.8.0)

This mapper supports an arbitrary number of nested mappers and returns the results of the first mapper that matches. This is different from composite mapper which collects the results of all matching children.

Examples:

```
<firstmatchmapper>
  <globmapper from="*.txt" to="*.bak"/>
  <globmapper from="*A.*" to="*B.*"/>
</firstmatchmapper>
```

Source file name	Target file names
`foo/bar/A.txt`	`foo/bar/A.bak`
`foo/bar/A.java`	`foo/bar/B.java`

The firstmatchmapper has no corresponding `<mapper type>` attribute.

FilterChains and FilterReaders

Consider the flexibility of Unix pipes. If you wanted, for example, to copy just those lines that contained the string blee from the first 10 lines of a text file 'foo' (*you wouldn't want to filter a binary file*) to a file 'bar', you would do something like:

```
cat foo|head -n10|grep blee > bar
```

Ant was not flexible enough. There was no way for the `<copy>` task to do something similar. If you wanted the `<copy>` task to get the first 10 lines, you would have had to create special attributes:

```
<copy file="foo" tofile="bar" head="10" contains="blee"/>
```

The obvious problem thus surfaced: Ant tasks would not be able to accommodate such data transformation attributes as they would be endless. The task would also not know in which order these attributes were to be interpreted. That is, must the task execute the contains attribute first and then the head attribute or vice-versa? What Ant tasks needed was a mechanism to allow pluggable filter (data transformer) chains. Ant would provide a few filters for which there have been repeated requests. Users with special filtering needs would be able to easily write their own and plug them in.

The solution was to refactor data transformation oriented tasks to support FilterChains. A FilterChain is a group of ordered FilterReaders. Users can define their own FilterReaders by just extending the java.io.FilterReader class. Such custom FilterReaders can be easily plugged in as nested elements of `<filterchain>` by using `<filterreader>` elements.

Example:

```
<copy file="${src.file}" tofile="${dest.file}">
  <filterchain>
    <filterreader classname="your.extension.of.java.io.FilterReader">
      <param name="foo" value="bar"/>
    </filterreader>
    <filterreader classname="another.extension.of.java.io.FilterReader">
      <classpath>
        <pathelement path="${classpath}"/>
      </classpath>
      <param name="blah" value="blee"/>
      <param type="abra" value="cadabra"/>
    </filterreader>
  </filterchain>
</copy>
```

Ant provides some built-in filter readers. These filter readers can also be declared using a syntax similar to the above syntax. However, they can be declared using some simpler syntax also.

Example:

```
<loadfile srcfile="${src.file}" property="src.file.head">
  <filterchain>
    <headfilter lines="15"/>
  </filterchain>
</loadfile>
```

is equivalent to:

```
<loadfile srcfile="${src.file}" property="src.file.head">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.HeadFilter">
      <param name="lines" value="15"/>
    </filterreader>
  </filterchain>
</loadfile>
```

The following built-in tasks support nested `<filterchain>` elements.
Concat,
Copy,
LoadFile,
LoadProperties,
Move

A FilterChain is formed by defining zero or more of the following nested elements.
FilterReader
ClassConstants
EscapeUnicode
ExpandProperties
HeadFilter
LineContains
LineContainsRegExp
PrefixLines
ReplaceTokens
StripJavaComments
StripLineBreaks
StripLineComments
SuffixLines
TabsToSpaces
TailFilter
DeleteCharacters
ConcatFilter
TokenFilter
FixCRLF
SortFilter

FilterReader

The filterreader element is the generic way to define a filter. User defined filter elements are defined in the build file using this. Please note that built in filter readers can also be defined using this syntax. A FilterReader element must be supplied with a class name as an attribute value. The class resolved by this name must extend java.io.FilterReader. If the custom filter reader needs to be parameterized, it must implement org.apache.tools.type.Parameterizable.

Attribute	Description	Required
classname	The class name of the filter reader.	Yes

Nested Elements:

`<filterreader>` supports `<classpath>` and `<param>` as nested elements. Each `<param>` element may take in the following attributes - name, type and value.

The following FilterReaders are supplied with the default distribution.

ClassConstants

This filters basic constants defined in a Java Class, and outputs them in lines composed of the format *name=value*. This filter uses the *bcel* library to understand the Java Class file. See Library Dependencies.

Important: This filter is different from most of the other filters. Most of the filters operate on a sequence of characters. This filter operates on the sequence of bytes that makes up a class. However the bytes arrive to the filter as a sequence

of characters. This means that one must be careful on the choice of character encoding to use. Most encoding lose information on conversion from an arbitary sequence of bytes to characters and back again to bytes. In particular the usual default character encodings (CP152 and UTF-8) do. For this reason, *since Ant 1.7*, the character encoding **ISO-8859-1** is used to convert from characters back to bytes, so one **has** to use this encoding for reading the java class file.

Example:

This loads the basic constants defined in a Java class as Ant properties.

```
<loadproperties srcfile="foo.class" encoding="ISO-8859-1">
  <filterchain>
    <classconstants/>
  </filterchain>
</loadproperties>
```

This loads the constants from a Java class file as Ant properties, prepending the names with a prefix.

```
<loadproperties srcfile="build/classes/org/acme/bar.class"
                encoding="ISO-8859-1">
  <filterchain>
    <classconstants/>
    <prefixlines prefix="ini."/>
  </filterchain>
</loadproperties>
```

EscapeUnicode

This filter converts its input by changing all non US-ASCII characters into their equivalent unicode escape backslash u plus 4 digits.

since Ant 1.6

Example:

This loads the basic constants defined in a Java class as Ant properties.

```
<loadproperties srcfile="non_ascii_property.properties">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.EscapeUnicode"/>
  </filterchain>
</loadproperties>
```

Convenience method:

```
<loadproperties srcfile="non_ascii_property.properties">
  <filterchain>
    <escapeunicode/>
  </filterchain>
</loadproperties>
```

ExpandProperties

If the data contains data that represents Ant properties (of the form ${...}), that is substituted with the property's actual value.

Example:

This results in the property modifiedmessage holding the value "All these moments will be lost in time, like teardrops in the rain"

```
<echo
  message="All these moments will be lost in time, like teardrops in the ${weather}"
  file="loadfile1.tmp"
  />
<property name="weather" value="rain"/>
<loadfile property="modifiedmessage" srcFile="loadfile1.tmp">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.ExpandProperties"/>
  </filterchain>
</loadfile>
```

Convenience method:

```
<echo
  message="All these moments will be lost in time, like teardrops in the ${weather}"
  file="loadfile1.tmp"
  />
<property name="weather" value="rain"/>
<loadfile property="modifiedmessage" srcFile="loadfile1.tmp">
  <filterchain>
    <expandproperties/>
  </filterchain>
</loadfile>
```

HeadFilter

This filter reads the first few lines from the data supplied to it.

Parameter Name	Parameter Value	Required
lines	Number of lines to be read. Defaults to "10" A negative value means that all lines are passed (useful with *skip*)	No
skip	Number of lines to be skipped (from the beginning). Defaults to "0"	No

Example:

This stores the first 15 lines of the supplied data in the property src.file.head

```
<loadfile srcfile="${src.file}" property="src.file.head">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.HeadFilter">
      <param name="lines" value="15"/>
    </filterreader>
  </filterchain>
</loadfile>
```

Convenience method:

```
<loadfile srcfile="${src.file}" property="src.file.head">
  <filterchain>
    <headfilter lines="15"/>
  </filterchain>
</loadfile>
```

This stores the first 15 lines, skipping the first 2 lines, of the supplied data in the property src.file.head. (Means: lines 3-17)

```
<loadfile srcfile="${src.file}" property="src.file.head">
  <filterchain>
    <headfilter lines="15" skip="2"/>
  </filterchain>
</loadfile>
```

See the testcases for more examples (*src\etc\testcases\filters\head-tail.xml* in the source distribution).

LineContains

This filter includes only those lines that contain all the user-specified strings.

Parameter Type	Parameter Value	Required
contains	Substring to be searched for.	Yes
negate	Whether to select *non*-matching lines only. **Since Ant 1.7**	No

Example:

This will include only those lines that contain `foo` and `bar`.

```
<filterreader classname="org.apache.tools.ant.filters.LineContains">
  <param type="contains" value="foo"/>
  <param type="contains" value="bar"/>
</filterreader>
```

Convenience method:

```
<linecontains>
  <contains value="foo"/>
  <contains value="bar"/>
</linecontains>
```

Negation:

```
<filterreader classname="org.apache.tools.ant.filters.LineContains">
  <param type="negate" value="true"/>
  <param type="contains" value="foo"/>
  <param type="contains" value="bar"/>
</filterreader>
```

or

```
<linecontains negate="true">
  <contains value="foo"/>
  <contains value="bar"/>
</linecontains>
```

LineContainsRegExp

Filter which includes only those lines that contain the user-specified regular expression matching strings.

Parameter Type	Parameter Value	Required
regexp	Regular expression to be searched for.	Yes
negate	Whether to select *non*-matching lines only. **Since Ant 1.7**	No

See Regexp Type for the description of the nested element regexp and of the choice of regular expression implementation.

Example:

This will fetch all those lines that contain the pattern `foo`

```
<filterreader classname="org.apache.tools.ant.filters.LineContainsRegExp">
  <param type="regexp" value="foo*"/>
</filterreader>
```

Convenience method:

```
<linecontainsregexp>
  <regexp pattern="foo*"/>
</linecontainsregexp>
```

Negation:

```
<filterreader classname="org.apache.tools.ant.filters.LineContainsRegExp">
  <param type="negate" value="true"/>
  <param type="regexp" value="foo*"/>
</filterreader>
```

or

```
<linecontainsregexp negate="true">
  <regexp pattern="foo*"/>
</linecontainsregexp>
```

PrefixLines

Attaches a prefix to every line.

Parameter Name	Parameter Value	Required
prefix	Prefix to be attached to lines.	Yes

Example:

This will attach the prefix `Foo` to all lines.

```
<filterreader classname="org.apache.tools.ant.filters.PrefixLines">
  <param name="prefix" value="Foo"/>
</filterreader>
```

Convenience method:

```
<prefixlines prefix="Foo"/>
```

SuffixLines

Attaches a suffix to every line.

since Ant 1.8.0

Parameter Name	Parameter Value	Required
suffix	Suffix to be attached to lines.	Yes

Example:

This will attach the suffix `Foo` to all lines.

```
<filterreader classname="org.apache.tools.ant.filters.SuffixLines">
  <param name="suffix" value="Foo"/>
</filterreader>
```

Convenience method:

```
<suffixlines suffix="Foo"/>
```

ReplaceTokens

This filter reader replaces all strings that are sandwiched between begintoken and endtoken with user defined values.

Parameter Type	Parameter Name	Parameter Value	Required

tokenchar	begintoken	Character marking the beginning of a token. Defaults to @	No
tokenchar	endtoken	Character marking the end of a token. Defaults to @	No
User defined String.	token	User defined search String.	Yes
Not applicable.	propertiesfile	Properties file to take tokens from.	No
User defined String.	value	Replace-value for the token	No

Example:

This replaces occurrences of the string @DATE@ in the data with today's date and stores it in the property ${src.file.replaced}

```
<tstamp/>
<loadfile srcfile="${src.file}" property="${src.file.replaced}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.ReplaceTokens">
      <param type="token" name="DATE" value="${TODAY}"/>
    </filterreader>
  </filterchain>
</loadfile>
```

Convenience method:

```
<tstamp/>
<loadfile srcfile="${src.file}" property="${src.file.replaced}">
  <filterchain>
    <replacetokens>
      <token key="DATE" value="${TODAY}"/>
    </replacetokens>
  </filterchain>
</loadfile>
```

This will treat each properties file entry in sample.properties as a token/key pair :

```
<loadfile srcfile="${src.file}" property="${src.file.replaced}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.ReplaceTokens">
      <param type="propertiesfile" value="sample.properties"/>
    </filterreader>
  </filterchain>
</loadfile>
</filterchain>
```

StripJavaComments

This filter reader strips away comments from the data, using Java syntax guidelines. This filter does not take in any parameters.

Example:

```
<loadfile srcfile="${java.src.file}" property="${java.src.file.nocomments}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.StripJavaComments"/>
  </filterchain>
</loadfile>
```

Convenience method:

```
<loadfile srcfile="${java.src.file}" property="${java.src.file.nocomments}">
  <filterchain>
    <stripjavacomments/>
  </filterchain>
</loadfile>
```

StripLineBreaks

This filter reader strips away specific characters from the data supplied to it.

Parameter Name	Parameter Value	Required
linebreaks	Characters that are to be stripped out. Defaults to "\r\n"	No

Examples:

This strips the '\r' and '\n' characters.

```
<loadfile srcfile="${src.file}" property="${src.file.contents}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.StripLineBreaks"/>
  </filterchain>
</loadfile>
```

Convenience method:

```
<loadfile srcfile="${src.file}" property="${src.file.contents}">
  <filterchain>
    <striplinebreaks/>
  </filterchain>
</loadfile>
```

This treats the '(' and ')' characters as line break characters and strips them.

```
<loadfile srcfile="${src.file}" property="${src.file.contents}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.StripLineBreaks">
      <param name="linebreaks" value="()"/>
    </filterreader>
  </filterchain>
</loadfile>
```

StripLineComments

This filter removes all those lines that begin with strings that represent comments as specified by the user.

Parameter Type	Parameter Value	Required
comment	Strings that identify a line as a comment when they appear at the start of the line.	Yes

Examples:

This removes all lines that begin with #, --, REM, rem and //

```
<filterreader classname="org.apache.tools.ant.filters.StripLineComments">
  <param type="comment" value="#"/>
  <param type="comment" value="--"/>
  <param type="comment" value="REM "/>
  <param type="comment" value="rem "/>
  <param type="comment" value="//"/>
</filterreader>
```

Convenience method:

```
<striplinecomments>
  <comment value="#"/>
  <comment value="--"/>
  <comment value="REM "/>
  <comment value="rem "/>
  <comment value="//"/>
</striplinecomments>
```

TabsToSpaces

This filter replaces tabs with spaces

Parameter Name	Parameter Value	Required
tablength	Defaults to "8"	No

Examples:

This replaces tabs in ${src.file} with spaces.

```
<loadfile srcfile="${src.file}" property="${src.file.notab}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.TabsToSpaces"/>
  </filterchain>
</loadfile>
```

Convenience method:

```
<loadfile srcfile="${src.file}" property="${src.file.notab}">
  <filterchain>
    <tabstospaces/>
  </filterchain>
</loadfile>
```

TailFilter

This filter reads the last few lines from the data supplied to it.

Parameter Name	Parameter Value	Required
lines	Number of lines to be read. Defaults to "10" A negative value means that all lines are passed (useful with *skip*)	No
skip	Number of lines to be skipped (from the end). Defaults to "0"	No

Background:

With HeadFilter and TailFilter you can extract each part of a text file you want. This graphic shows the dependencies:

Content	Filter
Line 1	

Line 2	`<filterchain>`
	`  <headfilter lines="2"/>`
	`</filterchain>`

Line 3	`<filterchain>`
	`  <tailfilter lines="-1" skip="2"/>`
	`</filterchain>`

Line 4	`<filterchain>`
	`  <headfilter lines="-1" skip="2"/>`
	`</filterchain>`

Line 5	`<filterchain>`
	`  <headfilter lines="-1" skip="2"/>`
	`  <tailfilter lines="-1" skip="2"/>`
	`</filterchain>`

Lines ...	`<filterchain>`
	`  <tailfilter lines="2"/>`
	`</filterchain>`

Line 95	
Line 96	
Line 97	
Line 98	
Line 99	

Examples:

This stores the last 15 lines of the supplied data in the property ${src.file.tail}

```
<loadfile srcfile="${src.file}" property="${src.file.tail}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.TailFilter">
      <param name="lines" value="15"/>
    </filterreader>
  </filterchain>
</loadfile>
```

Convenience method:

```
<loadfile srcfile="${src.file}" property="${src.file.tail}">
  <filterchain>
    <tailfilter lines="15"/>
  </filterchain>
</loadfile>
```

This stores the last 5 lines of the first 15 lines of the supplied data in the property ${src.file.mid}

```
<loadfile srcfile="${src.file}" property="${src.file.mid}">
  <filterchain>
    <filterreader classname="org.apache.tools.ant.filters.HeadFilter">
      <param name="lines" value="15"/>
    </filterreader>
    <filterreader classname="org.apache.tools.ant.filters.TailFilter">
      <param name="lines" value="5"/>
    </filterreader>
  </filterchain>
</loadfile>
```

Convenience method:

```
<loadfile srcfile="${src.file}" property="${src.file.mid}">
  <filterchain>
    <headfilter lines="15"/>
    <tailfilter lines="5"/>
  </filterchain>
</loadfile>
```

This stores the last 10 lines, skipping the last 2 lines, of the supplied data in the property src.file.head. (Means: if supplied data contains 60 lines, lines 49-58 are extracted)

```
<loadfile srcfile="${src.file}" property="src.file.head">
  <filterchain>
    <tailfilter lines="10" skip="2"/>
  </filterchain>
</loadfile>
```

DeleteCharacters

This filter deletes specified characters.

since Ant 1.6

This filter is only available in the convenience form.

Parameter Name	Parameter Value	Required
chars	The characters to delete. This attribute is backslash enabled.	Yes

Examples:

Delete tabs and returns from the data.

```
<deletecharacters chars="\t\r"/>
```

ConcatFilter

This filter prepends or appends the content file to the filtered files.

since Ant 1.6

Parameter Name	Parameter Value	Required
prepend	The name of the file which content should be prepended to the file.	No
append	The name of the file which content should be appended to the file.	No

Examples:

Do nothing:

```
<filterchain>
    <concatfilter/>
</filterchain>
```

Adds a license text before each java source:

```
<filterchain>
    <concatfilter prepend="apache-license-java.txt"/>
</filterchain>
```

TokenFilter

This filter tokenizes the inputstream into strings and passes these strings to filters of strings. Unlike the other filterreaders, this does not support params, only convenience methods are implemented. The tokenizer and the string filters are defined by nested elements.

since Ant 1.6

Only one tokenizer element may be used, the LineTokenizer is the default if none are specified. A tokenizer splits the input into token strings and trailing delimiter strings.

There may be zero or more string filters. A string filter processes a token and either returns a string or a null. It the string is not null it is passed to the next filter. This proceeds until all the filters are called. If a string is returned after all the filters, the string is outputs with its associated token delimitier (if one is present). The trailing delimiter may be overridden by the *delimOutput* attribute.

blackslash interpretation A number of attributes (including *delimOutput*) interpret backslash escapes. The following are understood: \n, \r, \f, \t and \\.

Attribute	Description	Required
delimOutput	This overrides the tokendelimiter returned by the tokenizer if it is not empty. This attribute is backslash enabled.	No

The following tokenizers are provided by the default distribution.

LineTokenizer
FileTokenizer
StringTokenizer

The following string filters are provided by the default distribution.

ReplaceString
ContainsString
ReplaceRegex
ContainsRegex
Trim
IgnoreBlank
DeleteCharacters
UniqFilter

The following string filters are provided by the optional distribution.

ScriptFilter

Some of the filters may be used directly within a filter chain. In this case a tokenfilter is created implicitly. An extra attribute "byline" is added to the filter to specify whether to use a linetokenizer (byline="true") or a filetokenizer (byline="false"). The default is "true".

LineTokenizer

This tokenizer splits the input into lines. The tokenizer delimits lines by "\r", "\n" or "\r\n". This is the default tokenizer.

Attribute	Description	Required
includeDelims	Include the line endings in the token. Default is false.	No

Examples:

Convert input current line endings to unix style line endings.

```
<tokenfilter delimoutput="\n"/>
```

Remove blank lines.

```
<tokenfilter>
    <ignoreblank/>
</tokenfilter>
```

FileTokenizer

This tokenizer treats **all** the input as a token. So be careful not to use this on very large input.

Examples:

Replace the first occurrence of package with //package.

```
<tokenfilter>
      <filetokenizer/>
      <replaceregex pattern="([\n\r]+[ \t]*|^[ \t]*)package"
                    flags="s"
                    replace="\1//package"/>
</tokenfilter>
```

StringTokenizer

This tokenizer is based on java.util.StringTokenizer. It splits up the input into strings separated by white space, or by a

specified list of delimiting characters. If the stream starts with delimiter characters, the first token will be the empty string (unless the *delimsaretokens* attribute is used).

Attribute	Description	Required
delims	The delimiter characters. White space is used if this is not set. (White space is defined in this case by java.lang.Character.isWhitespace()).	No
delimsaretokens	If this is true, each delimiter character is returned as a token. Default is false.	No
suppressdelims	If this is true, delimiters are not returned. Default is false.	No
includeDelims	Include the delimiters in the token. Default is false.	No

Examples:

Surround each non space token with a "[]".

```
<tokenfilter>
    <stringtokenizer/>
    <replaceregex pattern="(.+)" replace="[\1]"/>
</tokenfilter>
```

ReplaceString

This is a simple filter to replace strings. This filter may be used directly within a filterchain.

Attribute	Description	Required
from	The string that must be replaced.	Yes
to	The new value for the replaced string. When omitted an empty string is used.	No

Examples:

Replace "sun" with "moon".

```
<tokenfilter>
    <replacestring from="sun" to="moon"/>
</tokenfilter>
```

ContainsString

This is a simple filter to filter tokens that contains a specified string.

Attribute	Description	Required
contains	The string that the token must contain.	Yes

Examples:

Include only lines that contain "foo";

```
<tokenfilter>
    <containsstring contains="foo"/>
</tokenfilter>
```

ReplaceRegex

This string filter replaces regular expressions. This filter may be used directly within a filterchain.

See Regexp Type concerning the choice of the implementation.

Attribute	Description	Required
pattern	The regular expression pattern to match in the token.	Yes
replace	The substitution pattern to replace the matched regular expression. When omitted an empty string is used.	No
flags	See ReplaceRegexp for an explanation of regex flags.	No

Examples:

Replace all occurrences of "hello" with "world", ignoring case.

```
<tokenfilter>
    <replaceregex pattern="hello" replace="world" flags="gi"/>
</tokenfilter>
```

ContainsRegex

This filters strings that match regular expressions. The filter may optionally replace the matched regular expression. This filter may be used directly within a filterchain.

See Regexp Type concerning the choice of regular expression implementation.

Attribute	Description	Required
pattern	The regular expression pattern to match in the token.	Yes
replace	The substitution pattern to replace the matched regular expression. When omitted the orignal token is returned.	No
flags	See ReplaceRegexp for an explanation of regex flags.	No

Examples:

Filter lines that contain "hello" or "world", ignoring case.

```
<tokenfilter>
    <containsregex pattern="(hello|world)" flags="i"/>
</tokenfilter>
```

This example replaces lines like "SUITE(TestSuite, bits);" with "void register_bits();" and removes other lines.

```
<tokenfilter>
    <containsregex
        pattern="^ *SUITE\(.*,\s*(.*)\s*\).*"
        replace="void register_\1();"/>
</tokenfilter>
```

Trim

This filter trims whitespace from the start and end of tokens. This filter may be used directly within a filterchain.

IgnoreBlank

This filter removes empty tokens. This filter may be used directly within a filterchain.

DeleteCharacters

This filter deletes specified characters from tokens.

Attribute	Description	Required
chars	The characters to delete. This attribute is backslash enabled.	Yes

Examples:

Delete tabs from lines, trim the lines and removes empty lines.

```
<tokenfilter>
    <deletecharacters chars="\t"/>
    <trim/>
    <ignoreblank/>
</tokenfilter>
```

UniqFilter

Suppresses all tokens that match their ancestor token. It is most useful if combined with a sort filter.

This filter may be used directly within a filterchain.

Example:

This suppresses duplicate lines.

```
<tokenfilter>
  <uniqfilter/>
</tokenfilter>
```

ScriptFilter

This is an optional filter that executes a script in a Apache BSF or JSR 223 supported language.

See the Script task for an explanation of scripts and dependencies.

The script is provided with an object *self* that has getToken() and setToken(String) methods. The getToken() method returns the current token. The setToken(String) method replaces the current token.

This filter may be used directly within a filterchain.

Attribute	Description	Required
language	The programming language the script is written in. Must be a supported Apache BSF or JSR 223 language	Yes
manager	The script engine manager to use. See the script task for using this attribute.	No - default is "auto"
src	The location of the script as a file, if not inline	No
setbeans	whether to have all properties, references and targets as global variables in the script. *since Ant 1.8.0*	No, default is "true".
classpath	The classpath to pass into the script.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No

This filter can take a nested <classpath> element. See the script task on how to use this element.

Examples:

Convert to uppercase:

```
<tokenfilter>
    <scriptfilter language="javascript">
        self.setToken(self.getToken().toUpperCase());
    </scriptfilter>
</tokenfilter>
```

Remove lines containing the string "bad" while copying text files:

```
<copy todir="dist">
  <fileset dir="src" includes="**/*.txt"/>
  <filterchain>
    <scriptfilter language="beanshell">
        if (self.getToken().indexOf("bad") != -1) {
            self.setToken(null);
        }
    </scriptfilter>
  </filterchain>
</copy>
```

Custom tokenizers and string filters

Custom string filters and tokenizers may be plugged in by extending the interfaces org.apache.tools.ant.filters.TokenFilter.Filter and org.apache.tools.ant.util.Tokenizer respectly. They are defined the build file using `<typedef/>`. For example a string filter that capitalizes words may be declared as:

```
package my.customant;
import org.apache.tools.ant.filters.TokenFilter;

public class Capitalize
    implements TokenFilter.Filter
{
    public String filter(String token) {
        if (token.length() == 0)
            return token;
        return token.substring(0, 1).toUpperCase() +
                token.substring(1);
   }
}
```

This may be used as follows:

```
  <typedef name="capitalize" classname="my.customant.Capitalize"
           classpath="my.customant.path"/>
  <copy file="input" tofile="output">
    <filterchain>
      <tokenfilter>
        <stringtokenizer/>
        <capitalize/>
      </tokenfilter>
    </filterchain>
  </copy>
```

SortFilter

since Ant 1.8.0

The sort filter reads all lines and sorts them. The sort order can be reversed and it is possible to specify a custom implementation of the `java.util.Comparator` interface to get even more control.

Parameter Name	Parameter Value	Required
reverse	whether to reverse the sort order, defaults to false. **Note:** this parameter is ignored if the comparator parameter is present as well.	No

comparator	Class name of a class that implements `java.util.Comparator` for Strings. This class will be used to determine the sort order of lines.	No

This filter is also available using the name `sortfilter`. The `reverse` parameter becomes an attribute, `comparator` can be specified by using a nested element.

Examples:

```
<copy todir="build">
    <fileset dir="input" includes="*.txt"/>
    <filterchain>
        <sortfilter/>
    </filterchain>
</copy>
```

Sort all files `*.txt` from *src* location into *build* location. The lines of each file are sorted in ascendant order comparing the lines via the `String.compareTo(Object o)` method.

```
<copy todir="build">
    <fileset dir="input" includes="*.txt"/>
    <filterchain>
        <sortfilter reverse="true"/>
    </filterchain>
</copy>
```

Sort all files `*.txt` from *src* location into reverse order and copy them into *build* location.

```
<copy todir="build">
    <fileset dir="input" includes="*.txt"/>
    <filterchain>
        <filterreader classname="org.apache.tools.ant.filters.SortFilter">
          <param name="comparator" value="org.apache.tools.ant.filters.EvenFirstCmp"/>
        </filterreader>
    </filterchain>
</copy>
```

Sort all files `*.txt` from *src* location using as sorting criterium `EvenFirstCmp` class, that sorts the file lines putting even lines first then odd lines for example. The modified files are copied into *build* location. The `EventFirstCmp`, has to an instanciable class via `Class.newInstance()`, therefore in case of inner class has to be *static*. It also has to implement `java.util.Comparator` interface, for example:

```
 package org.apache.tools.ant.filters;
 ...(omitted)
   public final class EvenFirstCmp implements <b>Comparator</b> {
     public int compare(Object o1, Object o2) {
     ...(omitted)
     }
   }
```

The example above is equivalent to:

```
<componentdef name="evenfirst"
              classname="org.apache.tools.ant.filters.EvenFirstCmp"/>
<copy todir="build">
    <fileset dir="input" includes="*.txt"/>
    <filterchain>
        <sortfilter>
            <evenfirst/>
        </sortfilter>
    </filterchain>
</copy>
```

FilterSet

FilterSets are groups of filters. Filters can be defined as token-value pairs or be read in from a file. FilterSets can appear inside tasks that support this feature or at the same level as `<target>` - i.e., as children of `<project>`.

FilterSets support the `id` and `refid` attributes. You can define a FilterSet with an `id` attribute and then refer to that definition from another FilterSet with a `refid` attribute. It is also possible to nest filtersets into filtersets to get a set union of the contained filters.

In addition, FilterSets can specify `begintoken` and/or `endtoken` attributes to define what to match.

Filtersets are used for doing replacements in tasks such as `<copy>`, etc.

If you specify multiple values for the same token, the last one defined within a filterset will be used.

Note: When a filterset is used in an operation, the files are processed in text mode and the filters applied line by line. This means that the copy operations will typically corrupt binary files. When applying filters you should ensure that the set of files being filtered are all text files.

Filterset

Attribute	Description	Default	Required
begintoken	The string marking the beginning of a token (eg., `@DATE@`).	@	No
endtoken	The string marking the end of a token (eg., `@DATE@`).	@	No
filtersfile	Specify a single filtersfile.	*none*	No
recurse	Indicates whether the replacement text of tokens should be searched for more tokens. **Since Ant 1.6.3**	*true*	No
onmissingfiltersfile	Indicate behavior when a nonexistent *filtersfile* is specified. One of "fail", "warn", "ignore". **Since Ant 1.7**	"fail"	No

Filter

Attribute	Description	Required
token	The token to replace (eg., `@DATE@`)	Yes
value	The value to replace it with (eg., `Thursday, April 26, 2001`).	Yes

Filtersfile

Attribute	Description	Required
file	A properties file of name-value pairs from which to load the tokens.	Yes

Examples

You are copying the `version.txt` file to the `dist` directory from the `build` directory but wish to replace the token `@DATE@` with today's date.

```
<copy file="${build.dir}/version.txt" toFile="${dist.dir}/version.txt">
  <filterset>
    <filter token="DATE" value="${TODAY}"/>
  </filterset>
</copy>
```

You are copying the `version.txt` file to the `dist` directory from the build directory but wish to replace the token `%DATE*` with today's date.

```
<copy file="${build.dir}/version.txt" toFile="${dist.dir}/version.txt">
  <filterset begintoken="%" endtoken="*">
    <filter token="DATE" value="${TODAY}"/>
  </filterset>
</copy>
```

Copy all the docs but change all dates and appropriate notices as stored in a file.

```
<copy toDir="${dist.dir}/docs">
  <fileset dir="${build.dir}/docs">
    <include name="**/*.html">
  </fileset>
  <filterset begintoken="%" endtoken="*">
    <filtersfile file="${user.dir}/dist.properties"/>
  </filterset>
</copy>
```

Define a FilterSet and reference it later.

```
<filterset id="myFilterSet" begintoken="%" endtoken="*">
  <filter token="DATE" value="${TODAY}"/>
</filterset>

<copy file="${build.dir}/version.txt" toFile="${dist.dir}/version.txt">
  <filterset refid="myFilterSet"/>
</copy>
```

PatternSet

Patterns can be grouped to sets and later be referenced by their `id` attribute. They are defined via a `patternset` element, which can appear nested into a FileSet or a directory-based task that constitutes an implicit FileSet. In addition, `patternsets` can be defined as a stand alone element at the same level as `target` — i.e., as children of `project` as well as as children of `target`.

Patterns can be specified by nested `<include>`, or `<exclude>` elements or the following attributes.

Attribute	Description
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.
includesfile	the name of a file; each line of this file is taken to be an include pattern. You can specify more than one include file by using a nested includesfile elements.
excludes	comma- or space-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.
excludesfile	the name of a file; each line of this file is taken to be an exclude pattern. You can specify more than one exclude file by using a nested excludesfile elements.

Parameters specified as nested elements

`include` and `exclude`

Each such element defines a single pattern for files to include or exclude.

Attribute	Description	Required
name	the pattern to in/exclude.	Yes
if	Only use this pattern if the named property is set.	No
unless	Only use this pattern if the named property is **not** set.	No

`includesfile` and `excludesfile`

If you want to list the files to include or exclude external to your build file, you should use the includesfile/excludesfile attributes or elements. Using the attribute, you can only specify a single file of each type, while the nested elements can be specified more than once - the nested elements also support if/unless attributes you can use to test the existance of a property.

Attribute	Description	Required
name	the name of the file holding the patterns to in/exclude.	Yes
if	Only read this file if the named property is set.	No
unless	Only read this file if the named property is **not** set.	No

`patternset`

Patternsets may be nested within one another, adding the nested patterns to the parent patternset.

invert

A nested patternset can be inverted using the `<invert>` element. *Since Ant 1.7.1*

Examples

```
<patternset id="non.test.sources">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</patternset>
```

Builds a set of patterns that matches all `.java` files that do not contain the text `Test` in their name. This set can be referred to via `<patternset refid="non.test.sources"/>`, by tasks that support this feature, or by FileSets.

Note that while the `includes` and `excludes` attributes accept multiple elements separated by commas or spaces, the nested `<include>` and `<exclude>` elements expect their name attribute to hold a single pattern.

The nested elements allow you to use if and unless arguments to specify that the element should only be used if a property is set, or that it should be used only if a property is not set.

For example

```
<patternset id="sources">
  <include name="std/**/*.java"/>
  <include name="prof/**/*.java" if="professional"/>
  <exclude name="**/*Test*"/>
</patternset>
```

will only include the files in the sub-directory *prof* if the property *professional* is set to some value.

The two sets

```
<patternset includesfile="some-file"/>
```

and

```
<patternset>
  <includesfile name="some-file"/>
<patternset/>
```

are identical. The include patterns will be read from the file `some-file`, one pattern per line.

```
<patternset>
  <includesfile name="some-file"/>
  <includesfile name="${some-other-file}"
                if="some-other-file"
  />
<patternset/>
```

will also read include patterns from the file the property `some-other-file` points to, if a property of that name has been defined.

Permissions

Permissions represents a set of security permissions granted or revoked to a specific part code executed in the JVM where ant is running in. The actual Permissions are specified via a set of nested permission items either `<grant>`ed or `<revoke>`d.

In the base situation a base set of permissions granted. Extra permissions can be granted. A granted permission can be overruled by revoking a permission. The security manager installed by the permissions will throw an `SecurityException` if the code subject to these permissions try to use an permission that has not been granted or that has been revoked.

Nested elements

grant

Indicates a specific permission is always granted. Its attributes indicate which permissions are granted.

Attribute	Description	Required
class	The fully qualified name of the Permission class.	Yes
name	The name of the Permission. The actual contents depends on the Permission class.	No
actions	The actions allowed. The actual contents depend on the Permission class and name.	No

Implied permissions are granted.

Please note that some Permission classes may actually need a name and / or actions in order to function properly. The name and actions are parsed by the actual Permission class.

revoke

Indicates a specific permission is revoked.

Attribute	Description	Required
class	The fully qualified name of the Permission class.	Yes
name	The name of the Permission. The actual contents depends on the Permission class.	No
actions	The actions allowed. The actual contents depend on the Permission class and name.	No

Implied permissions are not resolved and therefore also not revoked.

The name can handle the * wildcard at the end of the name, in which case all permissions of the specified class of which the name starts with the specified name (excluding the *) are revoked. Note that the - wildcard often supported by the granted properties is not supported. If the name is left empty all names match, and are revoked. If the actions are left empty all actions match, and are revoked.

Base set

A permissions set implictly contains the following permissions:

```
<grant class="java.net.SocketPermission" name="localhost:1024-" actions="listen">
```

```
<grant class="java.util.PropertyPermission" name="java.version" actions="read">
<grant class="java.util.PropertyPermission" name="java.vendor" actions="read">
<grant class="java.util.PropertyPermission" name="java.vendor.url" actions="read">
<grant class="java.util.PropertyPermission" name="java.class.version" actions="read">
<grant class="java.util.PropertyPermission" name="os.name" actions="read">
<grant class="java.util.PropertyPermission" name="os.version" actions="read">
<grant class="java.util.PropertyPermission" name="os.arch" actions="read">
<grant class="java.util.PropertyPermission" name="file.encoding" actions="read">
<grant class="java.util.PropertyPermission" name="file.separator" actions="read">
<grant class="java.util.PropertyPermission" name="path.separator" actions="read">
<grant class="java.util.PropertyPermission" name="line.separator" actions="read">
<grant class="java.util.PropertyPermission" name="java.specification.version"
actions="read">
<grant class="java.util.PropertyPermission" name="java.specification.vendor"
actions="read">
<grant class="java.util.PropertyPermission" name="java.specification.name"
actions="read">
<grant class="java.util.PropertyPermission" name="java.vm.specification.version"
actions="read">
<grant class="java.util.PropertyPermission" name="java.vm.specification.vendor"
actions="read">
<grant class="java.util.PropertyPermission" name="java.vm.specification.name"
actions="read">
<grant class="java.util.PropertyPermission" name="java.vm.version" actions="read">
<grant class="java.util.PropertyPermission" name="java.vm.vendor" actions="read">
<grant class="java.util.PropertyPermission" name="java.vm.name" actions="read">
```

These permissions can be revoked via `<revoke>` elements if necessary.

Examples

```
<permissions>
  <grant class="java.security.AllPermission"/>
  <revoke class="java.util.PropertyPermission"/>
</permissions>
```

Grants all permissions to the code except for those handling Properties.

```
<permissions>
  <grant class="java.net.SocketPermission" name="foo.bar.com" action="connect"/>
  <grant class="java.util.PropertyPermission" name="user.home" action="read,write"/>
</permissions>
```

Grants the base set of permissions with the addition of a SocketPermission to connect to foo.bar.com and the permission to read and write the user.home system property.

PropertySet

Since Ant 1.6

Groups a set of properties to be used by reference in a task that supports this.

Attribute	Description	Required
dynamic	Whether to reevaluate the set everytime the set is used. Default is "`true`".	No
negate	Whether to negate results. If "`true`", all properties *not* selected by nested elements will be returned. Default is "`false`". *Since Ant 1.6.2*	No

Parameters specified as nested elements

propertyref

Selects properties from the current project to be included in the set.

Attribute	Description	Required
name	Select the property with the given name.	Exactly one of these.
prefix	Select the properties whose name starts with the given string.	
regex	Select the properties that match the given regular expression. Similar to regexp type mappers, this requires a supported regular expression library.	
builtin	Selects a builtin set of properties. Valid values for this attribute are `all` for all Ant properties, `system` for the system properties and `commandline` for all properties specified on the command line when invoking Ant (plus a number of special internal Ant properties).	

propertyset

A `propertyset` can be used as the set union of more `propertysets`.

For example:

```
<propertyset id="properties-starting-with-foo">
  <propertyref prefix="foo"/>
</propertyset>
<propertyset id="properties-starting-with-bar">
  <propertyref prefix="bar"/>
</propertyset>
<propertyset id="my-set">
  <propertyset refid="properties-starting-with-foo"/>
  <propertyset refid="properties-starting-with-bar"/>
</propertyset>
```

collects all properties whose name starts with either "foo" or "bar" in the set named "my-set".

mapper

A mapper - at maximum one mapper can be specified. The mapper is used to change the names of the property keys, for example:

```
<propertyset id="properties-starting-with-foo">
  <propertyref prefix="foo"/>
```

```
  <mapper type="glob" from="foo*" to="bar*"/>
</propertyset>
```

collects all properties whose name starts with "foo", but changes the names to start with "bar" instead.

If supplied, the nested mapper will be applied subsequent to any negation of matched properties.

I/O redirection

For many tasks, input and output can be defined in a fairly straightforward fashion. The exec task, used to execute an external process, stands as a very basic example. The executed process may accept input, produce output, or do either or both depending upon various circumstances. Output may be classified as "output" or as "error output." The `<redirector>` type provides a concrete means of redirecting input and output featuring the use of File Mappers to specify source (input) and destination (output/error) files. *Since Ant 1.6.2*

The `<redirector>` element accepts the following attributes:

Attribute	Description	Required
output	Name of a file to which output should be written. If the error stream is not also redirected to a file or property, it will appear in this output.	No
error	The file to which the standard error of the command should be redirected.	No
logError	This attribute is used when you wish to see error output in Ant's log and you are redirecting output to a file/property. The error output will not be included in the output file/property. If you redirect error with the *error* or *errorProperty* attributes, this will have no effect.	No
append	Whether output and error files should be appended to rather than overwritten. Defaults to `false`.	No
createemptyfiles	Whether output and error files should be created even when empty. Defaults to `true`.	No
outputproperty	The name of a property in which the output of the command should be stored. Unless the error stream is redirected to a separate file or stream, this property will include the error output.	No
errorproperty	The name of a property in which the standard error of the command should be stored.	No
input	A file from which the executed command's standard input is taken. This attribute is mutually exclusive with the *inputstring* attribute.	No
inputstring	A string which serves as the input stream for the executed command. This attribute is mutually exclusive with the *input* attribute.	No
inputencoding	The input encoding.	No
outputencoding	The output encoding.	No
errorencoding	The error encoding.	No
alwayslog	Always send to the log in addition to any other destination. *Since Ant 1.6.3*.	No, default is `false`
loginputstring	Controls the display of *inputstring*'s value in log messages. Set to `false` when sending sensitive data (e.g. passwords) to external processes. *Since Ant 1.6.3*.	No, default is `true`

Parameters specified as nested elements

inputmapper

A single File Mapper used to redirect process input. Multiple mapping results should concatenate all mapped files as input. Mapping will ordinarily be performed on a task-specified sourcefile; consult the documentation of the individual task for more details. A nested `<inputmapper>` is not compatible with either of the *input* or *inputstring* attributes.

outputmapper

A single File Mapper used to redirect process output. Mapping will ordinarily be performed on a task-specified sourcefile; consult the documentation of the individual task for more details. A nested `<outputmapper>` is not compatible with the *output* attribute.

errormapper

A single File Mapper used to redirect error output. Mapping will ordinarily be performed on a task-specified sourcefile; consult the documentation of the individual task for more details. A nested `<errormapper>` is not compatible with the *error* attribute.

inputfilterchain

A FilterChain can be applied to the process input.

outputfilterchain

A FilterChain can be applied to the process output.

errorfilterchain

A FilterChain can be applied to the error output.

Usage

Tasks known to support I/O redirection:

- Exec
- Apply
- Java

The expected behavior of a `<redirector>` is to a great degree dependent on the supporting task. Any possible points of confusion should be noted at the task level.

Regexp

Regexp represents a regular expression.

Parameters

Attribute	Description	Required
pattern	regular expression pattern	Yes

Examples

```
    <regexp id="myregexp" pattern="alpha(.+)beta"/>
```

Defines a regular expression for later use with id myregexp.

```
    <regexp refid="myregexp"/>
```

Use the regular expression with id myregexp.

Choice of regular expression implementation

Ant comes with wrappers for the java.util.regex package of JDK 1.4, jakarta-regexp and jakarta-ORO, See installation dependencies concerning the supporting libraries.

The property `ant.regexp.regexpimpl` governs which regular expression implementation will be chosen. Possible values for this property are :

- org.apache.tools.ant.util.regexp.Jdk14RegexpRegexp
- org.apache.tools.ant.util.regexp.JakartaOroRegexp
- org.apache.tools.ant.util.regexp.JakartaRegexpRegexp

It can also be another implementation of the interface `org.apache.tools.ant.util.regexp.Regexp`. If `ant.regexp.regexpimpl` is not defined, Ant uses Jdk14Regexp as this is always available.

There are cross-platform issues for matches related to line terminator. For example if you use $ to anchor your regular expression on the end of a line the results might be very different depending on both your platform and the regular expression library you use. It is 'highly recommended' that you test your pattern on both Unix and Windows platforms before you rely on it.

- Jakarta Oro defines a line terminator as '\n' and is consistent with Perl.
- Jakarta RegExp uses a system-dependent line terminator.
- JDK 1.4 uses '\n', '\r\n', '\u0085', '\u2028', '\u2029' as a default but is configured in the wrapper to use only '\n' (UNIX_LINE)

*We **strongly** recommend that you use Jakarta Oro.*

Usage

The following tasks and types use the Regexp type :

- ReplaceRegExp task
- LineContainsRegexp filter

These string filters also use the mechanism of regexp to choose a regular expression implementation :

- ContainsRegex string filter
- ReplaceRegex string filter
- filename selector
- name resource selector
- containsregexp selector

Resources

A file-like entity can be abstracted to the concept of a *resource*. In addition to providing access to file-like attributes, a resource implementation should, when possible, provide the means to read content from and/or write content to the underlying entity. Although the resource concept was introduced in *Ant 1.5.2*, resources are available for explicit use beginning in **Ant 1.7**.

The built-in resource types are:

- resource - a basic resource.
- bzip2resource - a BZip2 compressed resource.
- file - a file.
- gzipresource - a GZip compressed resource.
- javaresource - a resource loadable via a Java classloader.
- propertyresource - an Ant property.
- string - a text string.
- tarentry - an entry in a tar file.
- url - a URL.
- zipentry - an entry in a zip file.

resource

A basic resource. Other resource types derive from this basic type; as such all its attributes are available, though in most cases irrelevant attributes will be ignored. This and all resource implementations are also usable as single-element Resource Collections.

Attribute	Description	Required
name	The name of this resource	No
exists	Whether this resource exists	No, default *true*
lastmodified	The last modification time of this resource	No
directory	Whether this resource is directory-like	No, default *false*
size	The size of this resource	No

file

Represents a file accessible via local filesystem conventions.

Attribute	Description	Required
file	The file represented by this resource	Yes
basedir	The base directory of this resource. When this attribute is set, attempts to access the name of the resource will yield a path relative to this location.	No

javaresource

Represents a resource loadable via a Java classloader.

Attribute	Description	Required

name	The name of the resource.	Yes
classpath	the classpath to use when looking up a resource.	No
classpathref	the classpath to use when looking up a resource, given as reference to a `<path>` defined elsewhere..	No
loaderRef	the name of the loader that is used to load the resource, constructed from the specified classpath.	No
parentFirst	Whether to consult the parent classloader first - the parent classloader most likely is the system classloader - when using a nested classpath. Defaults to `true`. *Since Ant 1.8.0*	No

The classpath can also be specified as nested classpath element, where **<classpath>** is a path-like structure.

zipentry

Represents an entry in a ZIP archive. The archive can be specified using the archive attribute or a nested single-element resource collection. `zipentry` only supports file system resources as nested elements.

Attribute	Description	Required
zipfile or its alias name archive	The zip file containing this resource	Yes, unless a nested resource collection has been specified.
name	The name of the archived resource	Yes
encoding	The encoding of the zipfile	No; platform default used if unspecified

tarentry

Represents an entry in a TAR archive. The archive can be specified using the archive attribute or a nested single-element resource collection.

Attribute	Description	Required
archive	The tar archive containing this resource	Yes, unless a nested resource collection has been specified.
name	The name of the archived resource	Yes

gzipresource

This is not a stand-alone resource, but a wrapper around another resource providing compression of the resource's contents on the fly. A single element resource collection must be specified as a nested element.

bzip2resource

This is not a stand-alone resource, but a wrapper around another resource providing compression of the resource's contents on the fly. A single element resource collection must be specified as a nested element.

url

Represents a URL.

Attribute	Description	Required

url	The url to expose	Exactly one of these
file	The file to expose as a file: url	
baseUrl	The base URL which must be combined with relativePath	
relativePath	Relative path that defines the url combined with baseUrl	If using baseUrl

string

Represents a Java String. It can be written to, but only once, after which it will be an error to write to again.

Attribute	Description	Required
value	The value of this resource	No

The resource also supports nested text, which can only be supplied if the `value` attribute is unset:

```
<string>
      self.log("Ant version =${ant.version}");
</string>
```

propertyresource

Represents an Ant property.

Attribute	Description	Required
name	The property name	Yes

Resource Collections

A Resource Collection is an abstraction of an entity that groups together a number of resources. Several of Ant's "legacy" datatypes have been modified to behave as Resource Collections:

- fileset, dirset, filelist, and path (and derivative types) expose file resources
- tarfileset can expose file or tarentry resources depending on configuration
- zipfileset can expose file or zipentry resources depending on configuration
- propertyset exposes property resources

Strangely, some tasks can even legitimately behave as resource collections:

- concat exposes a concatenated resource, and adds e.g. filtering to Ant's resource-related capabilities.

The additional built-in resource collections are:

- resources - generic resource collection
- files - collection of files similar to fileset
- restrict - restrict a resource collection to include only resources meeting specified criteria
- sort - sorted resource collection
- first - first *n* resources from a nested collection
- last - last *n* resources from a nested collection
- tokens - string tokens gathered from a nested collection
- union - set union of nested resource collections

- intersect - set intersection of nested resource collections
- difference - set difference of nested resource collections
- mappedresources - generic resource collection wrapper that maps the names of the nested resources using a mapper.
- archives - wraps around different resource collections and treats the nested resources as ZIP or TAR archives that will be extracted on the fly.
- resourcelist - a collection of resources who's names have been read from another resource.

resources

A generic resource collection, designed for use with references. For example, if a third-party Ant task generates a Resource Collection of an unknown type, it can still be accessed via a `<resources>` collection. The secondary use of this collection type is as a container of other resource collections, preserving the order of nested collections as well as duplicate resources (contrast with union).

Attribute	Description	Required
cache	Whether to cache results. *since Ant 1.8.0*	No, default *false*

files

A group of files. These files are matched by **absolute** patterns taken from a number of PatternSets. These can be specified as nested `<patternset>` elements. In addition, `<files>` holds an implicit PatternSet and supports the nested `<include>`, `<includesfile>`, `<exclude>` and `<excludesfile>` elements of PatternSet directly, as well as PatternSet's attributes.

File Selectors are available as nested elements. A file must be selected by all selectors in order to be included; `<files>` is thus equivalent to an `<and>` file selector container.

More simply put, this type is equivalent to a fileset with no base directory. **Please note** that without a base directory, filesystem scanning is based entirely on include and exclude patterns. A filename (or any) selector can *only* influence the scanning process *after* the file has been included based on pattern-based selection.

Attribute	Description	Required
includes	comma- or space-separated list of patterns of files that must be included	At least one of these
includesfile	the name of a file; each line of this file is taken to be an include pattern.	
excludes	comma- or space-separated list of patterns of files that must be excluded	No, default none (except default excludes when true)
excludesfile	the name of a file; each line of this file is taken to be an exclude pattern.	
defaultexcludes	Whether default excludes should be used	No, default *true*
casesensitive	Whether patterns are case-sensitive	No, default *true*
followsymlinks	Whether to follow symbolic links (see note below)	No, default *true*

Note: All files/directories for which the canonical path is different from its path are considered symbolic links. On Unix systems this usually means the file really is a symbolic link but it may lead to false results on other platforms.

restrict

Restricts a nested resource collection using resource selectors:

Attribute	Description	Required
cache	Whether to cache results; disabling may seriously impact performance	No, default *true*

Parameters specified as nested elements

A single resource collection is required.

Nested resource selectors are used to "narrow down" the included resources. These are patterned after file selectors but are, unsurprisingly, targeted to resources. Several built-in resource selectors are available in the internal antlib `org.apache.tools.ant.types.resources.selectors`:

- name - select resources by name.
- exists - select existing resources.
- date - select resources by date.
- type - select resources by type.
- size - select resources by size.
- instanceof - select resources by class or Ant datatype.
- and - "and" nested resource selectors.
- or - "or" nested resource selectors.
- not - "not" a nested resource selector.
- none - select resources selected by no nested resource selectors.
- majority - select resources selected by a majority of nested resource selectors.
- modified - select resources which content has changed.
- contains - select resources containing a particular text string.
- containsregexp - select resources whose contents match a particular regular expression.
- compare - select resources based on comparison to other resources.
- readable - Select files (resources must be files) if they are readable.
- writable - Select files (resources must be files) if they are writable.

name

Selects resources by name.

Attribute	Description	Required
name	The name pattern to test using standard Ant patterns.	Exactly one of the two
regex	The regular expression matching files to select.	
casesensitive	Whether name comparisons are case-sensitive	No, default *true*
handledirsep	If this is specified, the mapper will treat a \ character in a resource name or name attribute as a / for the purposes of matching. This attribute can be true or false, the default is false. *Since Ant 1.8.0.*	No

exists

Selects existing resources.

date

Selects resources by date.

Attribute	Description	Required
millis	The comparison date/time in ms since January 1, 1970	One of these
datetime	The formatted comparison date/time	
pattern	SimpleDateFormat-compatible pattern for use with the `datetime` attribute	No, default is "MM/DD/YYYY HH:MM AM_or_PM"
granularity	The number of milliseconds leeway to use when comparing file modification times. This is needed because not every file system supports tracking the last modified time to the millisecond level.	No; default varies by platform: FAT filesystems = 2 sec; Unix = 1 sec; NTFS = 1 ms.
when	One of "before", "after", "equal"	No, default "equal"

type

Selects resources by type (file or directory).

Attribute	Description	Required
type	One of "file", "dir", "any" (since Ant 1.8)	Yes

size

Selects resources by size.

Attribute	Description	Required
size	The size to compare	Yes
when	One of "equal", "eq", "greater", "gt", "less", "lt", "ge" (greater or equal), "ne" (not equal), "le" (less or equal)	No, default "equal"

instanceof

Selects resources by type.

Attribute	Description	Required
class	The class of which the resource must be an instance	One of these
type	The Ant type that must be assignable from the resource	
uri	The uri in which *type* must be defined	No

and

Selects a resource if it is selected by all nested resource selectors.

or

Selects a resource if it is selected by at least one nested resource selector.

not

Negates the selection result of the single nested resource selector allowed.

none

Selects a resource if it is selected by no nested resource selectors.

majority

Selects a resource if it is selected by the majority of nested resource selectors.

Attribute	Description	Required
allowtie	Whether a tie (when there is an even number of nested resource selectors) is considered a majority	No, default *true*

compare

Selects a resource based on its comparison to one or more "control" resources using nested resource comparators.

Attribute	Description	Required
when	Comparison ("equal"/"eq", "greater"/"gt", "less"/"lt", "le" (less or equal), "ge" (greater or equal), "ne" (not equal).	No, default "equal"
against	Quantifier ("all"/"each"/"every", "any"/"some", (exactly) "one", "most"/"majority", "none".	No, default "all"

Parameters specified as nested elements

The resources against which comparisons will be made must be specified using the nested <control> element, which denotes a resources collection.

Examples

Assuming the namespace settings

```
  rsel="antlib:org.apache.tools.ant.types.resources.selectors"
  rcmp="antlib:org.apache.tools.ant.types.resources.comparators"
```

```
<restrict>
  <fileset dir="src" includes="a,b,c,d,e,f,g" />
  <rsel:compare when="le" against="all">
    <control>
      <resource name="d" />
    </control>
    <rcmp:name />
  </rsel:compare>
</restrict>
```

Selects files a, b, c, and d.

```
    <project rsel="antlib:org.apache.tools.ant.types.resources.selectors">
        <macrodef name="copyFromPath">
            <attribute name="todir"/>
```

```
            <attribute name="refid"/>
            <element name="nested-resource-selectors" optional="yes" implicit="true"/>
            <sequential>
                <mkdir dir="@{todir}" taskname="copyFromPath"/>
                <copy todir="@{todir}" taskname="copyFromPath">
                    <restrict>
                        <path refid="@{refid}"/>
                        <rsel:or>
                            <nested-resource-selectors/>
                        </rsel:or>
                    </restrict>
                    <flattenmapper/>
                </copy>
            </sequential>
        </macrodef>
        <copyFromPath refid="classpath" todir="todir">
            <rsel:name name="log4j.properties"/>
            <rsel:name name="default.properties"/>
        </copyFromPath>
    </project>
```

Creates the `todir` directory and copies (if present) the files `log4j.properties` and `default.properties` from the Classpath (already used while compiling).

```
  <project>
      <filelist id="allfiles" dir="${ant.home}/bin"
files="ant.cmd,foo.txt,ant.bat,bar.txt,ant"/>
      <restrict id="missingfiles">
          <filelist refid="allfiles"/>
          <rsel:not xmlns:rsel="antlib:org.apache.tools.ant.types.resources.selectors">
              <rsel:exists/>
          </rsel:not>
      </restrict>
      <echo>These files are missed: ${toString:missingfiles}</echo>
  </project>
```

The resource collection *allfiles* defines a list of files which are expected. The restrict *missingfiles* uses the `<not><exists>` selector for getting all files which are not present. Finally we use the *toString:* pathshortcut for getting them in a readable form: `[echo] These files are missed: ....foo.txt;....bar.txt`

sort

Sorts a nested resource collection according to the resources' natural order, or by one or more nested resource comparators:

Attribute	Description	Required
cache	Whether to cache results; disabling may seriously impact performance	No, default *true*

Parameters specified as nested elements

A single resource collection is required.

The sort can be controlled and customized by specifying one or more resource comparators. Resources can be sorted according to multiple criteria; the first specified is the "outermost", while the last specified is the "innermost". Several built-in resource comparators are available in the internal antlib `org.apache.tools.ant.types.resources.comparators`:

Resource Comparators:

- name - sort resources by name
- exists - sort resources by existence

- date - sort resources by date
- type - sort resources by type
- size - sort resources by size
- content - sort resources by content
- reverse - reverse the natural sort order, or that of a single nested resource comparator

name

Sort resources by name.

exists

Sort resources by existence. Not existing is considered "less than" existing.

date

Sort resources by date.

type

Sort resources by type (file or directory). Because directories contain files, they are considered "greater".

size

Sort resources by size.

content

Sort resources by content.

Attribute	Description	Required
binary	Whether content should be compared in binary mode. If *false*, *content will be compared without regard to platform-specific line-ending conventions.*	No, default *true*

reverse

Reverse the natural sort order, or that of a single nested comparator.

Examples

```
    <property name="eol" value="${line.separator}" />
    <pathconvert property="sorted" pathsep="${eol}">
      <sort>
        <tokens>
          <string value="foo bar etc baz" />
          <stringtokenizer />
        </tokens>
      </sort>
    </pathconvert>
```

The resource of type string "foo bar etc baz" is split into four tokens by the stringtokenizer. These tokens are sorted and there *sorted* gets the value of "bar baz etc foo".

```
<sort>
  <fileset dir="foo" />
  <reverse xmlns="antlib:org.apache.tools.ant.types.resources.comparators">
    <date />
  </reverse>
</sort>
```

This takes all files from *foo* and sorts them by modification date in reverse order. Because the resource comparators used (`<reverse>` and `<date>`) are in an internal antlib their namespace must be set explicitly.

first

Includes the first *count* resources from a nested resource collection. This can be used in conjunction with the sort collection, for example, to select the first few oldest, largest, etc. resources from a larger collection.

Attribute	Description	Required
count	The number of resources to include	No, default 1
cache	Whether to cache results; disabling may seriously impact performance	No, default *true*

Parameters specified as nested elements

A single resource collection is required.

last

Includes the last *count* resources from a nested resource collection. This can be used in conjunction with the sort collection, for example, to select the last few oldest, largest, etc. resources from a larger collection. **Since Ant 1.7.1**.

Attribute	Description	Required
count	The number of resources to include	No, default 1
cache	Whether to cache results; disabling may seriously impact performance	No, default *true*

Parameters specified as nested elements

A single resource collection is required.

tokens

Includes the string tokens gathered from a nested resource collection. Uses the same tokenizers supported by the TokenFilter. Imaginative use of this resource collection can implement equivalents for such Unix functions as `sort`, `grep -c`, `wc` and `wc -l`.

Attribute	Description	Required
encoding	The encoding of the nested resources	No, default is platform default
cache	Whether to cache results; disabling may seriously impact performance	No, default *true*

Parameters specified as nested elements

- A single resource collection is required.

- One nested tokenizer may be specified. If omitted, a LineTokenizer will be used.

Examples

```
<concat>
  <union>
    <sort>
      <tokens>
        <resources refid="input" />
        <linetokenizer includedelims="true" />
      </tokens>
    </sort>
  </union>
</concat>
```

Implements Unix *sort -u* against resource collection *input*.

Set operations

The following resource collections implement set operations:

- union
- intersect
- difference

union

Union of nested resource collections.

intersect

Intersection of nested resource collections.

difference

Difference of nested resource collections.

The following attributes apply to all set-operation resource collections:

Attribute	Description	Required
cache	Whether to cache results; disabling may seriously impact performance	No, default *true*

Examples

```
    <resources id="A">
        <string value="a"/>
        <string value="b"/>
    </resources>
    <resources id="B">
        <string value="b"/>
        <string value="c"/>
    </resources>
    <union id="union"><resources refid="A"/><resources refid="B"/></union>
    <intersect id="intersect"><resources refid="A"/><resources refid="B"/></intersect>
    <difference id="difference"><resources refid="A"/><resources
refid="B"/></difference>
    <echo>
      A: ${toString:A}                    = a;b
      B: ${toString:B}                    = b;c
```

```
  union     : ${toString:union}       = a;b;c
  intersect : ${toString:intersect}   = b
  difference: ${toString:difference}  = a;c
</echo>
```

mappedresources

Since Ant 1.8.0

Wraps another resource collection and maps the names of the nested resources using a mapper.

Even if *mappedresources* wraps a resource collection that consists of file-system based resources, *mappedresources* will not appear to be file-system based. This means you can't use *mappedresources* with tasks that only allow file-system based resources.

Parameters specified as attributes

Attribute	Description	Required
cache	Whether to cache results; enabling may improve performance. *Since Ant 1.8.1*	No, default *false*
enablemultiplemappings	If true the the collection will use all the mappings for a given source path. If false the it will only process the first resource. *since Ant 1.8.1*.	No - defaults to false.

Parameters specified as nested elements

A single resource collection is required.

A single mapper can be used to map names. If no mapper has been given (which doesn't make any sense, honestly), an identity mapper will be used.

Examples

Copies all files from a given directory to a target directory adding ".bak" as an extension. Note this could be done with a *mapper* nested into *copy* directly as well.

```
<copy todir="${target}">
  <mapperesources>
    <fileset dir="${src}"/>
    <globmapper from="*" to="*.bak"/>
  </mapperesources>
</copy>
```

Creates a WAR archive adding all CLASSPATH entries that are files to the `WEB-INF/lib` directory without keeping their files-system structure.

```
<war destfile="${output}">
  <mappedresources>
    <restrict>
      <path path="${java.class.path}"/>
      <type type="file"/>
    </restrict>
    <chainedmapper>
      <flattenmapper/>
      <globmapper from="*" to="WEB-INF/lib/*"/>
```

```
        </chainedmapper>
      </mappedresources>
    </war>
```

archives

Since Ant 1.8.0

This resource collection accepts an arbitrary number of nested resources and assumes that all those resources must be either ZIP or TAR archives. The resources returned by `<archives>` are the contents of the nested archives.

This resource collection is a generalization of zipgroupfileset which is only supported by the zip family of tasks.

archives doesn't support any attributes.

Parameters specified as nested elements

`<archives>` has two nested elements `<zips>` and `<tars>` that are unions themselves, i.e. they accept arbitrary many resource(collection)s as nested elements.

The nested resources of <zips> are treated as ZIP archives, the nested resources of <tars> as TAR archives.

Examples

Copies all files from all jars that are on the classpath to `${target}`.

```
    <copy todir="${target}">
      <archives>
        <zips>
          <restrict>
            <path path="${java.class.path}"/>
            <name name="*.jar"/>
          </restrict>
        </zips>
      </archives>
    </copy>
```

resourcelist

Since Ant 1.8.0

This resource collection accepts an arbitrary number of nested resources, reads those resources and returns a resource for each line read.

If the line contains a colon, Ant will try to use it as an URL and if that fails (or the line doesn't contain a colon) will return a file resource with the line's content as its name.

Properties will be expanded for each line. If the property expansion yields a resource object rather than a string (for example because of custom property helpers), the resources will be returned directly.

`<resourcelist>` is a generalization of `<filelist>`.

Attribute	Description	Required
encoding	The encoding of the nested resources	No, default is platform default

Parameters specified as nested elements

`<resourcelist>` accepts arbitrary many resource(collection)s as nested elements.

In addition `<resourcelist>` supports nested `<filterchain>` elements that can be used to filter/modify the read resources before their lines get expanded. Such a nested element corresponds to a filterchain.

Examples

The following example copies a file from the first URL of several alternatives that can actually be reached. It assumes that the file mirrors.txt looks like

```
    mirrors.txt:
http://best.mirror.example.org/
http://second.best.mirror.example.org/mirror/of/best/
https://yet.another.mirror/
http://the.original.site/
```

```
    <copy todir="${target}">
      <first>
        <restrict>
          <resourcelist>
            <file file="mirrors.txt"/>
          </resourcelist>
          <exists/>
        </restrict>
      </first>
    </copy>
```

Selectors

Selectors are a mechanism whereby the files that make up a `<fileset>` can be selected based on criteria other than filename as provided by the `<include>` and `<exclude>` tags.

How to use a Selector

A selector is an element of FileSet, and appears within it. It can also be defined outside of any target by using the `<selector>` tag and then using it as a reference.

Different selectors have different attributes. Some selectors can contain other selectors, and these are called `Selector Containers`. There is also a category of selectors that allow user-defined extensions, called `Custom Selectors`. The ones built in to Ant are called `Core Selectors`.

Core Selectors

Core selectors are the ones that come standard with Ant. They can be used within a fileset and can be contained within Selector Containers.

The core selectors are:

- `<contains>` - Select files that contain a particular text string
- `<date>` - Select files that have been modified either before or after a particular date and time
- `<depend>` - Select files that have been modified more recently than equivalent files elsewhere
- `<depth>` - Select files that appear so many directories down in a directory tree
- `<different>` - Select files that are different from those elsewhere
- `<filename>` - Select files whose name matches a particular pattern. Equivalent to the include and exclude elements of a patternset.
- `<present>` - Select files that either do or do not exist in some other location
- `<containsregexp>` - Select files that match a regular expression
- `<size>` - Select files that are larger or smaller than a particular number of bytes.
- `<type>` - Select files that are either regular files or directories.
- `<modified>` - Select files if the return value of the configured algorithm is different from that stored in a cache.
- `<signedselector>` - Select files if they are signed, and optionally if they have a signature of a certain name.
- `<scriptselector>` - Use a BSF or JSR 223 scripting language to create your own selector
- `<readable>` - Select files if they are readable.
- `<writable>` - Select files if they are writable.

Contains Selector

The `<contains>` tag in a FileSet limits the files defined by that fileset to only those which contain the string specified by the `text` attribute. .

The `<contains>` selector can be used as a ResourceSelector (see the <restrict> ResourceCollection).

Attribute	Description	Required
text	Specifies the text that every file must contain	Yes
casesensitive	Whether to pay attention to case when looking for the string in the `text` attribute.	No

	Default is true.	
ignorewhitespace	Whether to eliminate whitespace before checking for the string in the `text` attribute. Default is false.	No

Here is an example of how to use the Contains Selector:

```
<fileset dir="${doc.path}" includes="**/*.html">
    <contains text="script" casesensitive="no"/>
</fileset>
```

Selects all the HTML files that contain the string `script`.

Date Selector

The `<date>` tag in a FileSet will put a limit on the files specified by the include tag, so that tags whose last modified date does not meet the date limits specified by the selector will not end up being selected.

Attribute	Description	Required
datetime	Specifies the date and time to test for. Should be in the format MM/DD/YYYY HH:MM AM_or_PM, or an alternative pattern specified via the *pattern* attribute.	At least one of the two.
millis	The number of milliseconds since 1970 that should be tested for. It is usually much easier to use the datetime attribute.	
when	Indicates how to interpret the date, whether the files to be selected are those whose last modified times should be before, after, or equal to the specified value. Acceptable values for this attribute are: • before - select files whose last modified date is before the indicated date • after - select files whose last modified date is after the indicated date • equal - select files whose last modified date is this exact date The default is equal.	No
granularity	The number of milliseconds leeway to use when comparing file modification times. This is needed because not every file system supports tracking the last modified time to the millisecond level. Default is 0 milliseconds, or 2 seconds on DOS systems.	No
pattern	The `SimpleDateFormat`-compatible pattern to use when interpreting the *datetime* attribute. *Since Ant 1.6.2*	No
checkdirs	Indicates whether or not to check dates on directories.	No, defaults to *false*

Here is an example of how to use the Date Selector:

```
<fileset dir="${jar.path}" includes="**/*.jar">
    <date datetime="01/01/2001 12:00 AM" when="before"/>
</fileset>
```

Selects all JAR files which were last modified before midnight January 1, 2001.

Depend Selector

The `<depend>` tag selects files whose last modified date is later than another, equivalent file in another location.

The <depend> tag supports the use of a contained <mapper> element to define the location of the file to be compared against. If no <mapper> element is specified, the identity type mapper is used.

The <depend> selector is case-sensitive.

Attribute	Description	Required
targetdir	The base directory to look for the files to compare against. The precise location depends on a combination of this attribute and the <mapper> element, if any.	Yes
granularity	The number of milliseconds leeway to give before deciding a file is out of date. This is needed because not every file system supports tracking the last modified time to the millisecond level. Default is 0 milliseconds, or 2 seconds on DOS systems.	No

Here is an example of how to use the Depend Selector:

```
<fileset dir="${ant.1.5}/src/main" includes="**/*.java">
    <depend targetdir="${ant.1.4.1}/src/main"/>
</fileset>
```

Selects all the Java source files which were modified in the 1.5 release.

Depth Selector

The <depth> tag selects files based on how many directory levels deep they are in relation to the base directory of the fileset.

Attribute	Description	Required
min	The minimum number of directory levels below the base directory that a file must be in order to be selected. Default is no limit.	At least one of the two.
max	The maximum number of directory levels below the base directory that a file can be and still be selected. Default is no limit.	

Here is an example of how to use the Depth Selector:

```
<fileset dir="${doc.path}" includes="**/*">
    <depth max="1"/>
</fileset>
```

Selects all files in the base directory and one directory below that.

Different Selector

The <different> selector will select a file if it is deemed to be 'different' from an equivalent file in another location. The rules for determining difference between the two files are as follows:

1. If a file is only present in the resource collection you apply the selector to but not in targetdir (or after applying the mapper) the file is selected.
2. If a file is only present in targetdir (or after applying the mapper) it is ignored.
3. Files with different lengths are different.
4. If ignoreFileTimes is turned off, then differing file timestamps will cause files to be regarded as different.
5. Unless ignoreContents is set to true, a byte-for-byte check is run against the two files.

This is a useful selector to work with programs and tasks that don't handle dependency checking properly; even if a predecessor task always creates its output files, followup tasks can be driven off copies made with a different selector,

so their dependencies are driven on the absolute state of the files, not just a timestamp. For example: anything fetched from a web site, or the output of some program. To reduce the amount of checking, when using this task inside a `<copy>` task, set `preservelastmodified` to *true* to propagate the timestamp from the source file to the destination file.

The `<different>` selector supports the use of a contained `<mapper>` element to define the location of the file to be compared against. If no `<mapper>` element is specified, the `identity` type mapper is used.

Attribute	Description	Required
targetdir	The base directory to look for the files to compare against. The precise location depends on a combination of this attribute and the `<mapper>` element, if any.	Yes
ignoreFileTimes	Whether to use file times in the comparison or not. Default is true (time differences are ignored).	No
ignoreContents	Whether to do a byte per byte compare. Default is false (contents are compared). Since Ant 1.6.3	No
granularity	The number of milliseconds leeway to give before deciding a file is out of date. This is needed because not every file system supports tracking the last modified time to the millisecond level. Default is 0 milliseconds, or 2 seconds on DOS systems.	No

Here is an example of how to use the Different Selector:

```
<fileset dir="${ant.1.5}/src/main" includes="**/*.java">
    <different targetdir="${ant.1.4.1}/src/main"
        ignoreFileTimes="true"/>
</fileset>
```

Compares all the Java source files between the 1.4.1 and the 1.5 release and selects those who are different, disregarding file times.

Filename Selector

The `<filename>` tag acts like the `<include>` and `<exclude>` tags within a fileset. By using a selector instead, however, one can combine it with all the other selectors using whatever selector container is desired.

The `<filename>` selector is case-sensitive.

Attribute	Description	Required
name	The name of files to select. The name parameter can contain the standard Ant wildcard characters.	Exactly one of the two
regex	The regular expression matching files to select.	
casesensitive	Whether to pay attention to case when looking at file names. Default is "true".	No
negate	Whether to reverse the effects of this filename selection, therefore emulating an exclude rather than include tag. Default is "false".	No

Here is an example of how to use the Filename Selector:

```
<fileset dir="${doc.path}" includes="**/*">
    <filename name="**/*.css"/>
</fileset>
```

Selects all the cascading style sheet files.

Present Selector

The <present> tag selects files that have an equivalent file in another directory tree.

The <present> tag supports the use of a contained <mapper> element to define the location of the file to be tested against. If no <mapper> element is specified, the identity type mapper is used.

The <present> selector is case-sensitive.

Attribute	Description	Required
targetdir	The base directory to look for the files to compare against. The precise location depends on a combination of this attribute and the <mapper> element, if any.	Yes
present	Whether we are requiring that a file is present in the src directory tree only, or in both the src and the target directory tree. Valid values are: • srconly - select files only if they are in the src directory tree but not in the target directory tree • both - select files only if they are present both in the src and target directory trees Default is both. Setting this attribute to "srconly" is equivalent to wrapping the selector in the <not> selector container.	No

Here is an example of how to use the Present Selector:

```
<fileset dir="${ant.1.5}/src/main" includes="**/*.java">
    <present present="srconly" targetdir="${ant.1.4.1}/src/main"/>
</fileset>
```

Selects all the Java source files which are new in the 1.5 release.

Regular Expression Selector

The <containsregexp> tag in a FileSet limits the files defined by that fileset to only those which contents contain a match to the regular expression specified by the expression attribute.

The <containsregexp> selector can be used as a ResourceSelector (see the <restrict> ResourceCollection).

Attribute	Description	Required
expression	Specifies the regular expression that must match true in every file	Yes

Here is an example of how to use the regular expression Selector:

```
<fileset dir="${doc.path}" includes="*.txt">
    <containsregexp expression="[4-6]\.[0-9]"/>
</fileset>
```

Selects all the text files that match the regular expression (have a 4,5 or 6 followed by a period and a number from 0 to 9).

Size Selector

The <size> tag in a FileSet will put a limit on the files specified by the include tag, so that tags which do not meet the size limits specified by the selector will not end up being selected.

Attribute	Description	Required
value	The size of the file which should be tested for.	Yes
units	The units that the `value` attribute is expressed in. When using the standard single letter SI designations, such as "k","M", or "G", multiples of 1000 are used. If you want to use power of 2 units, use the IEC standard: "Ki" for 1024, "Mi" for 1048576, and so on. The default is no units, which means the `value` attribute expresses the exact number of bytes.	No
when	Indicates how to interpret the size, whether the files to be selected should be larger, smaller, or equal to that value. Acceptable values for this attribute are: • less - select files less than the indicated size • more - select files greater than the indicated size • equal - select files this exact size The default is equal.	No

Here is an example of how to use the Size Selector:

```
<fileset dir="${jar.path}">
  <patternset>
    <include name="**/*.jar"/>
  </patternset>
  <size value="4" units="Ki" when="more"/>
</fileset>
```

Selects all JAR files that are larger than 4096 bytes.

Type Selector

The `<type>` tag selects files of a certain type: directory or regular.

Attribute	Description	Required
type	The type of file which should be tested for. Acceptable values are: • file - regular files • dir - directories	Yes

Here is an example of how to use the Type Selector to select only directories in `${src}`

```
<fileset dir="${src}">
  <type type="dir"/>
</fileset>
```

The Type Selector is often used in conjunction with other selectors. For example, to select files that also exist in a `template` directory, but avoid selecting empty directories, use:

```
<fileset dir="${src}">
    <and>
        <present targetdir="template"/>
        <type type="file"/>
    </and>
</fileset>
```

Modified Selector

The `<modified>` selector computes a value for a file, compares that to the value stored in a cache and select the file, if these two values differ.

Because this selector is highly configurable the order in which the selection is done is:

1. get the absolute path for the file
2. get the cached value from the configured cache (absolute path as key)
3. get the new value from the configured algorithm
4. compare these two values with the configured comparator
5. update the cache if needed and requested
6. do the selection according to the comparison result

The comparison, computing of the hashvalue and the store is done by implementation of special interfaces. Therefore they may provide additional parameters.

The `<modified>` selector can be used as a ResourceSelector (see the <restrict> ResourceCollection). In that case it maps simple file resources to files and does its job. If the resource is from another type, the `<modified>` selector tries to (**attention!**) copy the content into a local file for computing the hashvalue.

Attribute	Description	Required
algorithm	The type of algorithm should be used. Acceptable values are (further information see later): • hashvalue - HashvalueAlgorithm • digest - DigestAlgorithm • checksum - ChecksumAlgorithm	No, defaults to *digest*
cache	The type of cache should be used. Acceptable values are (further information see later): • propertyfile - PropertyfileCache	No, defaults to *propertyfile*
comparator	The type of comparator should be used. Acceptable values are (further information see later): • equal - EqualComparator • rule - java.text.RuleBasedCollator *(see note for restrictions)*	No, defaults to *equal*
algorithmclass	Classname of custom algorithm implementation. Lower priority than *algorithm*.	No
cacheclass	Classname of custom cache implementation. Lower priority than *cache*.	No
comparatorclass	Classname of custom comparator implementation. Lower priority than *comparator*.	No
update	Should the cache be updated when values differ? (boolean)	No, defaults to *true*
seldirs	Should directories be selected? (boolean)	No, defaults to *true*
selres	Should Resources whithout an InputStream, and therefore without checking, be selected? (boolean)	No, defaults to *true*. Only relevant when used as ResourceSelector.

delayupdate	If set to *true*, the storage of the cache will be delayed until the next finished BuildEvent; task finished, target finished or build finished, whichever comes first. This is provided for increased performance. If set to *false*, the storage of the cache will happen with each change. This attribute depends upon the *update* attribute. (boolean)	No, defaults to *true*

These attributes can be set with nested `<param/>` tags. With `<param/>` tags you can set other values too - as long as they are named according to the following rules:

- **algorithm** : same as attribute algorithm
- **cache** : same as attribute cache
- **comparator** : same as attribute comparator
- **algorithmclass** : same as attribute algorithmclass
- **cacheclass** : same as attribute cacheclass
- **comparatorclass** : same as attribute comparatorclass
- **update** : same as attribute update
- **seldirs** : same as attribute seldirs
- **algorithm.*** : Value is transfered to the algorithm via its *set*XX-methods
- **cache.*** : Value is transfered to the cache via its *set*XX-methods
- **comparator.*** : Value is transfered to the comparator via its *set*XX-methods

Algorithm options	
Name	**Description**
hashvalue	Reads the content of a file into a java.lang.String and use thats hashValue(). No additional configuration required.
digest	Uses java.security.MessageDigest. This Algorithm supports the following attributes: • *algorithm.algorithm* (optional): Name of the Digest algorithm (e.g. 'MD5' or 'SHA', default = *MD5*) • *algorithm.provider* (optional): Name of the Digest provider (default = *null*)
checksum	Uses java.util.zip.Checksum. This Algorithm supports the following attributes: • *algorithm.algorithm* (optional): Name of the algorithm (e.g. 'CRC' or 'ADLER', default = *CRC*)
Cache options	
propertyfile	Use the java.util.Properties class and its possibility to load and store to file. This Cache implementation supports the following attributes: • *cache.cachefile* (optional): Name of the properties-file (default = *cache.properties*)
Comparator options	
equal	Very simple object comparison.
rule	Uses *java.text.RuleBasedCollator* for Object comparison. *(see note for restrictions)*

The `<modified>` selector supports a nested `<classpath>` element that represents a PATH like structure for finding custom interface implementations.

Here are some examples of how to use the Modified Selector:

```
    <copy todir="dest">
```

```
        <fileset dir="src">
            <modified/>
        </fileset>
    </copy>
```

This will copy all files from *src* to *dest* which content has changed. Using an updating PropertyfileCache with cache.properties and MD5-DigestAlgorithm.

```
    <copy todir="dest">
        <fileset dir="src">
            <modified update="true"
                      seldirs="true"
                      cache="propertyfile"
                      algorithm="digest"
                      comparator="equal">
                <param name="cache.cachefile"     value="cache.properties"/>
                <param name="algorithm.algorithm" value="MD5"/>
            </modified>
        </fileset>
    </copy>
```

This is the same example rewritten as CoreSelector with setting the all the values (same as defaults are).

```
    <copy todir="dest">
        <fileset dir="src">
            <custom
class="org.apache.tools.ant.types.selectors.modifiedselector.ModifiedSelector">
                <param name="update"     value="true"/>
                <param name="seldirs"    value="true"/>
                <param name="cache"      value="propertyfile"/>
                <param name="algorithm"  value="digest"/>
                <param name="comparator" value="equal"/>
                <param name="cache.cachefile"     value="cache.properties"/>
                <param name="algorithm.algorithm" value="MD5"/>
            </custom>
        </fileset>
    </copy>
```

And this is the same rewritten as CustomSelector.

```
  <target name="generate-and-upload-site">
      <echo> generate the site using forrest </echo>
      <antcall target="site"/>

      <echo> upload the changed file </echo>
      <ftp server="${ftp.server}" userid="${ftp.user}" password="${ftp.pwd}">
          <fileset dir="htdocs/manual">
              <modified/>
          </fileset>
      </ftp>
  </target>
```

A useful scenario for this selector inside a build environment for homepage generation (e.g. with Apache Forrest). Here all **changed** files are uploaded to the server. The CacheSelector saves therefore much upload time.

```
  <modified cacheclassname="com.mycompany.MyCache">
      <classpath>
          <pathelement location="lib/mycompony-antutil.jar"/>
      </classpath>
  </modified>
```

Uses `com.mycompany.MyCache` from a jar outside of Ants own classpath as cache implementation

Note on RuleBasedCollator

The RuleBasedCollator needs a format for its work, but its needed while instantiation. There is a problem in the initialization algorithm for this case. Therefore you should not use this (or tell me the workaround :-).

Signed Selector

The `<signedselector>` tag selects signed files and optionaly signed with a certain name.

This selector has been added in Apache Ant 1.7.

Attribute	Description	Required
name	The signature name to check for.	no

Readable Selector

The `<readable>` selector selects only files that are readable. Ant only invokes `java.io.File#canRead` so if a file is unreadable but the Java VM cannot detect this state, this selector will still select the file.

Writable Selector

The `<writable>` selector selects only files that are writable. Ant only invokes `java.io.File#canWrite` so if a file is unwritable but the Java VM cannot detect this state, this selector will still select the file.

Script Selector

The `<scriptselector>` element enables you to write a complex selection algorithm in any Apache BSF or JSR 223 supported language. See the Script task for an explanation of scripts and dependencies.

This selector was added in Apache Ant 1.7.

Attribute	Description	Required
language	language of the script.	yes
manager	The script engine manager to use. See the script task for using this attribute.	No - default is "auto"
src	filename of the script	no
setbeans	whether to have all properties, references and targets as global variables in the script.	No, default is "true".
classpath	The classpath to pass into the script.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No

This selector can take a nested <classpath> element. See the script task on how to use this element.

If no `src` attribute is supplied, the script must be nested inside the selector declaration.

The embedded script is invoked for every test, with the bean `self` is bound to the selector. It has an attribute `selected` must can be set using `setSelected(boolean)` to select that file.

The following beans are configured for every script, alongside the classic set of project, properties, and targets.

Bean	Description	Type
self	selector instance	org.apache.tools.ant.types.optional
filename	filename of the selection	String
file	file of the selection	java.io.File
basedir	Fileset base directory	java.io.File

The `self` bean maps to the selector, which has the following attributes. Only the `selected` flag is writeable, the rest are read only via their getter methods.

Attribute	Description	Type
selected	writeable flag to select this file	boolean
filename	filename of the selection	String
file	file of the selection	java.io.File
basedir	Fileset base directory	java.io.File

Example

```
    <scriptselector language="javascript">
      self.setSelected(true);
    </scriptselector>
```

Selects every file.

```
    <scriptselector language="javascript">
      self.setSelected((filename.length%2)==0);
    </scriptselector>
```

Select files whose filename length is even.

Selector Containers

To create more complex selections, a variety of selectors that contain other selectors are available for your use. They combine the selections of their child selectors in various ways.

The selector containers are:

- `<and>` - select a file only if all the contained selectors select it.
- `<majority>` - select a file if a majority of its selectors select it.
- `<none>` - select a file only if none of the contained selectors select it.
- `<not>` - can contain only one selector, and reverses what it selects and doesn't select.
- `<or>` - selects a file if any one of the contained selectors selects it.
- `<selector>` - contains only one selector and forwards all requests to it without alteration, provided that any `"if"` or `"unless"` conditions are met. This is the selector to use if you want to define a reference. It is usable as an element of `<project>`. It is also the one to use if you want selection of files to be dependent on Ant property settings.

All selector containers can contain any other selector, including other containers, as an element. Using containers, the selector tags can be arbitrarily deep. Here is a complete list of allowable selector elements within a container:

- `<and>`
- `<contains>`
- `<custom>`
- `<date>`
- `<depend>`

- <depth>
- <filename>
- <majority>
- <none>
- <not>
- <or>
- <present>
- <selector>
- <size>

And Selector

The <and> tag selects files that are selected by all of the elements it contains. It returns as soon as it finds a selector that does not select the file, so it is not guaranteed to check every selector.

Here is an example of how to use the And Selector:

```
<fileset dir="${dist}" includes="**/*.jar">
    <and>
        <size value="4" units="Ki" when="more"/>
        <date datetime="01/01/2001 12:00 AM" when="before"/>
    </and>
</fileset>
```

Selects all the JAR file larger than 4096 bytes which haven't been update since the last millenium.

Majority Selector

The <majority> tag selects files provided that a majority of the contained elements also select it. Ties are dealt with as specified by the allowtie attribute.

Attribute	Description	Required
allowtie	Whether files should be selected if there are an even number of selectors selecting them as are not selecting them. Default is true.	No

Here is an example of how to use the Majority Selector:

```
<fileset dir="${docs}" includes="**/*.html">
    <majority>
        <contains text="project" casesensitive="false"/>
        <contains text="taskdef" casesensitive="false"/>
        <contains text="IntrospectionHelper" casesensitive="true"/>
    </majority>
</fileset>
```

Selects all the HTML files which contain at least two of the three phrases "project", "taskdef", and "IntrospectionHelper" (this last phrase must match case exactly).

None Selector

The <none> tag selects files that are not selected by any of the elements it contains. It returns as soon as it finds a selector that selects the file, so it is not guaranteed to check every selector.

Here is an example of how to use the None Selector:

```
<fileset dir="${src}" includes="**/*.java">
    <none>
        <present targetdir="${dest}"/>
        <present targetdir="${dest}">
            <mapper type="glob" from="*.java" to="*.class"/>
        </present>
    </none>
</fileset>
```

Selects only Java files which do not have equivalent java or class files in the dest directory.

Not Selector

The `<not>` tag reverses the meaning of the single selector it contains.

Here is an example of how to use the Not Selector:

```
<fileset dir="${src}" includes="**/*.java">
    <not>
        <contains text="test"/>
    </not>
</fileset>
```

Selects all the files in the src directory that do not contain the string "test".

Or Selector

The `<or>` tag selects files that are selected by any one of the elements it contains. It returns as soon as it finds a selector that selects the file, so it is not guaranteed to check every selector.

Here is an example of how to use the Or Selector:

```
<fileset dir="${basedir}">
    <or>
        <depth max="0"/>
        <filename name="*.png"/>
        <filename name="*.gif"/>
        <filename name="*.jpg"/>
    </or>
</fileset>
```

Selects all the files in the top directory along with all the image files below it.

Selector Reference

The `<selector>` tag is used to create selectors that can be reused through references. It is the only selector which can be used outside of any target, as an element of the `<project>` tag. It can contain only one other selector, but of course that selector can be a container.

The `<selector>` tag can also be used to select files conditionally based on whether an Ant property exists or not. This functionality is realized using the `"if"` and `"unless"` attributes in exactly the same way they are used on targets or on the `<include>` and `<exclude>` tags within a `<patternset>`.

Attribute	Description	Required
if	Allow files to be selected only if the named property is set.	No
unless	Allow files to be selected only if the named property is **not** set.	No

Here is an example of how to use the Selector Reference:

```
<project default="all" basedir="./ant">

    <selector id="completed">
        <none>
            <depend targetdir="build/classes">
                <mapper type="glob" from="*.java" to="*.class"/>
            </depend>
            <depend targetdir="docs/manual/api">
                <mapper type="glob" from="*.java" to="*.html"/>
            </depend>
        </none>
    </selector>

    <target>
        <zip>
            <fileset dir="src/main" includes="**/*.java">
                <selector refid="completed"/>
            </fileset>
        </zip>
    </target>

</project>
```

Zips up all the Java files which have an up-to-date equivalent class file and javadoc file associated with them.

And an example of selecting files conditionally, based on whether properties are set:

```
<fileset dir="${working.copy}">
    <or>
        <selector if="include.tests">
            <filename name="**/*Test.class">
        </selector>
        <selector if="include.source">
            <and>
                <filename name="**/*.java">
                <not>
                    <selector unless="include.tests">
                        <filename name="**/*Test.java">
                    </selector>
                </not>
            </and>
        </selector>
    </or>
</fileset>
```

A fileset that conditionally contains Java source files and Test source and class files.

Custom Selectors

You can write your own selectors and use them within the selector containers by specifying them within the `<custom>` tag.

First, you have to write your selector class in Java. The only requirement it must meet in order to be a selector is that it implements the `org.apache.tools.ant.types.selectors.FileSelector` interface, which contains a single method. See Programming Selectors in Ant for more information.

Once that is written, you include it in your build file by using the `<custom>` tag.

Attribute	Description	Required
classname	The name of your class that implements `org.apache.tools.ant.types.selectors.FileSelector`.	Yes
classpath	The classpath to use in order to load the custom selector class. If neither this classpath nor the classpathref are specified, the class will be loaded from the classpath that Ant uses.	No

classpathref	A reference to a classpath previously defined. If neither this reference nor the classpath above are specified, the class will be loaded from the classpath that Ant uses.	No

Here is how you use `<custom>` to use your class as a selector:

```
<fileset dir="${mydir}" includes="**/*">
    <custom classname="com.mydomain.MySelector">
        <param name="myattribute" value="myvalue"/>
    </custom>
</fileset>
```

A number of core selectors can also be used as custom selectors by specifying their attributes using `<param>` elements. These are

- Contains Selector with classname `org.apache.tools.ant.types.selectors.ContainsSelector`
- Date Selector with classname `org.apache.tools.ant.types.selectors.DateSelector`
- Depth Selector with classname `org.apache.tools.ant.types.selectors.DepthSelector`
- Filename Selector with classname `org.apache.tools.ant.types.selectors.FilenameSelector`
- Size Selector with classname `org.apache.tools.ant.types.selectors.SizeSelector`

Here is the example from the Depth Selector section rewritten to use the selector through `<custom>`.

```
<fileset dir="${doc.path}" includes="**/*">
    <custom classname="org.apache.tools.ant.types.selectors.DepthSelector">
        <param name="max" value="1"/>
    </custom>
</fileset>
```

Selects all files in the base directory and one directory below that.

For more details concerning writing your own selectors, consult Programming Selectors in Ant.

TarFileSet

TarFileSet has been added as a stand-alone type in Ant 1.7.

A `<tarfileset>` is a special form of a `<fileset>` which can behave in 2 different ways :

- When the *src* attribute is used - or a nested resource collection has been specified, the tarfileset is populated with tar entries found in the file *src*.
- When the *dir* attribute is used, the tarfileset is populated with filesystem files found under *dir*.

`<tarfileset>` supports all attributes of `<fileset>` in addition to those listed below.

A tarfileset can be defined with the *id* attribute and referred to with the *refid* attribute. This is also true for tarfileset which has been added in Ant 1.7.

Parameters

Attribute	Description	Required
prefix	all files in the fileset are prefixed with that path in the archive.	No
fullpath	the file described by the fileset is placed at that exact location in the archive.	No
src	may be used in place of the *dir* attribute to specify a tar file whose contents will be extracted and included in the archive.	No
filemode	A 3 digit octal string, specify the user, group and other modes in the standard Unix fashion. Only applies to plain files. Default is 644.	No
dirmode	A 3 digit octal string, specify the user, group and other modes in the standard Unix fashion. Only applies to directories. Default is 755.	No
username	The username for the tar entry. This is not the same as the UID.	No
group	The groupname for the tar entry. This is not the same as the GID.	No
uid	The user identifier (UID) for the tar entry. This is an integer value and is not the same as the username.	No
gid	The group identifier (GID) for the tar entry.	No
erroronmissingarchive	Specify what happens if the archive does not exist. If true, a build error will happen; if false, the fileset will be ignored/empty. Defaults to true. *Since Ant 1.8.0*	No

The *fullpath* attribute can only be set for filesets that represent a single file. The *prefix* and *fullpath* attributes cannot both be set on the same fileset.

When using the *src* attribute, include and exclude patterns may be used to specify a subset of the archive for inclusion in the archive as with the *dir* attribute.

Please note that currently only the tar task uses the permission and ownership attributes.

Parameters specified as nested elements

any resource or single element resource collection

The specified resource will be used as src.

Examples

```
<copy todir="some-dir">
  <tarfileset includes="lib/**">
    <bzip2resource>
      <url url="http://example.org/dist/some-archive.tar.bz2"/>
    </bzip2resource>
  </tarfileset>
</copy>
```

downloads the archive some-archive.tar.bz2, uncompresses and extracts it on the fly, copies the contents of the lib directory into some-dir and discards the rest of the archive. File timestamps will be compared between the archive's entries and files inside the target directory, no files get overwritten unless they are out-of-date.

XMLCatalog

An XMLCatalog is a catalog of public resources such as DTDs or entities that are referenced in an XML document. Catalogs are typically used to make web references to resources point to a locally cached copy of the resource.

This allows the XML Parser, XSLT Processor or other consumer of XML documents to efficiently allow a local substitution for a resource available on the web.

Note: This task *uses, but does not depend on* external libraries not included in the Ant distribution. See Library Dependencies for more information.

This data type provides a catalog of resource locations based on the OASIS "Open Catalog" standard. The catalog entries are used both for Entity resolution and URI resolution, in accordance with the `org.xml.sax.EntityResolver` and `javax.xml.transform.URIResolver` interfaces as defined in the Java API for XML Processing (JAXP) Specification.

For example, in a `web.xml` file, the DTD is referenced as:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

The XML processor, without XMLCatalog support, would need to retrieve the DTD from the URL specified whenever validation of the document was required.

This can be very time consuming during the build process, especially where network throughput is limited. Alternatively, you can do the following:

1. Copy `web-app_2_2.dtd` onto your local disk somewhere (either in the filesystem or even embedded inside a jar or zip file on the classpath).
2. Create an `<xmlcatalog>` with a `<dtd>` element whose `location` attribute points to the file.
3. Success! The XML processor will now use the local copy instead of calling out to the internet.

XMLCatalogs can appear inside tasks that support this feature or at the same level as `target` - i.e., as children of `project` for reuse across different tasks, e.g. XML Validation and XSLT Transformation. The XML Validate task uses XMLCatalogs for entity resolution. The XSLT Transformation task uses XMLCatalogs for both entity and URI resolution.

XMLCatalogs are specified as either a reference to another XMLCatalog, defined previously in a build file, or as a list of `dtd` or `entity` locations. In addition, external catalog files may be specified in a nested `catalogpath` , but they will be ignored unless the resolver library from xml-commons is available in the system classpath. **Due to backwards incompatible changes in the resolver code after the release of resolver 1.0, Ant will not support resolver.jar in version 1.0 - we expect a resolver release 1.1 to happen before Ant 1.6 gets released.** A separate classpath for entity resolution may be specified inline via nested `classpath` elements; otherwise the system classpath is used for this as well.

XMLCatalogs can also be nested inside other XMLCatalogs. For example, a "superset" XMLCatalog could be made by including several nested XMLCatalogs that referred to other, previously defined XMLCatalogs.

Resource locations can be specified either in-line or in external catalog file(s), or both. In order to use an external catalog file, the xml-commons resolver library ("resolver.jar") must be in your path. External catalog files may be either plain text format or XML format. If the xml-commons resolver library is not found in the classpath, external catalog files, specified in `catalogpath`, will be ignored and a warning will be logged. In this case, however, processing of inline entries will proceed normally.

Currently, only `<dtd>` and `<entity>` elements may be specified inline; these roughly correspond to OASIS catalog entry types `PUBLIC` and `URI` respectively. By contrast, external catalog files may use any of the entry types defined in the +OASIS specification.

Entity/DTD/URI Resolution Algorithm

When an entity, DTD, or URI is looked up by the XML processor, the XMLCatalog searches its list of entries to see if any match. That is, it attempts to match the `publicId` attribute of each entry with the PublicID or URI of the entity to be resolved. Assuming a matching entry is found, XMLCatalog then executes the following steps:

1. Filesystem lookup

The `location` is first looked up in the filesystem. If the `location` is a relative path, the ant project basedir attribute is used as the base directory. If the `location` specifies an absolute path, it is used as is. Once we have an absolute path in hand, we check to see if a valid and readable file exists at that path. If so, we are done. If not, we proceed to the next step.

2. Classpath lookup

The `location` is next looked up in the classpath. Recall that jar files are merely fancy zip files. For classpath lookup, the `location` is used as is (no base is prepended). We use a Classloader to attempt to load the resource from the classpath. For example, if hello.jar is in the classpath and it contains `foo/bar/blat.dtd` it will resolve an entity whose `location` is `foo/bar/blat.dtd`. Of course, it will *not* resolve an entity whose `location` is `blat.dtd`.

3a. Apache xml-commons resolver lookup

What happens next depends on whether the resolver library from xml-commons is available on the classpath. If so, we defer all further attempts at resolving to it. The resolver library supports extremely sophisticated functionality like URL rewriting and so on, which can be accessed by making the appropriate entries in external catalog files (XMLCatalog does not yet provide inline support for all of the entries defined in the OASIS standard).

3. URL-space lookup

Finally, we attempt to make a URL out of the `location`. At first this may seem like this would defeat the purpose of XMLCatalogs -- why go back out to the internet? But in fact, this can be used to (in a sense) implement HTTP redirects, substituting one URL for another. The mapped-to URL might also be served by a local web server. If the URL resolves to a valid and readable resource, we are done. Otherwise, we give up. In this case, the XML processor will perform its normal resolution algorithm. Depending on the processor configuration, further resolution failures may or may not result in fatal (i.e. build-ending) errors.

XMLCatalog attributes

Attribute	Description	Required
id	a unique name for an XMLCatalog, used for referencing the XMLCatalog's contents from another XMLCatalog	No
refid	the `id` of another XMLCatalog whose contents you would like to be used for this XMLCatalog	No

XMLCatalog nested elements

dtd/entity

The `dtd` and `entity` elements used to specify XMLCatalogs are identical in their structure

Attribute	Description	Required
publicId	The public identifier used when defining a dtd or entity, e.g. `"-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"`	Yes
location	The location of the local replacement to be used for the public identifier specified. This may be specified as a file name, resource name found on the classpath, or a URL. Relative paths will be resolved according to the base, which by default is the Ant project basedir.	Yes

classpath

The classpath to use for entity resolution. The nested `<classpath>` is a path-like structure.

catalogpath

The nested `catalogpath` element is a path-like structure listing catalog files to search. All files in this path are assumed to be OASIS catalog files, in either plain text format or XML format. Entries specifying nonexistent files will be ignored. If the resolver library from xml-commons is not available in the classpath, all `catalogpaths` will be ignored and a warning will be logged.

Examples

Set up an XMLCatalog with a single dtd referenced locally in a user's home directory:

```
<xmlcatalog>
    <dtd
        publicId="-//OASIS//DTD DocBook XML V4.1.2//EN"
        location="/home/dion/downloads/docbook/docbookx.dtd"/>
</xmlcatalog>
```

Set up an XMLCatalog with a multiple dtds to be found either in the filesystem (relative to the Ant project basedir) or in the classpath:

```
<xmlcatalog id="commonDTDs">
    <dtd
        publicId="-//OASIS//DTD DocBook XML V4.1.2//EN"
        location="docbook/docbookx.dtd"/>
    <dtd
        publicId="-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
        location="web-app_2_2.dtd"/>
</xmlcatalog>
```

Set up an XMLCatalog with a combination of DTDs and entities as well as a nested XMLCatalog and external catalog files in both formats:

```
<xmlcatalog id="allcatalogs">
    <dtd
        publicId="-//ArielPartners//DTD XML Article V1.0//EN"
        location="com/arielpartners/knowledgebase/dtd/article.dtd"/>
    <entity
        publicId="LargeLogo"
        location="com/arielpartners/images/ariel-logo-large.gif"/>
    <xmlcatalog refid="commonDTDs"/>
        <catalogpath>
            <pathelement location="/etc/sgml/catalog"/>
            <fileset
                dir="/anetwork/drive"
                includes="**/catalog"/>
            <fileset
```

```
                    dir="/my/catalogs"
                    includes="**/catalog.xml"/>
            </catalogpath>
        </xmlcatalog>
    </xmlcatalog>
```

To reference the above XMLCatalog in an `xslt` task:

```
    <xslt basedir="${source.doc}"
           destdir="${dest.xdocs}"
           extension=".xml"
           style="${source.xsl.converter.docbook}"
           includes="**/*.xml"
           force="true">
        <xmlcatalog refid="allcatalogs"/>
    </xslt>
```

ZipFileSet

A <zipfileset> is a special form of a <fileset> which can behave in 2 different ways :

- When the *src* attribute is used - or a nested resource collection has been specified (*since Ant 1.7*), the zipfileset is populated with zip entries found in the file *src*.
- When the *dir* attribute is used, the zipfileset is populated with filesystem files found under *dir*.

<zipfileset> supports all attributes of <fileset> in addition to those listed below.

Since Ant 1.6, a zipfileset can be defined with the *id* attribute and referred to with the *refid* attribute.

Parameters

Attribute	Description	Required
prefix	all files in the fileset are prefixed with that path in the archive.	No
fullpath	the file described by the fileset is placed at that exact location in the archive.	No
src	may be used in place of the *dir* attribute to specify a zip file whose contents will be extracted and included in the archive.	No
filemode	A 3 digit octal string, specify the user, group and other modes in the standard Unix fashion. Only applies to plain files. Default is 644. *since Ant 1.5.2*.	No
dirmode	A 3 digit octal string, specify the user, group and other modes in the standard Unix fashion. Only applies to directories. Default is 755. *since Ant 1.5.2*.	No
encoding	The character encoding to use for filenames inside the zip file. For a list of possible values see http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html. Defaults to the platform's default character encoding. **Only supported by zipfileset.**	No
erroronmissingarchive	Specify what happens if the archive does not exist. If true, a build error will happen; if false, the fileset will be ignored/empty. Defaults to true. *Since Ant 1.8.0*	No

The *fullpath* attribute can only be set for filesets that represent a single file. The *prefix* and *fullpath* attributes cannot both be set on the same fileset.

When using the *src* attribute, include and exclude patterns may be used to specify a subset of the archive for inclusion in the archive as with the *dir* attribute.

Please note that currently only the tar and zip tasks use the permission.

Parameters specified as nested elements

any file system based resource or single element resource collection

The specified resource will be used as src.

Examples

```
<zip destfile="${dist}/manual.zip">
  <zipfileset dir="htdocs/manual" prefix="docs/user-guide"/>
  <zipfileset dir="." includes="ChangeLog27.txt" fullpath="docs/ChangeLog.txt"/>
```

```
    <zipfileset src="examples.zip" includes="**/*.html" prefix="docs/examples"/>
  </zip>
```

zips all files in the `htdocs/manual` directory into the `docs/user-guide` directory in the archive, adds the file `ChangeLog27.txt` in the current directory as `docs/ChangeLog.txt`, and includes all the html files in `examples.zip` under `docs/examples`. The archive might end up containing the files:

```
docs/user-guide/html/index.html
docs/ChangeLog.txt
docs/examples/index.html
```

ClassFileSet

A classfileset is a specialised type of fileset which, given a set of "root" classes, will include all of the class files upon which the root classes depend. This is typically used to create a jar with all of the required classes for a particular application.

classfilesets are typically used by reference. They are declared with an "id" value and this is then used as a reference where a normal fileset is expected.

This type requires the `jakarta-BCEL` library.

Attributes

The class fileset support the following attributes in addition to those supported by the standard fileset:

Attribute	Description	Required
rootclass	A single root class name	No

Nested Elements

Root

When more than one root class is required, multiple nested `<root>` elements may be used

Attribute	Description	Required
classname	The fully qualified name of the root class	Yes

RootFileSet

A root fileset is used to add a set of root classes from a fileset. In this case the entries in the fileset are expected to be Java class files. The name of the Java class is determined by the relative location of the classfile in the fileset. So, the file `org/apache/tools/ant/Project.class` corresponds to the Java class `org.apache.tools.ant.Project`.

Examples

```
<classfileset id="reqdClasses" dir="${classes.dir}">
  <root classname="org.apache.tools.ant.Project"/>
</classfileset>
```

This example creates a fileset containing all the class files upon which the `org.apache.tools.ant.Project` class depends. This fileset could then be used to create a jar.

```
<jar destfile="minimal.jar">
  <fileset refid="reqdClasses"/>
</jar>
```

```
<classfileset id="reqdClasses" dir="${classes.dir}">
  <rootfileset dir="${classes.dir}" includes="org/apache/tools/ant/Project*.class"/>
</classfileset>
```

This example constructs the classfileset using all the class with names starting with Project in the org.apache.tools.ant package

Extension

Utility type that represents either an available "Optional Package" (formerly known as "Standard Extension") as described in the manifest of a JAR file, or the requirement for such an optional package.

Note that this type works with extensions as defined by the "Optional Package" specification. For more information about optional packages, see the document *Optional Package Versioning* in the documentation bundle for your Java2 Standard Edition package, in file `guide/extensions/versioning.html` or online at http://java.sun.com/j2se/1.3/docs/guide/extensions/versioning.html.

Attributes

The extension type supports the following attributes:

Attribute	Description	Required
extensionName	The name of extension	yes
specificationVersion	The version of extension specification (Must be in dewey decimal aka dotted decimal notation. 3.2.4)	no
specificationVendor	The specification vendor	no
implementationVersion	The version of extension implementation (Must be in dewey decimal aka dotted decimal notation. 3.2.4)	no
implementationVendor	The implementation vendor	no
implementationVendorId	The implementation vendor ID	no
implementationURL	The url from which to retrieve extension.	no

Examples

```
<extension id="e1"
    extensionName="MyExtensions"
    specificationVersion="1.0"
    specificationVendor="Peter Donald"
    implementationVendorID="vv"
    implementationVendor="Apache"
    implementationVersion="2.0"
    implementationURL="http://somewhere.com/myExt.jar"/>
```

Fully specific extension object.

```
<extension id="e1"
    extensionName="MyExtensions"
    specificationVersion="1.0"
    specificationVendor="Peter Donald"/>
```

Extension object that just species the specification details.

ExtensionSet

Utility type that represents a set of Extensions.

Note that this type works with extensions as defined by the "Optional Package" specification. For more information about optional packages, see the document *Optional Package Versioning* in the documentation bundle for your Java2 Standard Edition package, in file `guide/extensions/versioning.html` or online at http://java.sun.com/j2se/1.3/docs/guide/extensions/versioning.html.

Nested Elements

extension

Extension object to add to set.

fileset

FileSets all files contained contained within set that are jars and implement an extension are added to extension set.

LibFileSet

All files contained contained within set that are jars and implement an extension are added to extension set. However the extension information may be modified by attributes of libfileset

Examples

```
<extension id="e1"
    extensionName="MyExtensions"
    specificationVersion="1.0"
    specificationVendor="Peter Donald"
    implementationVendorID="vv"
    implementationVendor="Apache"
    implementationVersion="2.0"
    implementationURL="http://somewhere.com/myExt.jar"/>

<libfileset id="lfs"
               includeUrl="true"
               includeImpl="false"
               dir="tools/lib">
  <include name="*.jar"/>
</libfileset>

<extensionSet id="exts">
  <libfileset dir="lib">
    <include name="*.jar"/>
  </libfileset>
  <libfileset refid="lfs"/>
  <extension refid="e1"/>
</extensionSet>
```

XML Namespace Support

Ant 1.6 introduces support for XML namespaces.

History

All releases of Ant prior to Ant 1.6 do not support XML namespaces. No support basically implies two things here:

- Element names correspond to the "qname" of the tags, which is usually the same as the local name. But if the build file writer uses colons in names of defined tasks/types, those become part of the element name. Turning on namespace support gives colon-separated prefixes in tag names a special meaning, and thus build files using colons in user-defined tasks and types will break.
- Attributes with the names 'xmlns' and 'xmlns:<prefix>' are not treated specially, which means that custom tasks and types have actually been able to use such attributes as parameter names. Again, such tasks/types are going to break when namespace support is enabled on the parser.

Use of colons in element names has been discouraged in the past, and using any attribute starting with "xml" is actually strongly discouraged by the XML spec to reserve such names for future use.

Motivation

In build files using a lot of custom and third-party tasks, it is easy to get into name conflicts. When individual types are defined, the build file writer can do some namespacing manually (for example, using "tomcat-deploy" instead of just "deploy"). But when defining whole libraries of types using the <typedef> 'resource' attribute, the build file writer has no chance to override or even prefix the names supplied by the library.

Assigning Namespaces

Adding a 'prefix' attribute to <typedef> might have been enough, but XML already has a well-known method for namespacing. Thus, instead of adding a 'prefix' attribute, the <typedef> and <taskdef> tasks get a 'uri' attribute, which stores the URI of the XML namespace with which the type should be associated:

```
<typedef resource="org/example/tasks.properties" uri="http://example.org/tasks"/>
<my:task xmlns:my="http://example.org/tasks">
   ...
</my:task>
```

As the above example demonstrates, the namespace URI needs to be specified at least twice: one time as the value of the 'uri' attribute, and another time to actually map the namespace to occurrences of elements from that namespace, by using the 'xmlns' attribute. This mapping can happen at any level in the build file:

```
<project name="test" xmlns:my="http://example.org/tasks">
  <typedef resource="org/example/tasks.properties" uri="http://example.org/tasks"/>
  <my:task>
    ...
  </my:task>
</project>
```

Use of a namespace prefix is of course optional. Therefore the example could also look like this:

```
<project name="test">
  <typedef resource="org/example/tasks.properties" uri="http://example.org/tasks"/>
  <task xmlns="http://example.org/tasks">
    ...
  </task>
</project>
```

Here, the namespace is set as the default namespace for the <task> element and all its descendants.

Default namespace

The default namespace used by Ant is "antlib:org.apache.tools.ant".

```
<typedef resource="org/example/tasks.properties" uri="antlib:org.apache.tools.ant"/>
<task>
      ....
</task>
```

Namespaces and Nested Elements

Almost always in Ant 1.6, elements nested inside a namespaced element have the same namespace as their parent. So if 'task' in the example above allowed a nested 'config' element, the build file snippet would look like this:

```
<typedef resource="org/example/tasks.properties" uri="http://example.org/tasks"/>
<my:task xmlns:my="http://example.org/tasks">
  <my:config a="foo" b="bar"/>
  ...
</my:task>
```

If the element allows or requires a lot of nested elements, the prefix needs to be used for every nested element. Making the namespace the default can reduce the verbosity of the script:

```
<typedef resource="org/example/tasks.properties" uri="http://example.org/tasks"/>
         <task xmlns="http://example.org/tasks">
         <config a="foo" b="bar"/>
  ...
         </task>
```

From Ant 1.6.2, elements nested inside a namespaced element may also be in Ant's default namespace. This means that the following is now allowed:

```
<typedef resource="org/example/tasks.properties"
  uri="http://example.org/tasks"/>
<my:task xmlns:my="http://example.org/tasks">
  <config a="foo" b="bar"/>
  ...
</my:task>
```

Namespaces and Attributes

Attributes are only used to configure the element they belong to if:

- they have no namespace (note that the default namespace does *not* apply to attributes)
- they are in the same namespace as the element they belong to

Other attributes are simply ignored.

This means that both:

```
<my:task xmlns:my="http://example.org/tasks">
  <my:config a="foo" b="bar"/>
  ...
</my:task>
```

and

```
<my:task xmlns:my="http://example.org/tasks">
```

```
  <my:config my:a="foo" my:b="bar"/>
  ...
</my:task>
```

result in the parameters "a" and "b" being used as parameters to configure the nested "config" element.

It also means that you can use attributes from other namespaces to markup the build file with extra metadata, such as RDF and XML-Schema (whether that's a good thing or not). The same is not true for elements from unknown namespaces, which result in a error.

Mixing Elements from Different Namespaces

Now comes the difficult part: elements from different namespaces can be woven together under certain circumstances. This has a lot to do with the Ant 1.6 add type introspection rules: Ant types and tasks are now free to accept arbritrary named types as nested elements, as long as the concrete type implements the interface expected by the task/type. The most obvious example for this is the `<condition>` task, which supports various nested conditions, all of which extend the interface `Condition`. To integrate a custom condition in Ant, you can now simply `<typedef>` the condition, and then use it anywhere nested conditions are allowed (assuming the containing element has a generic `add(Condition)` or `addConfigured(Condition)` method):

```
<typedef resource="org/example/conditions.properties" uri="http://example.org/conditions"/>
<condition property="prop" xmlns="http://example.org/conditions">
  <and>
    <available file="bla.txt"/>
    <my:condition a="foo"/>
  </and>
</condition>
```

In Ant 1.6, this feature cannot be used as much as we'd all like to: a lot of code has not yet been adapted to the new introspection rules, and elements like Ant's built-in conditions and selectors are not really types in 1.6. This is expected to change in Ant 1.7.

Namespaces and Antlib

The new AntLib feature is also very much integrated with the namespace support in Ant 1.6. Basically, you can "import" Antlibs simply by using a special scheme for the namespace URI: the `antlib` scheme, which expects the package name in which a special `antlib.xml` file is located.

Antlib

Description

An antlib file is an xml file with a root element of "antlib". Antlib's elements are ant definition tasks - like and Taskdef, or any ant task that extends `org.apache.tools.ant.taskdefs.AntlibDefinition`.

The current set of declarations bundled with Ant that do this are:

1. Typedef
2. Taskdef
3. Macrodef
4. Presetdef
5. Scriptdef

A group of tasks and types may be defined together in an antlib file. For example the file *sample.xml* contains the following:

```
<?xml version="1.0"?>
<antlib>
   <typedef name="if" classname="org.acme.ant.If"/>
   <typedef name="scriptpathmapper"
            classname="org.acme.ant.ScriptPathMapper"
            onerror="ignore"/>
   <macrodef name="print">
      <attribute name="file"/>
      <sequential>
         <concat taskname="print">
            <fileset dir="." includes="@{file}"/>
         </concat>
      </sequential>
   </macrodef>
</antlib>
```

It defines two types or tasks, *if* and *scriptpathmapper*. This antlib file may be used in a build script as follows:

```
<typedef file="sample.xml"/>
```

The other attributes of `<typedef>` may be used as well. For example, assuming that the *sample.xml* is in a jar file *sample.jar* also containing the classes, the following build fragment will define the *if* and *scriptpathmapper* tasks/types and place them in the namespace uri *samples:/acme.org*.

```
<typedef resource="org/acme/ant/sample.xml"
         uri="samples:/acme.org"/>
```

The definitions may then be used as follows:

```
<sample:if valuetrue="${props}" xmlns:sample="samples:/acme.org">
   <sample:scriptpathmapper language="beanshell">
      some bean shell
   </sample:scriptpathmapper>
</sample:if>
```

Antlib namespace

The name space URIs with the pattern **antlib:*java package*** are given special treatment.

When ant encounters a element with a namespace URI with this pattern, it will check to see if there is a resource of the name *antlib.xml* in the package directory in the default classpath.

For example, assuming that the file *antcontrib.jar* has been placed in the directory *${ant.home}/lib* and it contains the resource *net/sf/antcontrib/antlib.xml* which has all antcontrib's definitions defined, the following build file will automatically load the antcontrib definitions at location *HERE*:

```
<project default="deletetest" xmlns:antcontrib="antlib:net.sf.antcontrib">
   <macrodef name="showdir">
      <attribute name="dir"/>
      <sequential>
         <antcontrib:shellscript shell="bash">  <!-- HERE -->
            ls -Rl @{dir}
         </antcontrib:shellscript>
      </sequential>
   </macrodef>

   <target name="deletetest">
      <delete dir="a" quiet="yes"/>
      <mkdir dir="a/b"/>
      <touch file="a/a.txt"/>
      <touch file="a/b/b.txt"/>
      <delete>
         <fileset dir="a"/>
      </delete>
      <showdir dir="a"/>
   </target>
</project>
```

The requirement that the resource is in the default classpath may be removed in future versions of Ant.

Load antlib from inside of the buildfile

If you want to seperate the antlib from your local Ant installation, e.g. because you want to hold that jar in your projects SCM system, you have to specify a classpath, so that Ant could find that jar. The best solution is loading the antlib with `<taskdef>`.

```
<project xmlns:antcontrib="antlib:net.sf.antcontrib">
   <taskdef uri="antlib:net.sf.antcontrib"
            resource="net/sf/antcontrib/antlib.xml"
            classpath="path/to/ant-contrib.jar"/>

   <target name="iterate">
      <antcontrib:for param="file">
         <fileset dir="."/>
         <sequential>
            <echo message="- @{file}"/>
         </sequential>
      </antcontrib:for>
   </target>
</project>
```

Current namespace

Definitions defined in antlibs may be used in antlibs. However the namespace that definitions are placed in are dependent on the `<typedef>` that uses the antlib. To deal with this problem, the definitions are placed in the namepace URI *ant:current* for the duration of the antlib execution. For example the following antlib defines the task `<if>`, the type `<isallowed>` and a macro `<ifallowed>` that makes use of the task and type:

```
<antlib xmlns:current="ant:current">
   <taskdef name="if" classname="org.acme.ant.If"/>
   <typedef name="isallowed" classname="org.acme.ant.Isallowed"/>
   <macrodef name="ifallowed">
      <attribute name="action"/>
      <element name="do"/>
      <sequential>
```

```
         <current:if>
            <current:isallowed test="@{action}"/>
            <current:then>
               <do/>
            </current:then>
         </current:if>
      </sequential>
   </macrodef>
</antlib>
```

Other examples and comments

Antlibs may make use of other antlibs.

As the names defined in the antlib are in the namespace uri as specified by the calling `<typedef>` or by automatic element resolution, one may reuse names from core ant types and tasks, provided the caller uses a namespace uri. For example, the following antlib may be used to define defaults for various tasks:

```
<antlib xmlns:antcontrib="antlib:net.sf.antcontrib">
   <presetdef name="javac">
      <javac deprecation="${deprecation}"
             debug="${debug}"/>
   </presetdef>
   <presetdef name="delete">
      <delete quiet="yes"/>
   </presetdef>
   <presetdef name="shellscript">
      <antcontrib:shellscript shell="bash"/>
   </presetdef>
</antlib>
```

This may be used as follows:

```
<project xmlns:local="localpresets">
   <typedef file="localpresets.xml" uri="localpresets"/>
   <local:shellscript>
      echo "hello world"
   </local:shellscript>
</project>
```

Custom Components

Overview

Custom components are conditions, selectors, filters and other objects that are defined outside ant core.

In Ant 1.6 custom conditions, selectors and filters has been overhauled.

It is now possible to define custom conditions, selectors and filters that behave like Ant Core components. This is achieved by allowing datatypes defined in build scripts to be used as custom components if the class of the datatype is compatible, or has been adapted by an adapter class.

The old methods of defining custom components are still supported.

Definition and use

A custom component is a normal Java class that implements a particular interface or extends a particular class, or has been adapted to the interface or class.

It is exactly like writing a custom task. One defines attributes and nested elements by writing *setter* methods and *add* methods.

After the class has been written, it is added to the ant system by using `<typedef>`.

Custom Conditions

Custom conditions are datatypes that implement `org.apache.tools.ant.taskdefs.condition.Condition`. For example a custom condition that returns true if a string is all upper case could be written as:

```
package com.mydomain;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.taskdefs.condition.Condition;

public class AllUpperCaseCondition implements Condition {
    private String value;

    // The setter for the "value" attribute
    public void setValue(String value) {
        this.value = value;
    }

    // This method evaluates the condition
    public boolean eval() {
        if (value == null) {
            throw new BuildException("value attribute is not set");
        }
        return value.toUpperCase().equals(value);
   }
}
```

Adding the condition to the system is achieved as follows:

```
<typedef
    name="alluppercase"
    classname="com.mydomain.AllUpperCaseCondition"
    classpath="${mydomain.classes}"/>
```

This condition can now be used wherever a Core Ant condition is used.

```
<condition property="allupper">
   <alluppercase value="THIS IS ALL UPPER CASE"/>
</condition>
```

Custom Selectors

Custom selectors are datatypes that implement `org.apache.tools.ant.types.selectors.FileSelector`.

There is only one method required. `public boolean isSelected(File basedir, String filename, File file)`. It returns true or false depending on whether the given file should be selected or not.

An example of a custom selection that selects filenames ending in ".java" would be:

```
package com.mydomain;
import java.io.File;
import org.apache.tools.ant.types.selectors.FileSelector;
public class JavaSelector implements FileSelector {
    public boolean isSelected(File b, String filename, File f) {
       return filename.toLowerCase().endsWith(".java");
    }
}
```

Adding the selector to the system is achieved as follows:

```
<typedef
    name="javaselector"
    classname="com.mydomain.JavaSelector"
    classpath="${mydomain.classes}"/>
```

This selector can now be used wherever a Core Ant selector is used, for example:

```
<copy todir="to">
   <fileset dir="src">
      <javaselector/>
   </fileset>
</copy>
```

One may use `org.apache.tools.ant.types.selectors.BaseSelector`, a convenience class that provides reasonable default behaviour. It has some predefined behaviours you can take advantage of. Any time you encounter a problem when setting attributes or adding tags, you can call setError(String errmsg) and the class will know that there is a problem. Then, at the top of your `isSelected()` method call `validate()` and a BuildException will be thrown with the contents of your error message. The `validate()` method also gives you a last chance to check your settings for consistency because it calls `verifySettings()`. Override this method and call `setError()` within it if you detect any problems in how your selector is set up.

To write custom selector containers one should extend `org.apache.tools.ant.types.selectors.BaseSelectorContainer`. Implement the `public boolean isSelected(File baseDir, String filename, File file)` method to do the right thing. Chances are you'll want to iterate over the selectors under you, so use `selectorElements()` to get an iterator that will do that.

For example to create a selector container that will select files if a certain number of contained selectors select, one could write a selector as follows:

```
public class MatchNumberSelectors extends BaseSelectorContainer {
    private int number = -1;
    public void setNumber(int number) {
        this.number = number;
    }
```

```
    public void verifySettings() {
        if (number < 0) {
           throw new BuildException("Number attribute should be set");
        }
    }
    public boolean isSelected(File baseDir, String filename, File file) {
        validate();
        int numberSelected = 0;
        for (Enumeration e = selectorElements(); e.hasNextElement();) {
            FileSelector s = (FileSelector) e.nextElement();
            if (s.isSelected(baseDir, filename, file)) {
                numberSelected++;
            }
        }
        return numberSelected == number;
    }
}
```

To define and use this selector one could do:

```
<typedef name="numberselected"
         classname="com.mydomain.MatchNumberSelectors"/>
...
<fileset dir="${src.path}">
   <numberselected number="2">
      <contains text="script" casesensitive="no"/>
      <size value="4" units="Ki" when="more"/>
      <javaselector/>
   </numberselected>
</fileset>
```

The custom selector

The custom selector was the pre ant 1.6 way of defining custom selectors. This method is still supported for backward compatibility.

You can write your own selectors and use them within the selector containers by specifying them within the `<custom>` tag.

To create a new Custom Selector, you have to create a class that implements `org.apache.tools.ant.types.selectors.ExtendFileSelector`. The easiest way to do that is through the convenience base class `org.apache.tools.ant.types.selectors.BaseExtendSelector`, which provides all of the methods for supporting `<param>` tags. First, override the `isSelected()` method, and optionally the `verifySettings()` method. If your custom selector requires parameters to be set, you can also override the `setParameters()` method and interpret the parameters that are passed in any way you like. Several of the core selectors demonstrate how to do that because they can also be used as custom selectors.

Once that is written, you include it in your build file by using the `<custom>` tag.

Attribute	Description	Required
classname	The name of your class that implements `org.apache.tools.ant.types.selectors.FileSelector`.	Yes
classpath	The classpath to use in order to load the custom selector class. If neither this classpath nor the classpathref are specified, the class will be loaded from the classpath that Ant uses.	No
classpathref	A reference to a classpath previously defined. If neither this reference nor the classpath above are specified, the class will be loaded from the classpath that Ant uses.	No

Here is how you use `<custom>` to use your class as a selector:

```
<fileset dir="${mydir}" includes="**/*">
    <custom classname="com.mydomain.MySelector">
        <param name="myattribute" value="myvalue"/>
```

```
    </custom>
</fileset>
```

The core selectors that can also be used as custom selectors are

- Contains Selector with classname `org.apache.tools.ant.types.selectors.ContainsSelector`
- Date Selector with classname `org.apache.tools.ant.types.selectors.DateSelector`
- Depth Selector with classname `org.apache.tools.ant.types.selectors.DepthSelector`
- Filename Selector with classname `org.apache.tools.ant.types.selectors.FilenameSelector`
- Size Selector with classname `org.apache.tools.ant.types.selectors.SizeSelector`

Here is the example from the Depth Selector section rewritten to use the selector through `<custom>`.

```
<fileset dir="${doc.path}" includes="**/*">
    <custom classname="org.apache.tools.ant.types.selectors.DepthSelector">
        <param name="max" value="1"/>
    </custom>
</fileset>
```

Selects all files in the base directory and one directory below that.

Custom Filter Readers

Custom filter readers selectors are datatypes that implement `org.apache.tools.ant.types.filters.ChainableReader`.

There is only one method required. `Reader chain(Reader reader)`. This returns a reader that filters input from the specified reader.

For example a filterreader that removes every second character could be:

```
public class RemoveOddCharacters implements ChainableReader {
   public Reader chain(Reader reader) {
      return new BaseFilterReader(reader) {
          int count = 0;
          public int read() throws IOException {
              while (true) {
                int c = in.read();
                if (c == -1) {
                    return c;
                }
                count++;
                if ((count % 2) == 1) {
                    return c;
                }
              }
          }
      }
   }
}
```

For line oriented filters it may be easier to extend `ChainableFilterReader` an inner class of `org.apache.tools.ant.filters.TokenFilter`.

For example a filter that appends the line number could be

```
public class AddLineNumber extends ChainableReaderFilter {
   private void lineNumber = 0;
   public String filter(String string) {
      lineNumber++;
      return "" + lineNumber + "\t" + string;
   }
}
```

Developing with Ant

Writing Your Own Task

It is very easy to write your own task:

1. Create a Java class that extends `org.apache.tools.ant.Task` or another class that was designed to be extended.
2. For each attribute, write a *setter* method. The setter method must be a `public void` method that takes a single argument. The name of the method must begin with `set`, followed by the attribute name, with the first character of the name in uppercase, and the rest in lowercase*. That is, to support an attribute named `file` you create a method `setFile`. Depending on the type of the argument, Ant will perform some conversions for you, see below.
3. If your task shall contain other tasks as nested elements (like `parallel`), your class must implement the interface `org.apache.tools.ant.TaskContainer`. If you do so, your task can not support any other nested elements. See below.
4. If the task should support character data (text nested between the start end end tags), write a `public void addText(String)` method. Note that Ant does **not** expand properties on the text it passes to the task.
5. For each nested element, write a *create*, *add* or *addConfigured* method. A create method must be a `public` method that takes no arguments and returns an `Object` type. The name of the create method must begin with `create`, followed by the element name. An add (or addConfigured) method must be a `public void` method that takes a single argument of an `Object` type with a no-argument constructor. The name of the add (addConfigured) method must begin with `add` (`addConfigured`), followed by the element name. For a more complete discussion see below.
6. Write a `public void execute` method, with no arguments, that throws a `BuildException`. This method implements the task itself.

---

* Actually the case of the letters after the first one doesn't really matter to Ant, using all lower case is a good convention, though.

The Life-cycle of a Task

1. The xml element that contains the tag corresponding to the task gets converted to an UnknownElement at parser time. This UnknownElement gets placed in a list within a target object, or recursivly within another UnknownElement.
2. When the target is executed, each UnknownElement is invoked using an `perform()` method. This instantiates the task. This means that tasks only gets instantiated at run time.
3. The task gets references to its project and location inside the buildfile via its inherited `project` and `location` variables.
4. If the user specified an `id` attribute to this task, the project registers a reference to this newly created task, at run time.
5. The task gets a reference to the target it belongs to via its inherited `target` variable.
6. `init()` is called at run time.
7. All child elements of the XML element corresponding to this task are created via this task's `createXXX()` methods or instantiated and added to this task via its `addXXX()` methods, at run time. Child elements corresponding to `addConfiguredXXX()` are created at this point but the actual `addCondifgired` method is not called.
8. All attributes of this task get set via their corresponding `setXXX` methods, at runtime.
9. The content character data sections inside the XML element corresponding to this task is added to the task via its `addText` method, at runtime.

`setXXX`

10. All attributes of all child elements get set via their corresponding methods, at runtime.
11. If child elements of the XML element corresponding to this task have been created for `addConfiguredXXX()` methods, those methods get invoked now.
12. `execute()` is called at runtime. If `target1` and `target2` both depend on `target3`, then running `'ant target1 target2'` will run all tasks in `target3` twice.

Conversions Ant will perform for attributes

Ant will always expand properties before it passes the value of an attribute to the corresponding setter method. **Since Ant 1.8**, it is possible to extend Ant's property handling such that a non-string Object may be the result of the evaluation of a string containing a single property reference. These will be assigned directly via setter methods of matching type. Since it requires some beyond-the-basics intervention to enable this behavior, it may be a good idea to flag attributes intended to permit this usage paradigm.

The most common way to write an attribute setter is to use a `java.lang.String` argument. In this case Ant will pass the literal value (after property expansion) to your task. But there is more! If the argument of you setter method is

- `boolean`, your method will be passed the value *true* if the value specified in the build file is one of `true`, `yes`, or `on` and *false* otherwise.
- `char` or `java.lang.Character`, your method will be passed the first character of the value specified in the build file.
- any other primitive type (`int`, `short` and so on), Ant will convert the value of the attribute into this type, thus making sure that you'll never receive input that is not a number for that attribute.
- `java.io.File`, Ant will first determine whether the value given in the build file represents an absolute path name. If not, Ant will interpret the value as a path name relative to the project's basedir.
- `org.apache.tools.ant.types.Resource` `org.apache.tools.ant.types.Resource`, Ant will resolve the string as a `java.io.File` as above, then pass in as a `org.apache.tools.ant.types.resources.FileResource`. **Since Ant 1.8**
- `org.apache.tools.ant.types.Path`, Ant will tokenize the value specified in the build file, accepting `:` and `;` as path separators. Relative path names will be interpreted as relative to the project's basedir.
- `java.lang.Class`, Ant will interpret the value given in the build file as a Java class name and load the named class from the system class loader.
- any other type that has a constructor with a single `String` argument, Ant will use this constructor to create a new instance from the value given in the build file.
- A subclass of `org.apache.tools.ant.types.EnumeratedAttribute`, Ant will invoke this classes `setValue` method. Use this if your task should support enumerated attributes (attributes with values that must be part of a predefined set of values). See `org/apache/tools/ant/taskdefs/FixCRLF.java` and the inner `AddAsisRemove` class used in `setCr` for an example.
- A (Java 5) enumeration. Ant will call the setter with the enum constant matching the value given in the build file. This is easier than using `EnumeratedAttribute` and can result in cleaner code, but of course your task will not run on JDK 1.4 or earlier. Note that any override of `toString()` in the enumeration is ignored; the build file must use the declared name (see `Enum.getName()`). You may wish to use lowercase enum constant names, in contrast to usual Java style, to look better in build files. *As of Ant 1.7.0.*

What happens if more than one setter method is present for a given attribute? A method taking a `String` argument will always lose against the more specific methods. If there are still more setters Ant could chose from, only one of them will be called, but we don't know which, this depends on the implementation of your Java virtual machine.

Supporting nested elements

Let's assume your task shall support nested elements with the name `inner`. First of all, you need a class that represents this nested element. Often you simply want to use one of Ant's classes like `org.apache.tools.ant.types.FileSet` to support nested `fileset` elements.

Attributes of the nested elements or nested child elements of them will be handled using the same mechanism used for tasks (i.e. setter methods for attributes, addText for nested text and create/add/addConfigured methods for child elements).

Now you have a class `NestedElement` that is supposed to be used for your nested `<inner>` elements, you have three options:

1. `public NestedElement createInner()`
2. `public void addInner(NestedElement anInner)`
3. `public void addConfiguredInner(NestedElement anInner)`

What is the difference?

Option 1 makes the task create the instance of `NestedElement`, there are no restrictions on the type. For the options 2 and 3, Ant has to create an instance of `NestedInner` before it can pass it to the task, this means, `NestedInner` must have a `public` no-arg constructor or a `public` one-arg constructor taking a Project class as a parameter. This is the only difference between options 1 and 2.

The difference between 2 and 3 is what Ant has done to the object before it passes it to the method. `addInner` will receive an object directly after the constructor has been called, while `addConfiguredInner` gets the object *after* the attributes and nested children for this new object have been handled.

What happens if you use more than one of the options? Only one of the methods will be called, but we don't know which, this depends on the implementation of your Java virtual machine.

Nested Types

If your task needs to nest an arbitary type that has been defined using `<typedef>` you have two options.

1. `public void add(Type type)`
2. `public void addConfigured(Type type)`

The difference between 1 and 2 is the same as between 2 and 3 in the previous section.

For example suppose one wanted to handle objects object of type org.apache.tools.ant.taskdefs.condition.Condition, one may have a class:

```
public class MyTask extends Task {
    private List conditions = new ArrayList();
    public void add(Condition c) {
        conditions.add(c);
    }
    public void execute() {
     // iterator over the conditions
    }
}
```

One may define and use this class like this:

```
<taskdef name="mytask" classname="MyTask" classpath="classes"/>
<typedef name="condition.equals"
         classname="org.apache.tools.ant.taskdefs.conditions.Equals"/>
<mytask>
    <condition.equals arg1="${debug}" arg2="true"/>
</mytask>
```

A more complicated example follows:

```
public class Sample {
    public static class MyFileSelector implements FileSelector {
         public void setAttrA(int a) {}
         public void setAttrB(int b) {}
         public void add(Path path) {}
         public boolean isSelected(File basedir, String filename, File file) {
             return true;
         }
     }

    interface MyInterface {
        void setVerbose(boolean val);
    }

    public static class BuildPath extends Path {
        public BuildPath(Project project) {
            super(project);
        }

        public void add(MyInterface inter) {}
        public void setUrl(String url) {}
    }

    public static class XInterface implements MyInterface {
        public void setVerbose(boolean x) {}
        public void setCount(int c) {}
    }
}
```

This class defines a number of static classes that implement/extend Path, MyFileSelector and MyInterface. These may be defined and used as follows:

```
<typedef name="myfileselector" classname="Sample$MyFileSelector"
         classpath="classes" loaderref="classes"/>
<typedef name="buildpath" classname="Sample$BuildPath"
         classpath="classes" loaderref="classes"/>
<typedef name="xinterface" classname="Sample$XInterface"
         classpath="classes" loaderref="classes"/>

<copy todir="copy-classes">
   <fileset dir="classes">
      <myfileselector attra="10" attrB="-10">
         <buildpath path="." url="abc">
            <xinterface count="4"/>
         </buildpath>
      </myfileselector>
   </fileset>
</copy>
```

TaskContainer

The `TaskContainer` consists of a single method, `addTask` that basically is the same as an add method for nested elements. The task instances will be configured (their attributes and nested elements have been handled) when your task's `execute` method gets invoked, but not before that.

When we said `execute` would be called, we lied ;-). In fact, Ant will call the `perform` method in `org.apache.tools.ant.Task`, which in turn calls `execute`. This method makes sure that Build Events will be triggered. If you execute the task instances nested into your task, you should also invoke `perform` on these instances instead of `execute`.

Example

Let's write our own task, which prints a message on the `System.out` stream. The task has one attribute, called `message`.

```
package com.mydomain;

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

public class MyVeryOwnTask extends Task {
    private String msg;

    // The method executing the task
    public void execute() throws BuildException {
        System.out.println(msg);
    }

    // The setter for the "message" attribute
    public void setMessage(String msg) {
        this.msg = msg;
    }
}
```

It's really this simple ;-)

Adding your task to the system is rather simple too:

1. Make sure the class that implements your task is in the classpath when starting Ant.
2. Add a `<taskdef>` element to your project. This actually adds your task to the system.
3. Use your task in the rest of the buildfile.

Example

```
<?xml version="1.0"?>

<project name="OwnTaskExample" default="main" basedir=".">
  <taskdef name="mytask" classname="com.mydomain.MyVeryOwnTask"/>

  <target name="main">
    <mytask message="Hello World! MyVeryOwnTask works!"/>
  </target>
</project>
```

Example 2

To use a task directly from the buildfile which created it, place the `<taskdef>` declaration inside a target *after the compilation*. Use the `classpath` attribute of `<taskdef>` to point to where the code has just been compiled.

```
<?xml version="1.0"?>

<project name="OwnTaskExample2" default="main" basedir=".">

  <target name="build" >
    <mkdir dir="build"/>
    <javac srcdir="source" destdir="build"/>
  </target>

  <target name="declare" depends="build">
    <taskdef name="mytask"
        classname="com.mydomain.MyVeryOwnTask"
        classpath="build"/>
  </target>

  <target name="main" depends="declare">
    <mytask message="Hello World! MyVeryOwnTask works!"/>
  </target>
</project>
```

Another way to add a task (more permanently), is to add the task name and implementing class name to the

`default.properties` file in the `org.apache.tools.ant.taskdefs` package. Then you can use it as if it were a built-in task.

Build Events

Ant is capable of generating build events as it performs the tasks necessary to build a project. Listeners can be attached to Ant to receive these events. This capability could be used, for example, to connect Ant to a GUI or to integrate Ant with an IDE.

To use build events you need to create an ant `Project` object. You can then call the `addBuildListener` method to add your listener to the project. Your listener must implement the `org.apache.tools.antBuildListener` interface. The listener will receive BuildEvents for the following events

- Build started
- Build finished
- Target started
- Target finished
- Task started
- Task finished
- Message logged

If the build file invokes another build file via `<ant>` or `<subant>` or uses `<antcall>`, you are creating a new Ant "project" that will send target and task level events of its own but never sends build started/finished events. Ant 1.6.2 introduces an extension of the BuildListener interface named SubBuildListener that will receive two new events for

- SubBuild started
- SubBuild finished

If you are interested in those events, all you need to do is to implement the new interface instead of BuildListener (and register the listener, of course).

If you wish to attach a listener from the command line you may use the `-listener` option. For example:

```
ant -listener org.apache.tools.ant.XmlLogger
```

will run Ant with a listener that generates an XML representation of the build progress. This listener is included with Ant, as is the default listener, which generates the logging to standard output.

Note: A listener must not access System.out and System.err directly since ouput on these streams is redirected by Ant's core to the build event system. Accessing these streams can cause an infinite loop in Ant. Depending on the version of Ant, this will either cause the build to terminate or the Java VM to run out of Stack space. A logger, also, may not access System.out and System.err directly. It must use the streams with which it has been configured.

Note2: All methods of a BuildListener except for the "Build Started" and "Build Finished" events may occur on several threads simultaneously - for example while Ant is executing a `<parallel>` task.

Source code integration

The other way to extend Ant through Java is to make changes to existing tasks, which is positively encouraged. Both

changes to the existing source and new tasks can be incorporated back into the Ant codebase, which benefits all users and spreads the maintenance load around.

Please consult the Getting Involved pages on the Jakarta web site for details on how to fetch the latest source and how to submit changes for reincorporation into the source tree.

Ant also has some task guidelines which provides some advice to people developing and testing tasks. Even if you intend to keep your tasks to yourself, you should still read this as it should be informative.

AntWork Plugin for the Jext Java Text Editor

by

- Klaus Hartlage (KHartlage@t-online.de)

You can download the plugin at: ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip

Installation instructions from the Readme.txt:

You have to enable the Jext Console to see the Ant output (menu: Edit->Options... - General Panel), because the Ant messages are redirected to the Jext console.

You can configure the Ant call in the Jext menu: Edit->Options... - Plugin Options - Antwork Plugin Panel; here you can set the ant home directory and the path to your build file.

You can start AntWork in the menu: Plugins->Ant->Work Now! In the appearing dialog box you can enter the target which you want to compile.

If a javac error occurs in the ant run an error-list opens within Jext. With a double-click on the error-message you jump to the error in the specified java text file.

Tasks Designed for Extension

These classes are designed to be extended. Always start here when writing your own task.

The links will not work in the online version of this document.

Class	Description
Task	Base class for all tasks.
AbstractCvsTask	Another task can extend this with some customized output processing
JDBCTask	Handles JDBC configuration needed by SQL type tasks.
MatchingTask	This is an abstract task that should be used by all those tasks that require to include or exclude files based on pattern matching.
Pack	Abstract Base class for pack tasks.
Unpack	Abstract Base class for unpack tasks.
DispatchTask	Abstract Base class for tasks that may have multiple actions.

InputHandler

Overview

When a task wants to prompt a user for input, it doesn't simply read the input from the console as this would make it impossible to embed Ant in an IDE. Instead it asks an implementation of the `org.apache.tools.ant.input.InputHandler` interface to prompt the user and hand the user input back to the task.

To do this, the task creates an `InputRequest` object and passes it to the `InputHandler` Such an `InputRequest` may know whether a given user input is valid and the `InputHandler` is supposed to reject all invalid input.

Exactly one `InputHandler` instance is associated with every Ant process, users can specify the implementation using the `-inputhandler` command line switch.

InputHandler

The `InputHandler` interface contains exactly one method

```
    void handleInput(InputRequest request)
        throws org.apache.tools.ant.BuildException;
```

with some pre- and postconditions. The main postcondition is that this method must not return unless the `request` considers the user input valid, it is allowed to throw an exception in this situation.

Ant comes with three built-in implementations of this interface:

DefaultInputHandler

This is the implementation you get, when you don't use the `-inputhandler` command line switch at all. This implementation will print the prompt encapsulated in the `request` object to Ant's logging system and re-prompt for input until the user enters something that is considered valid input by the `request` object. Input will be read from the console and the user will need to press the Return key.

PropertyFileInputHandler

This implementation is useful if you want to run unattended build processes. It reads all input from a properties file and makes the build fail if it cannot find valid input in this file. The name of the properties file must be specified in the Java system property `ant.input.properties`.

The prompt encapsulated in a `request` will be used as the key when looking up the input inside the properties file. If no input can be found, the input is considered invalid and an exception will be thrown.

Note that `ant.input.properties` must be a Java system property, not an Ant property. I.e. you cannot define it as a simple parameter to `ant`, but you can define it inside the `ANT_OPTS` environment variable.

GreedyInputHandler

Like the default implementation, this InputHandler reads from standard input. However, it consumes *all* available

input. This behavior is useful for sending Ant input via an OS pipe. **Since Ant 1.7**.

SecureInputHandler

This InputHandler calls `System.console().readPassword()`, available since Java 1.6. On earlier platforms it falls back to the behavior of DefaultInputHandler. **Since Ant 1.7.1**.

InputRequest

Instances of `org.apache.tools.ant.input.InputRequest` encapsulate the information necessary to ask a user for input and validate this input.

The instances of `InputRequest` itself will accept any input, but subclasses may use stricter validations. `org.apache.tools.ant.input.MultipleChoiceInputRequest` should be used if the user input must be part of a predefined set of choices.

Using Ant Tasks Outside of Ant

Rationale

Ant provides a rich set of tasks for buildfile creators and administrators. But what about programmers? Can the functionality provided by Ant tasks be used in java programs?

Yes, and its quite easy. Before getting into the details, however, we should mention the pros and cons of this approach:

Pros

Robust	Ant tasks are very robust. They have been banged on by many people. Ant tasks have been used in many different contexts, and have therefore been instrumented to take care of a great many boundary conditions and potentially obscure errors.
Cross Platform	Ant tasks are cross platform. They have been tested on all of the volume platforms, and several rather unusual ones (Netware and OS/390, to name a few).
Community Support	Using Ant tasks means you have less of your own code to support. Ant code is supported by the entire Apache Ant community.

Cons

Dependency on Ant Libraries	Obviously, if you use an Ant task in your code, you will have to add "ant.jar" to your path. Of course, you could use a code optimizer to remove the unnecessary classes, but you will still probably require a chunk of the Ant core.
Loss of Flexibility	At some point, if you find yourself having to modify the Ant code, it probably makes more sense to "roll your own." Of course, you can still steal some code snippets and good ideas. This is the beauty of open source!

Example

Let's say you want to unzip a zip file programmatically from java into a certain directory. Of course you could write your own routine to do this, but why not use the Ant task that has already been written?

In my example, I wanted to be able to unzip a file from within an XSLT Transformation. XSLT Transformers can be extended by plugging in static methods in java. I therefore need a function something like this:

```
/**
 * Unzip a zip file into a given directory.
 *
 * @param zipFilepath A pathname representing a local zip file
 * @param destinationDir where to unzip the archive to
 */
 static public void unzip(String zipFilepath, String destinationDir)
```

The Ant task to perform this function is `org.apache.tools.ant.taskdefs.Expand`. All we have to do is create a dummy Ant `Project` and `Target`, set the `Task` parameters that would normally be set in a buildfile, and call `execute()`.

First, let's make sure we have the proper includes:

```
import org.apache.tools.ant.Project;
import org.apache.tools.ant.Target;
import org.apache.tools.ant.taskdefs.Expand;
import java.io.File;
```

The function call is actually quite simple:

```
static public void unzip(String zipFilepath, String destinationDir) {

    final class Expander extends Expand {
        public Expander() {
            project = new Project();
            project.init();
            taskType = "unzip";
            taskName = "unzip";
            target = new Target();
        }
    }
    Expander expander = new Expander();
    expander.setSrc(new File(zipfile));
    expander.setDest(new File(destdir));
    expander.execute();
```

In actual practice, you will probably want to add your own error handling code and you may not want to use a local inner class. However, the point of the example is to show how an Ant task can be called programmatically in relatively few lines of code.

The question you are probably asking yourself at this point is: *How would I know which classes and methods have to be called in order to set up a dummy Project and Target?* The answer is: you don't. Ultimately, you have to be willing to get your feet wet and read the source code. The above example is merely designed to whet your appetite and get you started. Go for it!

The Ant frontend: ProjectHelper

What is a ProjectHelper?

The `ProjectHelper` in Ant is responsible for parsing the build file and creating java instances representing the build workflow. It also signals which kind of file it can parse, and which file name it expects as default input file.

Ant' default `ProjectHelper` (`org.apache.tools.ant.helper.ProjectHelper2`) parses the usual build.xml files. And if no build file is specified on the command line, it will expect to find a file named `build.xml`.

The immediate benefit of a such abstraction it that it is possible to make Ant understand other kind of descriptive languages than XML. Some experiments have been done around a pure java frontend, and a groovy one too (ask the dev mailing list for further info about these).

How is Ant is selecting the proper ProjectHelper

Ant knows about several implementations of `ProjectHelper` and has to decide which to use for each build file.

At startup Ant lists the all implementations found and keeps them in the same order they've been found in an internal 'repository':

- the first to be searched for is the one declared by the system property `org.apache.tools.ant.ProjectHelper` (see Java System Properties);
- then it searches with its class loader for a `ProjectHelper` service declarations in the META-INF: it searches in the classpath for a file `META-INF/services/org.apache.tools.ant.ProjectHelper`. This file will just contain the fully qualified name of the implementation of `ProjectHelper` to instanciate;
- it will also search with the system class loader for `ProjectHelper` service declarations in the META-INF;
- last but not least it will add its default `ProjectHelper` that can parse classical build.xml files.

In case of an error while trying to instanciate a `ProjectHelper`, Ant will log an error but won't stop. If you want further debugging info about the `ProjectHelper` internal 'repository', use the **system** property `ant.project-helper-repo.debug` and set it to `true`; the full stack trace will then also be printed.

When Ant is expected to parse a file, it will ask the `ProjectHelper` repository to find an implementation that will be able to parse the input file. Actually it will just iterate over the ordered list and the first implementation that returns `true` to `supportsBuildFile(File buildFile)` will be selected.

When Ant is started and no input file has been specified, it will search for a default input file. It will iterate over list of `ProjectHelpers` and will select the first one that expects a default file that actually exist.

Writing your own ProjectHelper

The class `org.apache.tools.ant.ProjectHelper` is the API expected to be implemented. So write your own `ProjectHelper` by extending that abstract class. You are then expected to implement at least the function `parse(Project project, Object source)`. Note also that your implementation will be instanciated by Ant, and it is expecting a default constructor with no arguments.

There are some functions that will help you define what your helper is capable of and what is is expecting:

- `getDefaultBuildFile()`: defines which file name is expected if none provided
- `supportsBuildFile(File buildFile)`: defines if your parser can parse the input file
- `canParseAntlibDescriptor(URL url)`: whether your implementation is capable of parsing a given Antlib descriptor. The base class returns `false`
- `parseAntlibDescriptor(Project containingProject, URL source)`: invoked to actually parse the Antlib descriptor if your implementation returned `true` for the previous method.

Now that you have your implementation ready, you have to declare it to Ant. Two solutions here:

- use the system property `org.apache.tools.ant.ProjectHelper` (see also the Java System Properties);
- use the service file in META-INF: in the jar you will build with your implementation, add a file `META-INF/services/org.apache.tools.ant.ProjectHelper`. And then in this file just put the fully qualified name of your implementation

Tutorial: Hello World with Ant

This document provides a step by step tutorial for starting java programming with Ant. It does **not** contain deeper knowledge about Java or Ant. This tutorial has the goal to let you see, how to do the easiest steps in Ant.

Content

Preparing the project

We want to separate the source from the generated files, so our java source files will be in `src` folder. All generated files should be under `build`, and there splitted into several subdirectories for the individual steps: `classes` for our compiled files and `jar` for our own JAR-file.

We have to create only the `src` directory. (Because I am working on Windows, here is the win-syntax - translate to your shell):

```
md src
```

The following simple Java class just prints a fixed message out to STDOUT, so just write this code into `src\oata\HelloWorld.java`.

```
package oata;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Now just try to compile and run that:

```
md build\classes
javac -sourcepath src -d build\classes src\oata\HelloWorld.java
java -cp build\classes oata.HelloWorld
```

which will result in

```
Hello World
```

Creating a jar-file is not very difficult. But creating a *startable* jar-file needs more steps: create a manifest-file containing the start class, creating the target directory and archiving the files.

```
echo Main-Class: oata.HelloWorld>myManifest
md build\jar
jar cfm build\jar\HelloWorld.jar myManifest -C build\classes .
java -jar build\jar\HelloWorld.jar
```

Note: Do not have blanks around the >-sign in the `echo Main-Class` instruction because it would falsify it!

Four steps to a running application

After finishing the java-only step we have to think about our build process. We *have* to compile our code, otherwise we couldn't start the program. Oh - "start" - yes, we could provide a target for that. We *should* package our application. Now it's only one class - but if you want to provide a download, no one would download several hundreds files ... (think about a complex Swing GUI - so let us create a jar file. A startable jar file would be nice ... And it's a good practise to have a "clean" target, which deletes all the generated stuff. Many failures could be solved just by a "clean build".

By default Ant uses `build.xml` as the name for a buildfile, so our `.\build.xml` would be:

```
<project>

    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes"/>
    </target>

    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="oata.HelloWorld"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

Now you can compile, package and run the application via

```
ant compile
ant jar
ant run
```

Or shorter with

```
ant compile jar run
```

While having a look at the buildfile, we will see some similar steps between Ant and the java-only commands:

java-only	Ant

`md build\classes`
`javac`
`    -sourcepath src`
`    -d build\classes`
`    src\oata\HelloWorld.java`
`echo Main-Class:`
`oata.HelloWorld>mf`
`md build\jar`
`jar cfm`
`    build\jar\HelloWorld.jar`
`    mf`
`    -C build\classes`
`    .`
 	
`java -jar build\jar\HelloWorld.jar`
	`<mkdir dir="build/classes"/>`
	`<javac`
	`    srcdir="src"`
	`    destdir="build/classes"/>`
	`<!-- automatically detected -->`
	`<!-- obsolete; done via manifest tag -->`
	`<mkdir dir="build/jar"/>`
	`<jar`
	`    destfile="build/jar/HelloWorld.jar"`
	
	`    basedir="build/classes">`
	`    <manifest>`
	`        <attribute name="Main-Class"`
	`value="oata.HelloWorld"/>`
	`    </manifest>`
	`</jar>`
	`<java jar="build/jar/HelloWorld.jar" fork="true"/>`

Enhance the build file

Now we have a working buildfile we could do some enhancements: many time you are referencing the same directories, main-class and jar-name are hard coded, and while invocation you have to remember the right order of build steps.

The first and second point would be addressed with *properties*, the third with a special property - an attribute of the <project>-tag and the fourth problem can be solved using dependencies.

```
<project name="HelloWorld" basedir="." default="main">

    <property name="src.dir"     value="src"/>

    <property name="build.dir"   value="build"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="jar.dir"     value="${build.dir}/jar"/>

    <property name="main-class"  value="oata.HelloWorld"/>



    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
    </target>

    <target name="clean-build" depends="clean,jar"/>

    <target name="main" depends="clean,run"/>

</project>
```

Now it's easier, just do a `ant` and you will get

```
Buildfile: build.xml

clean:

compile:
    [mkdir] Created dir: C:\...\build\classes
    [javac] Compiling 1 source file to C:\...\build\classes

jar:
    [mkdir] Created dir: C:\...\build\jar
      [jar] Building jar: C:\...\build\jar\HelloWorld.jar

run:
     [java] Hello World

main:
```

```
BUILD SUCCESSFUL
```

Using external libraries

Somehow told us not to use syso-statements. For log-Statements we should use a Logging-API - customizable on a high degree (including switching off during usual life (= not development) execution). We use Log4J for that, because

- it is not part of the JDK (1.4+) and we want to show how to use external libs
- it can run under JDK 1.2 (as Ant)
- it's highly configurable
- it's from Apache ;-)

We store our external libraries in a new directory `lib`. Log4J can be downloaded [1] from Logging's Homepage. Create the `lib` directory and extract the log4j-1.2.9.jar into that lib-directory. After that we have to modify our java source to use that library and our buildfile so that this library could be accessed during compilation and run.

Working with Log4J is documented inside its manual. Here we use the *MyApp*-example from the Short Manual [2]. First we have to modify the java source to use the logging framework:

```
package oata;

import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;

public class HelloWorld {
    static Logger logger = Logger.getLogger(HelloWorld.class);

    public static void main(String[] args) {
        BasicConfigurator.configure();
        logger.info("Hello World");           // the old SysO-statement
    }
}
```

Most of the modifications are "framework overhead" which has to be done once. The blue line is our "old System-out" statement.

Don't try to run `ant` - you will only get lot of compiler errors. Log4J is not inside the classpath so we have to do a little work here. But do not change the CLASSPATH environment variable! This is only for this project and maybe you would break other environments (this is one of the most famous mistakes when working with Ant). We introduce Log4J (or to be more precise: all libraries (jar-files) which are somewhere under `.\lib`) into our buildfile:

```
<project name="HelloWorld" basedir="." default="main">
    ...
    <property name="lib.dir"     value="lib"/>

    <path id="classpath">
        <fileset dir="${lib.dir}" includes="**/*.jar"/>
    </path>

    ...

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}" classpathref="classpath"/>
    </target>

    <target name="run" depends="jar">
        <java fork="true" classname="${main-class}">
            <classpath>
                <path refid="classpath"/>
                <path location="${jar.dir}/${ant.project.name}.jar"/>
            </classpath>
        </java>
```

```
    </target>

    ...
</project>
```

In this example we start our application not via its Main-Class manifest-attribute, because we could not provide a jarname *and* a classpath. So add our class in the red line to the already defined path and start as usual. Running `ant` would give (after the usual compile stuff):

```
[java] 0 [main] INFO oata.HelloWorld  - Hello World
```

What's that?

- *[java]* Ant task running at the moment
- *0* sorry don't know - some Log4J stuff
- *[main]* the running thread from our application
- *INFO* log level of that statement
- *oata.HelloWorld* source of that statement
- - separator
- *Hello World* the message

For another layout ... have a look inside Log4J's documentation about using other PatternLayout's.

Configuration files

Why we have used Log4J? "It's highly configurable"? No - all is hard coded! But that is not the debt of Log4J - it's ours. We had coded `BasicConfigurator.configure();` which implies a simple, but hard coded configuration. More confortable would be using a property file. In the java source delete the BasicConfiguration-line from the main() method (and the related import-statement). Log4J will search then for a configuration as described in it's manual. Then create a new file `src/log4j.properties`. That's the default name for Log4J's configuration and using that name would make life easier - not only the framework knows what is inside, you too!

```
log4j.rootLogger=DEBUG, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%m%n
```

This configuration creates an output channel ("Appender") to console named as `stdout` which prints the message (%m) followed by a line feed (%n) - same as the earlier System.out.println() :-) Oooh kay - but we haven't finished yet. We should deliver the configuration file, too. So we change the buildfile:

```
    ...
    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}" classpathref="classpath"/>
        <copy todir="${classes.dir}">
            <fileset dir="${src.dir}" excludes="**/*.java"/>
        </copy>
    </target>
    ...
```

This copies all resources (as long as they haven't the suffix ".java") to the build directory, so we could start the application from that directory and these files will included into the jar.

Testing the class

In this step we will introduce the usage of the JUnit [3] testframework in combination with Ant. Because Ant has a built-in JUnit 3.8.2 you could start directly using it. Write a test class in `src\HelloWorldTest.java`:

```
public class HelloWorldTest extends junit.framework.TestCase {

    public void testNothing() {
    }

    public void testWillAlwaysFail() {
        fail("An error message");
    }

}
```

Because we dont have real business logic to test, this test class is very small: just show how to start. For further information see the JUnit documentation [3] and the manual of junit task. Now we add a junit instruction to our buildfile:

```
    ...

    <target name="run" depends="jar">
        <java fork="true" classname="${main-class}">
            <classpath>
                <path refid="classpath"/>
                <path id="application" location="${jar.dir}/${ant.project.name}.jar"/>
            </classpath>
        </java>
    </target>

    <target name="junit" depends="jar">
        <junit printsummary="yes">
            <classpath>
                <path refid="classpath"/>
                <path refid="application"/>
            </classpath>

            <batchtest fork="yes">
                <fileset dir="${src.dir}" includes="*Test.java"/>
            </batchtest>
        </junit>
    </target>

    ...
```

We reuse the path to our own jar file as defined in run-target by giving it an ID. The `printsummary=yes` lets us see more detailed information than just a "FAILED" or "PASSED" message. How much tests failed? Some errors? Printsummary lets us know. The classpath is set up to find our classes. To run tests the `batchtest` here is used, so you could easily add more test classes in the future just by naming them `*Test.java`. This is a common naming scheme.

After a `ant junit` you'll get:

```
...
junit:
    [junit] Running HelloWorldTest
    [junit] Tests run: 2, Failures: 1, Errors: 0, Time elapsed: 0,01 sec
    [junit] Test HelloWorldTest FAILED

BUILD SUCCESSFUL
...
```

We can also produce a report. Something that you (and other) could read after closing the shell .... There are two steps: 1. let <junit> log the information and 2. convert these to something readable (browsable).

```
    ...
    <property name="report.dir"  value="${build.dir}/junitreport"/>
    ...
    <target name="junit" depends="jar">
        <mkdir dir="${report.dir}"/>
```

```
        <junit printsummary="yes">
            <classpath>
                <path refid="classpath"/>
                <path refid="application"/>
            </classpath>

            <formatter type="xml"/>

            <batchtest fork="yes" todir="${report.dir}">
                <fileset dir="${src.dir}" includes="*Test.java"/>
            </batchtest>
        </junit>
    </target>

    <target name="junitreport">
        <junitreport todir="${report.dir}">
            <fileset dir="${report.dir}" includes="TEST-*.xml"/>
            <report todir="${report.dir}"/>
        </junitreport>
    </target>
```

Because we would produce a lot of files and these files would be written to the current directory by default, we define a report directory, create it before running the `junit` and redirect the logging to it. The log format is XML so `junitreport` could parse it. In a second target `junitreport` should create a browsable HTML-report for all generated xml-log files in the report directory. Now you can open the ${report.dir}\index.html and see the result (looks something like JavaDoc).
Personally I use two different targets for junit and junitreport. Generating the HTML report needs some time and you dont need the HTML report just for testing, e.g. if you are fixing an error or a integration server is doing a job.

Resources


```
[1] http://www.apache.org/dist/logging/log4j/1.2.13/logging-log4j-1.2.13.zip
[2] http://logging.apache.org/log4j/docs/manual.html
[3] http://www.junit.org/index.htm
```

Tutorial: Writing Tasks

This document provides a step by step tutorial for writing tasks.

Content

Set up the build environment

Ant builds itself, we are using Ant too (why we would write a task if not? :-) therefore we should use Ant for our build.

We choose a directory as root directory. All things will be done here if I say nothing different. I will reference this directory as *root-directory* of our project. In this root-directory we create a text file names *build.xml*. What should Ant do for us?

- compiles my stuff
- make the jar, so that I can deploy it
- clean up everything

So the buildfile contains three targets.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="jar">

    <target name="clean" description="Delete all generated files">
        <delete dir="classes"/>
        <delete file="MyTasks.jar"/>
    </target>

    <target name="compile" description="Compiles the Task">
        <javac srcdir="src" destdir="classes"/>
    </target>

    <target name="jar" description="JARs the Task">
        <jar destfile="MyTask.jar" basedir="classes"/>
    </target>

</project>
```

This buildfile uses often the same value (src, classes, MyTask.jar), so we should rewrite that using `<property>`s. On second there are some handicaps: `<javac>` requires that the destination directory exists; a call of "clean" with a non existing classes directory will fail; "jar" requires the execution of some steps bofore. So the refactored code is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="jar">

    <property name="src.dir" value="src"/>
    <property name="classes.dir" value="classes"/>

    <target name="clean" description="Delete all generated files">
        <delete dir="${classes.dir}" failonerror="false"/>
        <delete file="${ant.project.name}.jar"/>
    </target>

    <target name="compile" description="Compiles the Task">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <target name="jar" description="JARs the Task" depends="compile">
        <jar destfile="${ant.project.name}.jar" basedir="${classes.dir}"/>
    </target>

</project>
```

ant.project.name is one of the build-in properties [1] of Ant.

Write the Task

Now we write the simplest Task - a HelloWorld-Task (what else?). Create a text file *HelloWorld.java* in the src-directory with:

```
public class HelloWorld {
    public void execute() {
        System.out.println("Hello World");
    }
}
```

and we can compile and jar it with `ant` (default target is "jar" and via its *depends*-clause the "compile" is executed before).

Use the Task

But after creating the jar we want to use our new Task. Therefore we need a new target "use". Before we can use our new task we have to declare it with `<taskdef>` [2]. And for easier process we change the default clause:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="use">

    ...

    <target name="use" description="Use the Task" depends="jar">
        <taskdef name="helloworld" classname="HelloWorld" classpath="${ant.project.name}.jar"/>
        <helloworld/>
    </target>

</project>
```

Important is the *classpath*-attribute. Ant searches in its /lib directory for tasks and our task isn't there. So we have to provide the right location.

Now we can type in `ant` and all should work ...

```
Buildfile: build.xml

compile:
```

```
    [mkdir] Created dir: C:\tmp\anttests\MyFirstTask\classes
    [javac] Compiling 1 source file to C:\tmp\anttests\MyFirstTask\classes

jar:
      [jar] Building jar: C:\tmp\anttests\MyFirstTask\MyTask.jar

use:
[helloworld] Hello World

BUILD SUCCESSFUL
Total time: 3 seconds
```

Integration with TaskAdapter

Our class has nothing to do with Ant. It extends no superclass and implements no interface. How does Ant know to integrate? Via name convention: our class provides a method with signature `public void execute()`. This class is wrapped by Ant's `org.apache.tools.ant.TaskAdapter` which is a task and uses reflection for setting a reference to the project and calling the *execute()* method.

Setting a reference to the project? Could be interesting. The Project class gives us some nice abilities: access to Ant's logging facilities getting and setting properties and much more. So we try to use that class:

```
import org.apache.tools.ant.Project;

public class HelloWorld {

    private Project project;

    public void setProject(Project proj) {
        project = proj;
    }

    public void execute() {
        String message = project.getProperty("ant.project.name");
        project.log("Here is project '" + message + "'.", Project.MSG_INFO);
    }
}
```

and the execution with `ant` will show us the expected

```
use:
Here is project 'MyTask'.
```

Deriving from Ant's Task

Ok, that works ... But usually you will extend `org.apache.tools.ant.Task`. That class is integrated in Ant, get's the project-reference, provides documentation fiels, provides easier access to the logging facility and (very useful) gives you the exact location where *in the buildfile* this task instance is used.

Oki-doki - let's us use some of these:

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

public class HelloWorld extends Task {
    public void execute() {
        // use of the reference to Project-instance
        String message = getProject().getProperty("ant.project.name");

        // Task's log method
        log("Here is project '" + message + "'.");

        // where this task is used?
```

```
        log("I am used in: " +  getLocation() );
    }
}
```

which gives us when running

```
use:
[helloworld] Here is project 'MyTask'.
[helloworld] I am used in: C:\tmp\anttests\MyFirstTask\build.xml:23:
```

Attributes

Now we want to specify the text of our message (it seems that we are rewriting the `<echo/>` task :-). First we well do that with an attribute. It is very easy - for each attribute provide a `public void set<attributename>(<type> newValue)` method and Ant will do the rest via reflection.

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

public class HelloWorld extends Task {

    String message;
    public void setMessage(String msg) {
        message = msg;
    }

    public void execute() {
        if (message==null) {
            throw new BuildException("No message set.");
        }
        log(message);
    }

}
```

Oh, what's that in execute()? Throw a *BuildException*? Yes, that's the usual way to show Ant that something important is missed and complete build should fail. The string provided there is written as build-failes-message. Here it's necessary because the log() method can't handle a *null* value as parameter and throws a NullPointerException. (Of course you can initialize the *message* with a default string.)

After that we have to modify our buildfile:

```
    <target name="use" description="Use the Task" depends="jar">
        <taskdef name="helloworld"
                 classname="HelloWorld"
                 classpath="${ant.project.name}.jar"/>
        <helloworld message="Hello World"/>
    </target>
```

That's all.

Some background for working with attributes: Ant supports any of these datatypes as arguments of the set-method:

- elementary data type like *int*, *long*, ...
- its wrapper classes like *java.lang.Integer*, *java.lang.Long*, ...
- *java.lang.String*
- some more classes (e.g. *java.io.File*; see Manual 'Writing Your Own Task' [3])
- Any Java Object parsed from Ant 1.8's Property Helper

Before calling the set-method all properties are resolved. So a `<helloworld message="${msg}"/>` would not set the message string to "${msg}" if there is a property "msg" with a set value.

Nested Text

Maybe you have used the `<echo>` task in a way like `<echo>Hello World</echo>`. For that you have to provide a `public void addText(String text)` method.

```
...
public class HelloWorld extends Task {
    ...
    public void addText(String text) {
        message = text;
    }
    ...
}
```

But here properties are **not** resolved! For resolving properties we have to use Project's `replaceProperties(String propname) : String` method which takes the property name as argument and returns its value (or ${propname} if not set).

Nested Elements

There are several ways for inserting the ability of handling nested elements. See the Manual [4] for other. We use the first way of the three described ways. There are several steps for that:

1. We create a class for collecting all the info the nested element should contain. This class is created by the same rules for attributes and nested elements as for the task (`set<attributename>()` methods).
2. The task holds multiple instances of this class in a list.
3. A factory method instantiates an object, saves the reference in the list and returns it to Ant Core.
4. The execute() method iterates over the list and evaluates its values.

```
import java.util.Vector;
import java.util.Iterator;
...
    public void execute() {
        if (message!=null) log(message);
        for (Iterator it=messages.iterator(); it.hasNext(); ) {      // 4
            Message msg = (Message)it.next();
            log(msg.getMsg());
        }
    }


    Vector messages = new Vector();                                   // 2

    public Message createMessage() {                                  // 3
        Message msg = new Message();
        messages.add(msg);
        return msg;
    }

    public class Message {                                            // 1
        public Message() {}

        String msg;
        public void setMsg(String msg) { this.msg = msg; }
        public String getMsg() { return msg; }
    }
...
```

Then we can use the new nested element. But where is xml-name for that defined? The mapping XML-name : classname is defined in the factory method: `public` *`classname`* `create`*`XML-name`*`()`. Therefore we write in the buildfile

```
        <helloworld>
            <message msg="Nested Element 1"/>
            <message msg="Nested Element 2"/>
        </helloworld>
```

Note that if you choose to use methods 2 or 3, the class that represents the nested element must be declared as

```
static
```

Our task in a little more complex version

For recapitulation now a little refactored buildfile:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="use">

    <property name="src.dir" value="src"/>
    <property name="classes.dir" value="classes"/>

    <target name="clean" description="Delete all generated files">
        <delete dir="${classes.dir}" failonerror="false"/>
        <delete file="${ant.project.name}.jar"/>
    </target>

    <target name="compile" description="Compiles the Task">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <target name="jar" description="JARs the Task" depends="compile">
        <jar destfile="${ant.project.name}.jar" basedir="${classes.dir}"/>
    </target>


    <target name="use.init"
            description="Taskdef the HelloWorld-Task"
            depends="jar">
        <taskdef name="helloworld"
                 classname="HelloWorld"
                 classpath="${ant.project.name}.jar"/>
    </target>


    <target name="use.without"
            description="Use without any"
            depends="use.init">
        <helloworld/>
    </target>

    <target name="use.message"
            description="Use with attribute 'message'"
            depends="use.init">
        <helloworld message="attribute-text"/>
    </target>

    <target name="use.fail"
            description="Use with attribute 'fail'"
            depends="use.init">
        <helloworld fail="true"/>
    </target>

    <target name="use.nestedText"
            description="Use with nested text"
            depends="use.init">
        <helloworld>nested-text</helloworld>
    </target>

    <target name="use.nestedElement"
            description="Use with nested 'message'"
            depends="use.init">
        <helloworld>
            <message msg="Nested Element 1"/>
            <message msg="Nested Element 2"/>
```

```
        </helloworld>
    </target>


    <target name="use"
            description="Try all (w/out use.fail)"
            depends="use.without,use.message,use.nestedText,use.nestedElement"
    />

</project>
```

And the code of the task:

```
import org.apache.tools.ant.Task;
import org.apache.tools.ant.BuildException;
import java.util.Vector;
import java.util.Iterator;

/**
 * The task of the tutorial.
 * Print a message or let the build fail.
 * @since 2003-08-19
 */
public class HelloWorld extends Task {


    /** The message to print. As attribute. */
    String message;
    public void setMessage(String msg) {
        message = msg;
    }

    /** Should the build fail? Defaults to false. As attribute. */
    boolean fail = false;
    public void setFail(boolean b) {
        fail = b;
    }

    /** Support for nested text. */
    public void addText(String text) {
        message = text;
    }


    /** Do the work. */
    public void execute() {
        // handle attribute 'fail'
        if (fail) throw new BuildException("Fail requested.");

        // handle attribute 'message' and nested text
        if (message!=null) log(message);

        // handle nested elements
        for (Iterator it=messages.iterator(); it.hasNext(); ) {
            Message msg = (Message)it.next();
            log(msg.getMsg());
        }
    }


    /** Store nested 'message's. */
    Vector messages = new Vector();

    /** Factory method for creating nested 'message's. */
    public Message createMessage() {
        Message msg = new Message();
        messages.add(msg);
        return msg;
    }

    /** A nested 'message'. */
    public class Message {
        // Bean constructor
        public Message() {}

        /** Message to print. */
        String msg;
        public void setMsg(String msg) { this.msg = msg; }
        public String getMsg() { return msg; }
    }
```

```
}
```

And it works:

```
C:\tmp\anttests\MyFirstTask>ant
Buildfile: build.xml

compile:
    [mkdir] Created dir: C:\tmp\anttests\MyFirstTask\classes
    [javac] Compiling 1 source file to C:\tmp\anttests\MyFirstTask\classes

jar:
      [jar] Building jar: C:\tmp\anttests\MyFirstTask\MyTask.jar

use.init:

use.without:

use.message:
[helloworld] attribute-text

use.nestedText:
[helloworld] nested-text

use.nestedElement:
[helloworld]
[helloworld]
[helloworld]
[helloworld]
[helloworld] Nested Element 1
[helloworld] Nested Element 2

use:

BUILD SUCCESSFUL
Total time: 3 seconds
C:\tmp\anttests\MyFirstTask>ant use.fail
Buildfile: build.xml

compile:

jar:

use.init:

use.fail:

BUILD FAILED
C:\tmp\anttests\MyFirstTask\build.xml:36: Fail requested.

Total time: 1 second
C:\tmp\anttests\MyFirstTask>
```

Next step: test ...

Test the Task

We have written a test already: the use.* tasks in the buildfile. But its difficult to test that automatically. Common (and in Ant) used is JUnit for that. For testing tasks Ant provides a baseclass `org.apache.tools.ant.BuildFileTest`. This class extends `junit.framework.TestCase` and can therefore be integrated into the unit tests. But this class provides some for testing tasks useful methods: initialize Ant, load a buildfile, execute targets, expecting BuildExceptions with a specified text, expect a special text in the output log ...

In Ant it is usual that the testcase has the same name as the task with a prepending *Test*, therefore we will create a file *HelloWorldTest.java*. Because we have a very small project we can put this file into *src* directory (Ant's own testclasses are in /src/testcases/...). Because we have already written our tests for "hand-test" we can use that for automatic tests, too. But there is one little problem we have to solve: all test supporting classes are not part of the binary distribution of Ant. So you can build the special jar file from source distro with target "test-jar" or you can

download a nightly build from [http://gump.covalent.net/jars/latest/ant/ant-testutil.jar [5]](http://gump.covalent.net/jars/latest/ant/ant-testutil.jar).

For executing the test and creating a report we need the optional tasks `<junit>` and `<junitreport>`. So we add to the buildfile:

```
...
<project name="MyTask" basedir="." default="test">
...
    <property name="ant.test.lib" value="ant-testutil.jar"/>
    <property name="report.dir"   value="report"/>
    <property name="junit.out.dir.xml"  value="${report.dir}/junit/xml"/>
    <property name="junit.out.dir.html" value="${report.dir}/junit/html"/>

    <path id="classpath.run">
        <path path="${java.class.path}"/>
        <path location="${ant.project.name}.jar"/>
    </path>

    <path id="classpath.test">
        <path refid="classpath.run"/>
        <path location="${ant.test.lib}"/>
    </path>

    <target name="clean" description="Delete all generated files">
        <delete failonerror="false" includeEmptyDirs="true">
            <fileset dir="." includes="${ant.project.name}.jar"/>
            <fileset dir="${classes.dir}"/>
            <fileset dir="${report.dir}"/>
        </delete>
    </target>

    <target name="compile" description="Compiles the Task">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}" classpath="${ant.test.lib}"/>
    </target>
...
    <target name="junit" description="Runs the unit tests" depends="jar">
        <delete dir="${junit.out.dir.xml}"/>
        <mkdir  dir="${junit.out.dir.xml}"/>
        <junit printsummary="yes" haltonfailure="no">
            <classpath refid="classpath.test"/>
            <formatter type="xml"/>
            <batchtest fork="yes" todir="${junit.out.dir.xml}">
                <fileset dir="${src.dir}" includes="**/*Test.java"/>
            </batchtest>
        </junit>
    </target>

    <target name="junitreport" description="Create a report for the rest result">
        <mkdir dir="${junit.out.dir.html}"/>
        <junitreport todir="${junit.out.dir.html}">
            <fileset dir="${junit.out.dir.xml}">
                <include name="*.xml"/>
            </fileset>
            <report format="frames" todir="${junit.out.dir.html}"/>
        </junitreport>
    </target>

    <target name="test"
            depends="junit,junitreport"
            description="Runs unit tests and creates a report"
    />
...
```

Back to the *src/HelloWorldTest.java*. We create a class extending *BuildFileTest* with String-constructor (JUnit-standard), a *setUp()* method initializing Ant and for each testcase (targets use.*) a *testXX()* method invoking that target.

```
import org.apache.tools.ant.BuildFileTest;

public class HelloWorldTest extends BuildFileTest {

    public HelloWorldTest(String s) {
        super(s);
    }

    public void setUp() {
        // initialize Ant
```

```
        configureProject("build.xml");
    }

    public void testWithout() {
        executeTarget("use.without");
        assertEquals("Message was logged but should not.", getLog(), "");
    }

    public void testMessage() {
        // execute target 'use.nestedText' and expect a message
        // 'attribute-text' in the log
        expectLog("use.message", "attribute-text");
    }

    public void testFail() {
        // execute target 'use.fail' and expect a BuildException
        // with text 'Fail requested.'
        expectBuildException("use.fail", "Fail requested.");
    }

    public void testNestedText() {
        expectLog("use.nestedText", "nested-text");
    }

    public void testNestedElement() {
        executeTarget("use.nestedElement");
        assertLogContaining("Nested Element 1");
        assertLogContaining("Nested Element 2");
    }
}
```

When starting `ant` we'll get a short message to STDOUT and a nice HTML-report.

```
C:\tmp\anttests\MyFirstTask>ant
Buildfile: build.xml

compile:
    [mkdir] Created dir: C:\tmp\anttests\MyFirstTask\classes
    [javac] Compiling 2 source files to C:\tmp\anttests\MyFirstTask\classes

jar:
      [jar] Building jar: C:\tmp\anttests\MyFirstTask\MyTask.jar

junit:
    [mkdir] Created dir: C:\tmp\anttests\MyFirstTask\report\junit\xml
    [junit] Running HelloWorldTest
    [junit] Tests run: 5, Failures: 0, Errors: 0, Time elapsed: 2,334 sec



junitreport:
    [mkdir] Created dir: C:\tmp\anttests\MyFirstTask\report\junit\html
[junitreport] Using Xalan version: Xalan Java 2.4.1
[junitreport] Transform time: 661ms

test:

BUILD SUCCESSFUL
Total time: 7 seconds
C:\tmp\anttests\MyFirstTask>
```

Resources

This tutorial and its resources are available via BugZilla [6]. The ZIP provided there contains

- this tutorial
- the buildfile (last version)
- the source of the task (last version)
- the source of the unit test (last version)
- the ant-testutil.jar (nightly build of 2003-08-18)

- generated classes
- generated jar
- generated reports

The last sources and the buildfile are also available here [7] inside the manual.

Used Links:
[1] http://ant.apache.org/manual/properties.html#built-in-props
[2] http://ant.apache.org/manual/CoreTasks/taskdef.html
[3] http://ant.apache.org/manual/develop.html#set-magic
[4] http://ant.apache.org/manual/develop.html#nested-elements
[5] http://gump.covalent.net/jars/latest/ant/ant-testutil.jar
[6] http://issues.apache.org/bugzilla/show_bug.cgi?id=22570
[7] tutorial-writing-tasks-src.zip

Tutorial: Tasks using Properties, Filesets & Paths

After reading the tutorial about writing tasks [1] this tutorial explains how to get and set properties and how to use nested filesets and paths. Finally it explains how to contribute tasks to Ant.

Content

The goal

The goal is to write a task, which searchs in a path for a file and saves the location of that file in a property.

Build environment

We can use the buildfile from the other tutorial and modify it a little bit. That's the advantage of using properties - we can reuse nearly the whole script. :-)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="FindTask" basedir="." default="test">
    ...
    <target name="use.init" description="Taskdef's the Find-Task" depends="jar">
        <taskdef name="find" classname="Find" classpath="${ant.project.name}.jar"/>
    </target>

    <!-- the other use.* targets are deleted -->
    ...
</project>
```

The buildfile is in the archive tutorial-tasks-filesets-properties.zip [2] in `/build.xml.01-propertyaccess` (future version saved as *.02..., final version as build.xml; same for sources).

Property access

Our first step is to set a property to a value and print the value of that property. So our scenario would be

```
    <find property="test" value="test-value"/>
    <find print="test"/>
```

ok, can be rewritten with the core tasks

```
    <property name="test" value="test-value"/>
    <echo message="${test}"/>
```

but I have to start on known ground :-)

So what to do? Handling three attributes (property, value, print) and an execute method. Because this is only an introduction example I don't do much checking:

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

public class Find extends Task {

    private String property;
    private String value;
    private String print;

    public void setProperty(String property) {
        this.property = property;
    }

    // setter for value and print

    public void execute() {
        if (print != null) {
            String propValue = getProject().getProperty(print);
            log(propValue);
        } else {
            if (property == null) throw new BuildException("property not set");
            if (value    == null) throw new BuildException("value not set");
            getProject().setNewProperty(property, value);
        }
    }
}
```

As said in the other tutorial, the property access is done via Project instance. We get this instance via the public `getProject()` method which we inherit from `Task` (more precise from ProjectComponent). Reading a property is done via `getProperty(`*`propertyname`*`)` (very simple, isn't it?). This property returns the value as String or *null* if not set.
Setting a property is ... not really difficult, but there is more than one setter. You can use the `setProperty()` method which will do the job like expected. But there is a golden rule in Ant: *properties are immutable*. And this method sets the property to the specified value - whether it has a value before that or not. So we use another way. `setNewProperty()` sets the property only if there is no property with that name. Otherwise a message is logged.

(by the way: a short word to ants "namespaces" (don't be confused with xml namespaces: an `<antcall>` creates a new space for property names. All properties from the caller are passed to the callee, but the callee can set its own properties without notice by the caller.)

There are some other setter, too (but I haven't used them, so I can't say something to them, sorry :-)

After putting our two line example from above into a target names `use.simple` we can call that from our testcase:

```
import org.apache.tools.ant.BuildFileTest;

public class FindTest extends BuildFileTest {

    public FindTest(String name) {
        super(name);
    }

    public void setUp() {
        configureProject("build.xml");
    }

    public void testSimple() {
        expectLog("use.simple", "test-value");
    }
}
```

and all works fine.

Using filesets

Ant provides a common way of bundling files: the fileset. Because you are reading this tutorial I think you know them and I don't have to spend more explanations about their usage in buildfiles. Our goal is to search a file in path. And on this step the path is simply a fileset (or more precise: a collection of filesets). So our usage would be

```
    <find file="ant.jar" location="location.ant-jar">
        <fileset dir="${ant.home}" includes="**/*.jar"/>
    </find>
```

What do we need? A task with two attributes (file, location) and nested filesets. Because we had attribute handling already explained in the example above and the handling of nested elements is described in the other tutorial the code should be very easy:

```
public class Find extends Task {

    private String file;
    private String location;
    private Vector filesets = new Vector();

    public void setFile(String file) {
        this.file = file;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public void addFileset(FileSet fileset) {
        filesets.add(fileset);
    }

    public void execute() {
    }
}
```

Ok - that task wouldn't do very much, but we can use it in the described manner without failure. On next step we have to implement the execute method. And before that we will implement the appropriate testcases (TDD - test driven development).

In the other tutorial we have reused the already written targets of our buildfile. Now we will configure most of the testcases via java code (sometimes it's much easier to write a target than doing it via java coding). What can be tested?

- not valid configured task (missing file, missing location, missing fileset)
- don't find a present file
- behaviour if file can't be found

Maybe you find some more testcases. But this is enough for now.
For each of these points we create a `testXX` method.

```
public class FindTest extends BuildFileTest {

    ... // constructor, setUp as above

    public void testMissingFile() {
        Find find = new Find();
        try {
            find.execute();
            fail("No 'no-file'-exception thrown.");
        } catch (Exception e) {
            // exception expected
            String expected = "file not set";
            assertEquals("Wrong exception message.", expected, e.getMessage());
        }
    }
```

```
    public void testMissingLocation() {
        Find find = new Find();
        find.setFile("ant.jar");
        try {
            find.execute();
            fail("No 'no-location'-exception thrown.");
        } catch (Exception e) {
            ... // similar to testMissingFile()
        }
    }

    public void testMissingFileset() {
        Find find = new Find();
        find.setFile("ant.jar");
        find.setLocation("location.ant-jar");
        try {
            find.execute();
            fail("No 'no-fileset'-exception thrown.");
        } catch (Exception e) {
            ... // similar to testMissingFile()
        }
    }

    public void testFileNotPresent() {
        executeTarget("testFileNotPresent");
        String result = getProject().getProperty("location.ant-jar");
        assertNull("Property set to wrong value.", result);
    }

    public void testFilePresent() {
        executeTarget("testFilePresent");
        String result = getProject().getProperty("location.ant-jar");
        assertNotNull("Property not set.", result);
        assertTrue("Wrong file found.", result.endsWith("ant.jar"));
    }
}
```

If we run this test class all test cases (except *testFileNotPresent*) fail. Now we can implement our task, so that these test cases will pass.

```
    protected void validate() {
        if (file==null) throw new BuildException("file not set");
        if (location==null) throw new BuildException("location not set");
        if (filesets.size()<1) throw new BuildException("fileset not set");
    }

    public void execute() {
        validate();                                                          // 1
        String foundLocation = null;
        for(Iterator itFSets = filesets.iterator(); itFSets.hasNext(); ) {   // 2
            FileSet fs = (FileSet)itFSets.next();
            DirectoryScanner ds = fs.getDirectoryScanner(getProject());      // 3
            String[] includedFiles = ds.getIncludedFiles();
            for(int i=0; i<includedFiles.length; i++) {
                String filename = includedFiles[i].replace('\\','/');        // 4
                filename = filename.substring(filename.lastIndexOf("/")+1);
                if (foundLocation==null && file.equals(filename)) {
                    File base  = ds.getBasedir();                            // 5
                    File found = new File(base, includedFiles[i]);
                    foundLocation = found.getAbsolutePath();
                }
            }
        }
        if (foundLocation!=null)                                             // 6
            getProject().setNewProperty(location, foundLocation);
    }
```

On //**1** we check the prerequisites for our task. Doing that in a `validate`-method is a common way, because we separate the prerequisites from the real work. On //**2** we iterate over all nested filesets. If we don't want to handle multiple filesets, the `addFileset()` method has to reject the further calls. We can get the result of a fileset via its DirectoryScanner like done in //**3**. After that we create a plattform independend String representation of the file path (//**4**, can be done in other ways of course). We have to do the `replace()`, because we work with a simple string comparison. Ant itself is platform independant and can therefore run on filesystems with slash (/, e.g. Linux) or backslash (\, e.g. Windows) as path separator. Therefore we have to unify that. If we found our file we create an absolute path representation on //**5**, so that we can use that information without knowing the basedir. (This is very

important on use with multiple filesets, because they can have different basedirs and the return value of the directory scanner is relative to its basedir.) Finally we store the location of the file as property, if we had found one (//**6**).

Ok, much more easier in this simple case would be to add the *file* as additional *include* element to all filesets. But I wanted to show how to handle complex situations whithout being complex :-)

The test case uses the ant property *ant.home* as reference. This property is set by the `Launcher` class which starts ant. We can use that property in our buildfiles as a build-in property [3]. But if we create a new ant environment we have to set that value for our own. And we use the `<junit>` task in fork-mode. Therefore we have do modify our buildfile:

```
    <target name="junit" description="Runs the unit tests" depends="jar">
        <delete dir="${junit.out.dir.xml}"/>
        <mkdir  dir="${junit.out.dir.xml}"/>
        <junit printsummary="yes" haltonfailure="no">
            <classpath refid="classpath.test"/>
            <sysproperty key="ant.home" value="${ant.home}"/>
            <formatter type="xml"/>
            <batchtest fork="yes" todir="${junit.out.dir.xml}">
                <fileset dir="${src.dir}" includes="**/*Test.java"/>
            </batchtest>
        </junit>
    </target>
```

Using nested paths

A task providing support for filesets is a very comfortable one. But there is another possibility of bundling files: the `<path>`. Fileset are easy if the files are all under a common base directory. But if this is not the case you have a problem. Another disadvantage is its speed: if you have only a few files in a huge directory structure, why not use a `<filelist>` instead? `<path>`s combines these datatypes in that way that a path contains other paths, filesets, dirsets and filelists. This is why Ant-Contribs [4] `<foreach>` task is modified to support paths instead of filesets. So we want that, too.

Changing from fileset to path support is very easy:

```
Change java code from:
    private Vector filesets = new Vector();
    public void addFileset(FileSet fileset) {
        filesets.add(fileset);
    }
to:
    private Vector paths = new Vector();                        *1
    public void addPath(Path path) {                            *2
        paths.add(path);
    }
and build file from:
    <find file="ant.jar" location="location.ant-jar">
        <fileset dir="${ant.home}" includes="**/*.jar"/>
    </find>
to:
    <find file="ant.jar" location="location.ant-jar">
        <path>                                                  *3
            <fileset dir="${ant.home}" includes="**/*.jar"/>
        </path>
    </find>
```

On ***1** we rename only the vector. Itï¿½s just for better reading the source. On ***2** we have to provide the right method: an add*Name*(*Type* t). Therefore replace the fileset with path here. Finally we have to modify our buildfile on ***3** because our task doesnï¿½t support nested filesets any longer. So we wrap the fileset inside a path.

And now we modify the testcase. Oh, not very much to do :-) Renaming the `testMissingFileset()` (not really a *must-be* but better itï¿½s named like the think it does) and update the *expected*-String in that method (now a `path not set` message is expected). The more complex test cases base on the buildscript. So the targets `testFileNotPresent` and `testFilePresent` have to be modified in the manner described above.

The test are finished. Now we have to adapt the task implementation. The easiest modification is in the `validate()` method where we change le last line to `if (paths.size()<1) throw new BuildException("path not set");`. In the `execute()` method we have a little more work. ... mmmh ... in reality it's lesser work, because the Path class does the whole DirectoryScanner-handling and creating-absolute-paths stuff for us. So the execute method is just:

```
    public void execute() {
        validate();
        String foundLocation = null;
        for(Iterator itPaths = paths.iterator(); itPaths.hasNext(); ) {
            Path path = (Path)itPaths.next();                                  // 1
            String[] includedFiles = path.list();                              // 2
            for(int i=0; i<includedFiles.length; i++) {
                String filename = includedFiles[i].replace('\\','/');
                filename = filename.substring(filename.lastIndexOf("/")+1);
                if (foundLocation==null && file.equals(filename)) {
                    foundLocation = includedFiles[i];                          // 3
                }
            }
        }
        if (foundLocation!=null)
            getProject().setNewProperty(location, foundLocation);
    }
```

Of course we have to do the typecase to Path on //**1**. On //**2** and //**3** we see that the Path class does the work for us: no DirectoryScanner (was at 2) and no creating of the absolute path (was at 3).

Returning a list

So far so good. But could a file be on more than one place in the path? - Of course.
And would it be good to get all of them? - It depends on ...

In this section we will extend that task to support returning a list of all files. Lists as property values are not supported by Ant natively. So we have to see how other tasks use lists. The most famous task using lists is Ant-Contribs `<foreach>`. All list elements are concatenated and separated with a customizable separator (default ',').

So we do the following:

```
    <find ... delimiter=""/> ... </find>
```

If the delimiter is set we will return all found files as list with that delimiter.

Therefore we have to

- provide a new attribute
- collect more than the first file
- delete duplicates
- create the list if necessary
- return that list

So we add as testcase:

```
in the buildfile:
    <target name="test.init">
        <mkdir dir="test1/dir11/dir111"/>                                  *1
        <mkdir dir="test1/dir11/dir112"/>
        ...
        <touch file="test1/dir11/dir111/test"/>
        <touch file="test1/dir11/dir111/not"/>
        ...
        <touch file="test1/dir13/dir131/not2"/>
        <touch file="test1/dir13/dir132/test"/>
```

```
        <touch file="test1/dir13/dir132/not"/>
        <touch file="test1/dir13/dir132/not2"/>
        <mkdir dir="test2"/>
        <copy todir="test2">                                                    *2
            <fileset dir="test1"/>
        </copy>
    </target>

    <target name="testMultipleFiles" depends="use.init,test.init">    *3
        <find file="test" location="location.test" delimiter=";">
            <path>
                <fileset dir="test1"/>
                <fileset dir="test2"/>
            </path>
        </find>
        <delete>                                                                *4
            <fileset dir="test1"/>
            <fileset dir="test2"/>
        </delete>
    </target>
```

in the test class:

```
    public void testMultipleFiles() {
        executeTarget("testMultipleFiles");
        String result = getProject().getProperty("location.test");
        assertNotNull("Property not set.", result);
        assertTrue("Only one file found.", result.indexOf(";") > -1);
    }
```

Now we need a directory structure where we CAN find files with the same name in different directories. Because we can't sure to have one we create one on ***1** and ***2**. And of course we clean up that on ***4**. The creation can be done inside our test target or in a separate one, which will be better for reuse later (***3**).

The task implementation is modified as followed:

```
    private Vector foundFiles = new Vector();
    ...
    private String delimiter = null;
    ...
    public void setDelimiter(String delim) {
        delimiter = delim;
    }
    ...
    public void execute() {
        validate();
        // find all files
        for(Iterator itPaths = paths.iterator(); itPaths.hasNext(); ) {
            Path path = (Path)itPaths.next();
            String[] includedFiles = path.list();
            for(int i=0; i<includedFiles.length; i++) {
                String filename = includedFiles[i].replace('\\','/');
                filename = filename.substring(filename.lastIndexOf("/")+1);
                if (file.equals(filename) && !foundFiles.contains(includedFiles[i])) {   // 1
                    foundFiles.add(includedFiles[i]);
                }
            }
        }

        // create the return value (list/single)
        String rv = null;
        if (foundFiles.size() > 0) {                                            // 2
            if (delimiter==null) {
                // only the first
                rv = (String)foundFiles.elementAt(0);
            } else {
                // create list
                StringBuffer list = new StringBuffer();
                for(Iterator it=foundFiles.iterator(); it.hasNext(); ) {    // 3
                    list.append(it.next());
                    if (it.hasNext()) list.append(delimiter);               // 4
                }
                rv = list.toString();
            }
        }

        // create the property
        if (rv!=null)
```

```
            getProject().setNewProperty(location, rv);
    }
```

The algorithm does: finding all files, creating the return value depending on the users wish, returning the value as property. On //**1** we eliminates the duplicates. //**2** ensures that we create the return value only if we have found one file. On //**3** we iterate over all found files and //**4** ensures that the last entry has no trailing delimiter.

Ok, first searching for all files and then returning only the first one ... You can tune the performance of your own :-)

Documentation

A task is useless if the only who is able to code the buildfile is the task developer (and he only the next few weeks :-). So documentation is also very important. In which form you do that depends on your favourite. But inside Ant there is a common format and it has advantages if you use that: all task users know that form, this form is requested if you decide to contribute your task. So we will doc our task in that form.

If you have a look at the manual page of the Java task [5] you will see that it:

- is plain html
- starts with the name
- has sections: description, parameters, nested elements, (maybe return codes) and (most important :-) examples
- parameters are listed in a table with columns for attribute name, its description and whether it's required (if you add a feature after an Ant release, provide a `since Ant xx` statement when it's introduced)
- describe the nested elements (since-statement if necessary)
- provide one or more useful examples; first code, then description.

As a template we have:

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<title>Taskname Task</title>
</head>

<body>

<h2><a name="taskname">Taskname</a></h2>
<h3>Description</h3>
<p> Describe the task.</p>

<h3>Parameters</h3>
<table border="1" cellpadding="2" cellspacing="0">
  <tr>
    <td valign="top"><b>Attribute</b></td>
    <td valign="top"><b>Description</b></td>
    <td align="center" valign="top"><b>Required</b></td>
  </tr>

  do this html row for each attribute (including inherited attributes)
  <tr>
    <td valign="top">classname</td>
    <td valign="top">the Java class to execute.</td>
    <td align="center" valign="top">Either jar or classname</td>
  </tr>

</table>

<h3>Parameters specified as nested elements</h3>

Describe each nested element (including inherited)
<h4>your nested element</h4>
<p>description</p>
<p><em>since Ant 1.6</em>.</p>
```

```
<h3>Examples</h3>
<pre>
    A code sample; don't forget to escape the < of the tags with &lt;
</pre>
What should that example do?

</body>
</html>
```

Here is an example documentation page for our task:

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<title>Find Task</title>
</head>

<body>

<h2><a name="find">Find</a></h2>
<h3>Description</h3>
<p>Searchs in a given path for a file and returns the absolute to it as property.
If delimiter is set this task returns all found locations.</p>

<h3>Parameters</h3>
<table border="1" cellpadding="2" cellspacing="0">
  <tr>
    <td valign="top"><b>Attribute</b></td>
    <td valign="top"><b>Description</b></td>
    <td align="center" valign="top"><b>Required</b></td>
  </tr>
  <tr>
    <td valign="top">file</td>
    <td valign="top">The name of the file to search.</td>
    <td align="center" valign="top">yes</td>
  </tr>
  <tr>
    <td valign="top">location</td>
    <td valign="top">The name of the property where to store the location</td>
    <td align="center" valign="top">yes</td>
  </tr>
  <tr>
    <td valign="top">delimiter</td>
    <td valign="top">A delimiter to use when returning the list</td>
    <td align="center" valign="top">only if the list is required</td>
  </tr>
</table>

<h3>Parameters specified as nested elements</h3>

<h4>path</h4>
<p>The path where to search the file.</p>

<h3>Examples</h3>
<pre>
    <find file="ant.jar" location="loc">
        <path>
            <fileset dir="${ant.home}"/>
        <path>
    </find>
</pre>
Searches in Ants home directory for a file <i>ant.jar</i> and stores its location in
property <i>loc</i> (should be ANT_HOME/bin/ant.jar).

<pre>
    <find file="ant.jar" location="loc" delimiter=";">
        <path>
            <fileset dir="C:/"/>
        <path>
    </find>
    <echo>ant.jar found in: ${loc}</echo>
</pre>
Searches in Windows C: drive for all <i>ant.jar</i> and stores their locations in
property <i>loc</i> delimited with <i>';'</i>. (should need a long time :-)
After that it prints out the result (e.g. C:/ant-1.5.4/bin/ant.jar;C:/ant-1.6/bin/ant.jar).

</body>
</html>
```

Contribute the new task

If we decide to contribute our task, we should do some things:

- is our task welcome? :-) Simply ask on the user list
- is the right package used?
- does the code conform to the styleguide?
- do all tests pass?
- does the code compile on JDK 1.2 (and passes all tests there)?
- code under Apache license
- create a patch file
- publishing that patch file

The Ant Task Guidelines [6] support additional information on that.

Now we will check the "Checklist before submitting a new task" described in that guideline.

- Java file begins with Apache license statement. ***must do that***
- Task does not depend on GPL or LGPL code. ***ok***
- Source code complies with style guidelines ***have to check (checkstyle)***
- Code compiles and runs on Java1.2 ***have to try***
- Member variables are private, and provide public accessor methods if access is actually needed. ***have to check (checkstyle)***
- *Maybe* Task has failonerror attribute to control failure behaviour ***hasn't***
- New test cases written and succeed ***passed on JDK 1.4, have to try on JDK 1.2***
- Documentation page written ***ok***
- Example task declarations in the documentation tested. ***ok (used in tests)***
- Patch files generated using cvs diff -u ***to do***
- patch files include a patch to defaults.properties to register the tasks ***to do***
- patch files include a patch to coretasklist.html or optionaltasklist.html to link to the new task page ***to do***
- Message to dev contains [SUBMIT] and task name in subject ***to do***
- Message body contains a rationale for the task ***to do***
- Message attachments contain the required files -source, documentation, test and patches zipped up to escape the HTML filter. ***to do***

Package / Directories

This task does not depend on any external library. Therefore we can use this as a core task. This task contains only one class. So we can use the standard package for core tasks: `org.apache.tools.ant.taskdefs`. Implementations are in the directory `src/main`, tests in `src/testcases` and buildfiles for tests in `src/etc/testcases`.

Now we integrate our work into Ants distribution. So first we do an update of our cvs tree. If not done yet, you have to checkout the ant module from Apaches cvs server as described in Access the Source Tree (AnonCVS) [7] (password is *anoncvs*):

```
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login                  //1
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic checkout ant           //2
```

If you have a local copy of Ants sources just do an update

```
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login
cd ant                                                                         //3
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic update                 //4
```

We use the *-d* flag on **//1** to specifiy the cvs directory. You can specify the environment variable CVSROOT with that value and after that you havenï¿½t to use that flag any more. On **//2** we get the whole cvs tree of ant. (Sorry, but that uses a lot of time ... 10 up to 30 minutes are not unusual ... but this has to be done only once :-). A cvs update doesn't use a modulename but you have to be inside the directory. Therefore we go into that on **//3** and do the update on **//4**.

Now we will build our Ant distribution and do a test. So we can see if there are any tests failing on our machine. (We can ignore these failing tests on later steps; windows syntax used here- translate to xNIX if needed):

```
ANTHOME> build                                                  // 1
ANTHOME> set ANT_HOME=%CD%\dist                                 // 2
ANTHOME> ant test -Dtest.haltonfailure=false                    // 3
```

First we have to build our Ant distribution (**//1**). On **//2** we set the ANT_HOME environment variable to the directory where the new created distribution is stored (%CD% is expanded to the current directory on Windows 2000 and XP, on 9x and NT write it out). On **//3** we let Ant do all the tests (which enforced a compile of all tests) without stopping on first failure.

Next we apply our work onto Ants sources. Because we haven't modified any, this is a relative simple step. *(Because I have a local copy of Ant and usually contribute my work, I work on the local copy just from the beginning. The advantage: this step isn't necessary and saves a lot of work if you modify existing source :-)*.

- move the Find.java to ANTHOME/src/main/org/apache/tools/ant/taskdefs/Find.java
- move the FindTest.java to ANTHOME/src/testcases/org/apache/tools/ant/taskdefs/FindTest.java
- move the build.xml to ANTHOME/src/etc/testcases/taskdefs/**find.xml** (!!! renamed !!!)
- add a `package org.apache.tools.ant.taskdefs;` at the beginning of the two java files
- delete all stuff from find.xml keeping the targets "testFileNotPresent", "testFilePresent", "test.init" and "testMultipleFiles"
- delete the dependency to "use.init" in the find.xml
- in FindTest.java change the line `configureProject("build.xml");` to `configureProject("src/etc/testcases/taskdefs/find.xml");`
- move the find.html to ANTHOME/docs/manual/CoreTasks/find.html
- add a `<a href="CoreTasks/find.html">Find</a><br>` in the ANTHOME/docs/manual/coretasklist.html

Now our modifications are done and we will retest it:

```
ANTHOME> build
ANTHOME> ant run-single-test                                    // 1
             -Dtestcase=org.apache.tools.ant.taskdefs.FindTest  // 2
             -Dtest.haltonfailure=false
```

Because we only want to test our new class, we use the target for single tests, specify the test to use and configure not to halt on the first failure - we want to see all failures of our own test (**//1 + 2**).

And ... oh, all tests fail: *Ant could not find the task or a class this task relies upon.*

Ok: in the earlier steps we told Ant to use the Find class for the `<find>` task (remember the `<taskdef>` statement in the "use.init" target). But now we want to introduce that task as a core task. And nobody wants to taskdef the javac, echo, ... So what to do? The answer is the src/main/.../taskdefs/default.properties. Here is the mapping between taskname and implementing class done. So we add a `find=org.apache.tools.ant.taskdefs.Find` as the last core task (just before the `# optional tasks` line). Now a second try:

```
ANTHOME> build                                                  // 1
ANTHOME> ant run-single-test
             -Dtestcase=org.apache.tools.ant.taskdefs.FindTest
             -Dtest.haltonfailure=false
```

We have to rebuild (**//1**) Ant because the test look in the %ANT_HOME%\lib\ant.jar (more precise: on the classpath) for the properties file. And we have only modified it in the source path. So we have to rebuild that jar. But now all tests pass and we check whether our class breaks some other tests.

```
ANTHOME> ant test -Dtest.haltonfailure=false
```

Because there are a lot of tests this step requires a little bit of time. So use the *run-single-test* during development and do the *test* only at the end (maybe sometimes during development too). We use the *-Dtest.haltonfailure=false* here because there could be other tests fail and we have to look into them.

This test run should show us two things: our test will run and the number of failing tests is the same as directly after the cvs update (without our modifications).

Apache license statement

Simply copy the license text from one the other source from the Ant source tree.

Test on JDK 1.2

Until version 1.5 Ant must be able to run on a JDK 1.1. With version 1.6 this is not a requisite any more. But JDK 1.2 is a must-to-work-with. So we have to test that. You can download older JDKs from Sun [8].

Clean the ANT_HOME variable, delete the *build*, *bootstrap* and *dist* directory and point JAVA_HOME to the JDK 1.2 home directory. Then do the `build`, set ANT_HOME and run `ant test` (like above).

Our test should pass.

Checkstyle

There are many things we have to ensure. Indentation with 4 spaces, blanks here and there, ... (all described in the Ant Task Guidelines [6] which includes the Sun code style [9]). Because there are so many things we would be happy to have a tool for do the checks. There is one: checkstyle. Checkstyle is available at Sourceforge [10] and Ant provides with the `check.xml` a buildfile which will do the job for us.

Download it and put the checkstyle-*-all.jar into your %USERPROFILE%\.ant\lib directory. All jar's stored there are available to Ant so you haven't to add it to you %ANT_HOME%\lib directory (this feature was added with Ant 1.6).

So we will run the tests with

```
ANTHOME> ant -f check.xml checkstyle htmlreport
```

I prefer the HTML report because there are lots of messages and we can navigate faster. Open the ANTHOME/build/reports/checkstyle/html/index.html and navigate to the Find.java. Now we see that there are some errors: missing whitespaces, unused imports, missing javadocs. So we have to do that.

Hint: start at the **buttom** of the file so the line numbers in the report will keep up to date and you will find the next error place much more easier without redoing the checkstyle.

After cleaning up the code according to the messages we delete the reports directory and do a second checkstyle run. Now our task isn't listed. That's fine :-)

Publish the task

Finally we publish that archive. As described in the Ant Task Guidelines [7] we can post it on the developer mailinglist or we create a BugZilla entry. For both we need some information:

subject	*short description*	Task for finding files in a path

body	*more details about the path*	This new task looks inside a nested `<path/>` for occurrences of a file and stores all locations as a property. See the included manual for details.
attachements	*all files needed to apply the path*	Archive containing a patch with the new and modified resources

Sending an email with these information is very easy and I think I haven't to show that. The other way - BugZilla - is slightly more difficult. But it has the advantage that entries will not be forgotten (once per week a report is generated). So I will show this way.

You must have a BugZilla account for that. So open the BugZilla Main Page [11] and follow the link Open a new Bugzilla account [12] and the steps described there if you haven't one.

1. From the BugZilla main page choose Enter a new bug report [13]
2. Choose "Ant" as product
3. Version is the last "Alpha (nightly)" (at this time 1.7)
4. Component is "Core tasks"
5. Plattform and Severity are ok with "Other" and "Normal"
6. Initial State is ok with "New"
7. Same with the empy "Assigned to"
8. It is not required to add yourself as CC, because you are the reporter and therefore will be informed on changes
9. URL: no url required
10. Summary: add the *subject* from the table
11. Description: add the *body* from the table
12. Then press "Commit"
13. After redirecting to the new created bug entry click "Create a New Attachment"
14. Enter the path to your local path file into "File" or choose it via the "File"'s button.
15. Enter a short description into "Description", so that you could guess, what the path file includes. Here we could add "Initial Patch".
16. The "Content Type" is "auto-detect". You could use the "patch" type, if you only provide a single path file, but we want do upload more that one, included in our patch.zip.
17. Then press "Commit"

Now the new task is uploaded into the bug database.

Resources

[1] tutorial-writing-tasks.html
[2] tutorial-tasks-filesets-properties.zip
[3] properties.html#built-in-props
[4] http://ant-contrib.sourceforge.net/
[5] CoreTasks/java.html
[6] http://ant.apache.org/ant_task_guidelines.html
[7] http://ant.apache.org/cvs.html
[8] http://java.sun.com/products/archive/index.html
[9] http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
[10] http://checkstyle.sourceforge.net/
[11] http://issues.apache.org/bugzilla/
[12] http://issues.apache.org/bugzilla/createaccount.cgi
[13] http://issues.apache.org/bugzilla/enter_bug.cgi

Javac

Description

Compiles a Java source tree.

The source and destination directory will be recursively scanned for Java source files to compile. Only Java files that have no corresponding `.class` file or where the class file is older than the `.java` file will be compiled.

Note: Ant uses only the names of the source and class files to find the classes that need a rebuild. It will not scan the source and therefore will have no knowledge about nested classes, classes that are named different from the source file, and so on. See the `<depend>` task for dependency checking based on other than just existence/modification times.

When the source files are part of a package, the directory structure of the source tree should follow the package hierarchy.

It is possible to refine the set of files that are being compiled. This can be done with the `includes`, `includesfile`, `excludes`, and `excludesfile` attributes. With the `includes` or `includesfile` attribute, you specify the files you want to have included. The `exclude` or `excludesfile` attribute is used to specify the files you want to have excluded. In both cases, the list of files can be specified by either the filename, relative to the directory(s) specified in the `srcdir` attribute or nested `<src>` element(s), or by using wildcard patterns. See the section on directory-based tasks, for information on how the inclusion/exclusion of files works, and how to write wildcard patterns.

It is possible to use different compilers. This can be specified by either setting the global `build.compiler` property, which will affect all `<javac>` tasks throughout the build, by setting the `compiler` attribute, specific to the current `<javac>` task or by using a nested element of any typedeffed or componentdeffed type that implements `org.apache.tools.ant.taskdefs.compilers.CompilerAdapter`. Valid values for either the `build.compiler` property or the `compiler` attribute are:

- `classic` (the standard compiler of JDK 1.1/1.2) – `javac1.1` and `javac1.2` can be used as aliases.
- `modern` (the standard compiler of JDK 1.3/1.4/1.5/1.6) – `javac1.3` and `javac1.4` and `javac1.5` and `javac1.6` can be used as aliases.
- `jikes` (the Jikes compiler).
- `jvc` (the Command-Line Compiler from Microsoft's SDK for Java / Visual J++) – `microsoft` can be used as an alias.
- `kjc` (the kopi compiler).
- `gcj` (the gcj compiler from gcc).
- `sj` (Symantec java compiler) – `symantec` can be used as an alias.
- `extJavac` (run either modern or classic in a JVM of its own).

The default is `javac1.x` with `x` depending on the JDK version you use while you are running Ant. If you wish to use a different compiler interface than those supplied, you can write a class that implements the CompilerAdapter interface (`package org.apache.tools.ant.taskdefs.compilers`). Supply the full classname in the `build.compiler` property or the `compiler` attribute.

The fork attribute overrides the `build.compiler` property or `compiler` attribute setting and expects a JDK1.1 or higher to be set in `JAVA_HOME`.

You can also use the `compiler` attribute to tell Ant which JDK version it shall assume when it puts together the command line switches - even if you set `fork="true"`. This is useful if you want to run the compiler of JDK 1.1 while you current JDK is 1.2+. If you use `compiler="javac1.1"` and (for example) `depend="true"` Ant will use the command line switch `-depend` instead of `-Xdepend`.

This task will drop all entries that point to non-existent files/directories from the classpath it passes to the compiler.

The working directory for a forked executable (if any) is the project's base directory.

Windows Note: When the modern compiler is used in unforked mode on Windows, it locks up the files present in the classpath of the `<javac>` task, and does not release them. The side effect of this is that you will not be able to delete or move those files later on in the build. The workaround is to fork when invoking the compiler.

Parameters

Attribute	Description	Required
srcdir	Location of the java files. (See the note below.)	Yes, unless nested `<src>` elements are present.
destdir	Location to store the class files.	No
includes	Comma- or space-separated list of files (may be specified using wildcard patterns) that must be included; all `.java` files are included when omitted.	No
includesfile	The name of a file that contains a list of files to include (may be specified using wildcard patterns).	No
excludes	Comma- or space-separated list of files (may be specified using wildcard patterns) that must be excluded; no files (except default excludes) are excluded when omitted.	No
excludesfile	The name of a file that contains a list of files to exclude (may be specified using wildcard patterns).	No
classpath	The classpath to use.	No
sourcepath	The sourcepath to use; defaults to the value of the srcdir attribute (or nested `<src>` elements). To suppress the sourcepath switch, use `sourcepath=""`.	No
bootclasspath	Location of bootstrap class files. (See below for using the -X and -J-X parameters for specifing the bootstrap classpath).	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No
sourcepathref	The sourcepath to use, given as a reference to a path defined elsewhere.	No
bootclasspathref	Location of bootstrap class files, given as a reference to a path defined elsewhere.	No
extdirs	Location of installed extensions.	No
encoding	Encoding of source files. (Note: gcj doesn't support this option yet.)	No
nowarn	Indicates whether the `-nowarn` switch should be passed to the compiler; defaults to `off`.	No
debug	Indicates whether source should be compiled with debug information; defaults to `off`. If set to `off`, `-g:none` will be passed on the command line for compilers that support it (for other compilers, no command line argument will be used). If set to `true`, the value of the `debuglevel` attribute determines the command line argument.	No
debuglevel	Keyword list to be appended to the `-g` command-line switch. This will be ignored by all implementations except `modern`, `classic(ver >= 1.2)` and `jikes`. Legal values are `none` or a comma-separated list of the following keywords: `lines`, `vars`, and `source`. If `debuglevel` is not specified, by	No

	default, nothing will be appended to `-g`. If `debug` is not turned on, this attribute will be ignored.	
optimize	Indicates whether source should be compiled with optimization; defaults to `off`. **Note** that this flag is just ignored by Sun's `javac` starting with JDK 1.3 (since compile-time optimization is unnecessary).	No
deprecation	Indicates whether source should be compiled with deprecation information; defaults to `off`.	No
target	Generate class files for specific VM version (e.g., `1.1` or `1.2`). **Note that the default value depends on the JVM that is running Ant. In particular, if you use JDK 1.4+ the generated classes will not be usable for a 1.1 Java VM unless you explicitly set this attribute to the value 1.1 (which is the default value for JDK 1.1 to 1.3). We highly recommend to always specify this attribute.** A default value for this attribute can be provided using the magic `ant.build.javac.target` property.	No
verbose	Asks the compiler for verbose output; defaults to `no`.	No
depend	Enables dependency-tracking for compilers that support this (`jikes` and `classic`).	No
includeAntRuntime	Whether to include the Ant run-time libraries in the classpath; defaults to `yes`, unless `build.sysclasspath` is set. *It is usually best to set this to false* so the script's behavior is not sensitive to the environment in which it is run.	No
includeJavaRuntime	Whether to include the default run-time libraries from the executing VM in the classpath; defaults to `no`. **Note:** In some setups the run-time libraries may be part of the "Ant run-time libraries" so you may need to explicitly set includeAntRuntime to false to ensure that the Java run-time libraries are not included.	No
fork	Whether to execute `javac` using the JDK compiler externally; defaults to `no`.	No
executable	Complete path to the `javac` executable to use in case of `fork="yes"`. Defaults to the compiler of the Java version that is currently running Ant. Ignored if `fork="no"`. Since Ant 1.6 this attribute can also be used to specify the path to the executable when using jikes, jvc, gcj or sj.	No
memoryInitialSize	The initial size of the memory for the underlying VM, if `javac` is run externally; ignored otherwise. Defaults to the standard VM memory setting. (Examples: `83886080`, `81920k`, or `80m`)	No
memoryMaximumSize	The maximum size of the memory for the underlying VM, if `javac` is run externally; ignored otherwise. Defaults to the standard VM memory setting. (Examples: `83886080`, `81920k`, or `80m`)	No
failonerror	Indicates whether compilation errors will fail the build; defaults to `true`.	No
errorProperty	The property to set (to the value "true") if compilation fails. *Since Ant 1.7.1*.	No
source	Value of the `-source` command-line switch; will be ignored by all implementations prior to `javac1.4` (or `modern` when Ant is not running in a 1.3 VM), `gcj` and `jikes`. If you use this attribute together with `gcj` or `jikes`, you must make sure that your version supports the `-source` (or `-fsource` for gcj) switch. By default, no `-source` argument will be used at all. **Note that the default value depends on the JVM that is running Ant. We highly recommend to always specify this attribute.** A default value for this attribute can be provided using the magic	No

	ant.build.javac.source property.	
compiler	The compiler implementation to use. If this attribute is not set, the value of the build.compiler property, if set, will be used. Otherwise, the default compiler for the current VM will be used. (See the above list of valid compilers.)	No
listfiles	Indicates whether the source files to be compiled will be listed; defaults to no.	No
tempdir	Where Ant should place temporary files. This is only used if the task is forked and the command line args length exceeds 4k. *Since Ant 1.6.*	No; default is *java.io.tmpdir*.
updatedProperty	The property to set (to the value "true") if compilation has taken place and has been successful. *Since Ant 1.7.1.*	No
includeDestClasses	This attribute controls whether to include the destination classes directory in the classpath given to the compiler. The default value of this is "true" and this means that previously compiled classes are on the classpath for the compiler. This means that "greedy" compilers will not recompile dependant classes that are already compiled. In general this is a good thing as it stops the compiler for doing unnecessary work. However, for some edge cases, involving generics, the javac compiler needs to compile the dependant classes to get the generics information. One example is documented in the bug report: Bug 40776 - a problem compiling a Java 5 project with generics. Setting the attribute to "false" will cause the compiler to recompile dependent classes. *Since Ant 1.7.1.*	No - default is "true"

Parameters specified as nested elements

This task forms an implicit FileSet and supports most attributes of <fileset> (dir becomes srcdir) as well as the nested <include>, <exclude> and <patternset> elements.

srcdir, classpath, sourcepath, bootclasspath and extdirs

<javac>'s srcdir, classpath, sourcepath, bootclasspath, and extdirs attributes are path-like structures and can also be set via nested <src> (note the different name!), <classpath>, <sourcepath>, <bootclasspath> and <extdirs> elements, respectively.

compilerarg

You can specify additional command line arguments for the compiler with nested <compilerarg> elements. These elements are specified like Command-line Arguments but have an additional attribute that can be used to enable arguments only if a given compiler implementation will be used.

Attribute	Description	Required
value	See Command-line Arguments.	Exactly one of these.
line		
file		
path		
prefix	See Command-line Arguments. *Since Ant 1.8.*	No
suffix		No
compiler	Only pass the specified argument if the chosen compiler implementation matches the value of this attribute. Legal values are the same as those in the above list of valid compilers.)	No

compilerclasspath *since Ant 1.8.0*

A PATH like structure holding the classpath to use when loading the compiler implementation if a custom class has been specified. Doesn't have any effect when using one of the built-in compilers.

Any nested element of a type that implements CompilerAdapter *since Ant 1.8.0*

If a defined type implements the `CompilerAdapter` interface a nested element of that type can be used as an alternative to the `compiler` attribute.

Examples

```
<javac srcdir="${src}"
       destdir="${build}"
       classpath="xyz.jar"
       debug="on"
       source="1.4"
/>
```

compiles all `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. The classpath used includes `xyz.jar`, and compiling with debug information is on. The source level is 1.4, so you can use `assert` statements.

```
<javac srcdir="${src}"
       destdir="${build}"
       fork="true"
       source="1.2"
       target="1.2"
/>
```

compiles all `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. This will fork off the javac compiler using the default `javac` executable. The source level is 1.2 (similar to 1.1 or 1.3) and the class files should be runnable under JDK 1.2+ as well.

```
<javac srcdir="${src}"
       destdir="${build}"
       fork="java$$javac.exe"
       source="1.5"
/>
```

compiles all `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. This will fork off the javac compiler, using the executable named `java$javac.exe`. Note that the `$` sign needs to be escaped by a second one. The source level is 1.5, so you can use generics.

```
<javac srcdir="${src}"
       destdir="${build}"
       includes="mypackage/p1/**,mypackage/p2/**"
       excludes="mypackage/p1/testpackage/**"
       classpath="xyz.jar"
       debug="on"
/>
```

compiles `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. The classpath used includes `xyz.jar`, and debug information is on. Only files under `mypackage/p1` and `mypackage/p2` are used. All files in and below the `mypackage/p1/testpackage` directory are excluded from compilation. You didn't specify a source or target level, so the actual values used will depend on which JDK you ran Ant with.

```
<javac srcdir="${src}:${src2}"
       destdir="${build}"
       includes="mypackage/p1/**,mypackage/p2/**"
       excludes="mypackage/p1/testpackage/**"
       classpath="xyz.jar"
       debug="on"
/>
```

is the same as the previous example, with the addition of a second source path, defined by the property `src2`. This can also be represented using nested `<src>` elements as follows:

```
  <javac destdir="${build}"
         classpath="xyz.jar"
         debug="on">
    <src path="${src}"/>
    <src path="${src2}"/>
    <include name="mypackage/p1/**"/>
    <include name="mypackage/p2/**"/>
    <exclude name="mypackage/p1/testpackage/**"/>
  </javac>
```

If you want to run the javac compiler of a different JDK, you should tell Ant, where to find the compiler and which version of JDK you will be using so it can choose the correct command line switches. The following example executes a JDK 1.1 javac in a new process and uses the correct command line switches even when Ant is running in a Java VM of a different version:

```
  <javac srcdir="${src}"
         destdir="${build}"
         fork="yes"
         executable="/opt/java/jdk1.1/bin/javac"
         compiler="javac1.1"
  />
```

Note: If you wish to compile only source files located in certain packages below a common root, use the `include`/`exclude` attributes or `<include>`/`<exclude>` nested elements to filter for these packages. Do not include part of your package structure in the `srcdir` attribute (or nested `<src>` elements), or Ant will recompile your source files every time you run your compile target. See the Ant FAQ for additional information.

If you wish to compile only files explicitly specified and disable javac's default searching mechanism then you can unset the sourcepath attribute:

```
  <javac sourcepath="" srcdir="${src}"
         destdir="${build}" >
    <include name="**/*.java"/>
    <exclude name="**/Example.java"/>
  </javac>
```

That way the javac will compile all java source files under "${src}" directory but skip the examples. The compiler will even produce errors if some of the non-example files refers to them.

If you wish to compile with a special JDK (another than the one Ant is currently using), set the `executable` and `fork` attribute. Using `taskname` could show in the log, that these settings are fix.

```
  <javac srcdir=""
         destdir=""
         executable="path-to-java14-home/bin/javac"
         fork="true"
         taskname="javac1.4" />
```

Note: If you are using Ant on Windows and a new DOS window pops up for every use of an external compiler, this may be a problem of the JDK you are using. This problem may occur with all JDKs < 1.2.

If you want to activate other compiler options like *lint* you could use the `<compilerarg>` element:

```
  <javac srcdir="${src.dir}"
         destdir="${classes.dir}"
         classpathref="libraries">
    <compilerarg value="-Xlint"/>
  </javac>
```

If you want to use a custom CompilerAdapter `org.example.MyAdapter` you can either use the compiler attribute:

```
<javac srcdir="${src.dir}"
```

```
       destdir="${classes.dir}"
       compiler="org.example.MyAdapter"/>
```

or a define a type and nest this into the task like in:

```
<componentdef classname="org.example.MyAdapter"
              name="myadapter"/>
<javac srcdir="${src.dir}"
       destdir="${classes.dir}">
  <myadapter/>
</javac>
```

in which case your compiler adapter can support attributes and nested elements of its own.

Jikes Notes

You need Jikes 1.15 or later.

Jikes supports some extra options, which can be set be defining the properties shown below prior to invoking the task. The setting for each property will be in affect for all `<javac>` tasks throughout the build. The Ant developers are aware that this is ugly and inflexible – expect a better solution in the future. All the options are boolean, and must be set to `true` or `yes` to be interpreted as anything other than false. By default, `build.compiler.warnings` is `true`, while all others are `false`.

Property	Description	Default
build.compiler.emacs	Enable emacs-compatible error messages.	`false`
build.compiler.fulldepend	Enable full dependency checking; see the `+F` switch in the Jikes manual.	`false`
build.compiler.pedantic	Enable pedantic warnings.	`false`
build.compiler.warnings **Deprecated**. Use `<javac>`'s `nowarn` attribute instead.	Don't disable warning messages.	`true`

Jvc Notes

Jvc will enable Microsoft extensions unless you set the property `build.compiler.jvc.extensions` to false before invoking `<javac>`.

Bootstrap Options

The Sun javac compiler has a *bootclasspath* command line option - this corresponds to the "bootclasspath" attribute/element of the <javac> task. The Sun compiler also allows more control over the boot classpath using the -X and -J-X attributes. One can set these by using the <compilerarg>. Since Ant 1.6.0, there is a shortcut to convert path references to strings that can by used in an OS independent fashion (see pathshortcut). For example:

```
  <path id="lib.path.ref">
    <fileset dir="lib" includes="*.jar"/>
  </path>
  <javac srcdir="src" destdir="classes">
    <compilerarg arg="-Xbootclasspath/p:${toString:lib.path.ref}"/>
  </javac>
```

OpenJDK Notes

The openjdk project has provided the javac compiler as an opensource project. The output of this project is a `javac.jar` which contains the javac compiler. This compiler may be used with the `<javac>` task with the use of a -

`Xbootclasspath/p` java argument. The argument needs to be given to the runtime system of the javac executable, so it needs to be prepended with a "-J". For example:

```
<property name="patched.javac.jar"
          location="${my.patched.compiler}/dist/lib/javac.jar"/>

<presetdef name="patched.javac">
  <javac fork="yes">
    <compilerarg value="-J-Xbootclasspath/p:${patched.javac.jar}"/>
  </javac>
</presetdef>


<patched.javac srcdir="src/java" destdir="build/classes"
               debug="yes"/>
```

Note on package-info.java

`package-info.java` files were introduced in Java5 to allow package level annotations. On compilation, if the java file does not contain runtime annotations, there will be no .class file for the java file. Up to **Ant 1.7.1**, when the <javac> task is run again, the task will try to compile the package-info java files again.

With Ant 1.7.1 a different kind of logic was introduced that involved the timestamp of the directory that would normally contain the .class file. This logic turned out to lead to Ant not recompiling `package-info.java` in certain setup.

Starting with Ant 1.8.0 Ant will create "empty" `package-info.class` files if it compiles a `package-info.java` and no `package-info.class` file has been created by the compiler itself.

Rmic

Description

Runs the rmic compiler for a certain class.

Rmic can be run on a single class (as specified with the classname attribute) or a number of classes at once (all classes below base that are neither _Stub nor _Skel classes). If you want to rmic a single class and this class is a class nested into another class, you have to specify the classname in the form `Outer$$Inner` instead of `Outer.Inner`.

It is possible to refine the set of files that are being rmiced. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `base`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

It is possible to use different compilers. This can be selected with the "build.rmic" property, the `compiler` attribute. or a nested element. Here are the choices:

- default -the default compiler (kaffe or sun) for the platform.
- sun (the standard compiler of the JDK)
- kaffe (the standard compiler of Kaffe)
- weblogic
- forking - the sun compiler forked into a separate process (since Ant 1.7)
- xnew - the sun compiler forked into a separate process, with the -Xnew option (since Ant 1.7). This is the most reliable way to use -Xnew
- "" (empty string). This has the same behaviour as not setting the compiler attribute. First the value of `build.rmic` is used if defined, and if not, the default for the platform is chosen. If build.rmic is set to this, you get the default.

The miniRMI project contains a compiler implementation for this task as well, please consult miniRMI's documentation to learn how to use it.

Parameters

Attribute	Description	Required
base	the location to store the compiled files. Also serves as the parent directory for any non-Fileset includes, etc. (This functionality has remained unchanged.)	*1
destdir	the location to store the compiled files.	
classname	the class for which to run `rmic`.	No
filtering	indicates whether token filtering should take place	No
sourcebase	Pass the "-keepgenerated" flag to rmic and move the generated source file to the given sourcebase directory.	No
stubversion	Specify the JDK version for the generated stub code. Specify "1.1" to pass the "-v1.1" option to rmic, "1.2" for -v12, compat for -vcompat.	No, default="compat"

	Since Ant1.7, if you do not specify a version, and do not ask for iiop or idl files, "compat" is selected.	
classpath	The classpath to use during compilation	No
classpathref	The classpath to use during compilation, given as reference to a PATH defined elsewhere	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
verify	check that classes implement Remote before handing them to rmic (default is false)	No
iiop	indicates that portable (RMI/IIOP) stubs should be generated	No
iiopopts	additional arguments for IIOP class generation	No
idl	indicates that IDL output files should be generated	No
idlopts	additional arguments for IDL file generation	No
debug	generate debug info (passes -g to rmic). Defaults to false.	No
includeAntRuntime	whether to include the Ant run-time libraries; defaults to `yes`.	No
includeJavaRuntime	whether to include the default run-time libraries from the executing VM; defaults to `no`.	No
extdirs	location of installed extensions.	No
compiler	The compiler implementation to use. If this attribute is not set, the value of the `build.rmic` property, if set, will be used. Otherwise, the default compiler for the current VM will be used. (See the above list of valid compilers.)	No
executable	Complete path to the `rmic` executable to use in case of the `forking` or `xnew` compiler. Defaults to the rmic compiler of the Java version that is currently running Ant. *Since Ant 1.8.0.*	No
listfiles	Indicates whether the source files to be compiled will be listed; defaults to `no`. *Since Ant 1.8.0.*	No

*1:

- Maintaining compatibility, `base`, when specified by itself, serves as both the parent directory for any source files AND the output directory.
- `destdir` can be used to specify the output directory, allowing for `base` to be used as the parent directory for any source files.
- At least one of either `base` or `destdir` must be specified and exist, or a runtime error will occur.

Parameters specified as nested elements

classpath and extdirs

`Rmic`

's *classpath* and *extdirs* attributes are PATH like structure and can also be set via a nested *classpath* and *extdirs* elements.

compilerarg

You can specify additional command line arguments for the compiler with nested `<compilerarg>` elements. These elements are specified like Command-line Arguments but have an additional attribute that can be used to enable arguments only if a given compiler implementation will be used.

Attribute	Description	Required
value	See Command-line Arguments.	Exactly one of these.
line		
file		
path		
prefix	See Command-line Arguments. *Since Ant 1.8.*	No
suffix		No
compiler	Only pass the specified argument if the chosen compiler implementation matches the value of this attribute. Legal values are the same as those in the above list of valid compilers.)	No

compilerclasspath *since Ant 1.8.0*

A PATH like structure holding the classpath to use when loading the compiler implementation if a custom class has been specified. Doesn't have any effect when using one of the built-in compilers.

Any nested element of a type that implements RmicAdapter *since Ant 1.8.0*

If a defined type implements the `RmicAdapter` interface a nested element of that type can be used as an alternative to the `compiler` attribute.

Examples

```
  <rmic classname="com.xyz.FooBar" base="${build}/classes"/>
```

runs the rmic compiler for the class `com.xyz.FooBar`. The compiled files will be stored in the directory `${build}/classes`.

```
  <rmic base="${build}/classes" includes="**/Remote*.class"/>
```

runs the rmic compiler for all classes with `.class` files below `${build}/classes` whose classname starts with *Remote*. The compiled files will be stored in the directory `${build}/classes`.

If you want to use a custom RmicAdapter `org.example.MyAdapter` you can either use the compiler attribute:

```
<rmic classname="com.xyz.FooBar"
      base="${build}/classes"
      compiler="org.example.MyAdapter"/>
```

or a define a type and nest this into the task like in:

```
<componentdef classname="org.example.MyAdapter"
              name="myadapter"/>
<rmic classname="com.xyz.FooBar"
      base="${build}/classes">
```

```
  <myadapter/>
</rmic>
```

in which case your compiler adapter can support attributes and nested elements of its own.

Exec

Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

Note that you cannot interact with the forked program, the only way to send input to it is via the input and inputstring attributes. Also note that since Ant 1.6, any attempt to read input in the forked program will receive an EOF (-1). This is a change from Ant 1.5, where such an attempt would block.

If you want to execute an executable using a path relative to the project's basedir, you may need to use `vmlauncher="false"` on some operating systems - but even this may fail (Solaris 8/9 has been reported as problematic). The `resolveexecutable` attribute should be more reliable, as would be something like

```
  <property name="executable-full-path"
            location="../relative/path/to/executable"/>
  <exec executable="${executable-full-path}" ...
```

Windows Users

The `<exec>` task delegates to `Runtime.exec` which in turn apparently calls `::CreateProcess`. It is the latter Win32 function that defines the exact semantics of the call. In particular, if you do not put a file extension on the executable, only ".EXE" files are looked for, not ".COM", ".CMD" or other file types listed in the environment variable PATHEXT. That is only used by the shell.

Note that *.bat* files cannot in general by executed directly. One normally needs to execute the command shell executable `cmd` using the `/c` switch.

```
<target name="help">
  <exec executable="cmd">
    <arg value="/c"/>
    <arg value="ant.bat"/>
    <arg value="-p"/>
  </exec>
</target>
```

A common problem is not having the executable on the PATH. In case you get an error message `Cannot run program "...":CreateProcess error=2. The system cannot find the path specified.` have a look at your PATH variable. Just type the command directly on the command line and if Windows finds it, Ant should do it too. (Otherwise ask on the user mailinglist for help.) If Windows can not execute the program add the directory of the programm to the PATH (`set PATH=%PATH%;dirOfProgram`) or specify the absolute path in the `executable` attribute in your buildfile.

Cygwin Users

The `<exec>` task will not understand paths such as /bin/sh for the executable parameter. This is because the Java VM in which Ant is running is a standard Windows executable and is not aware of the Cygwin environment (i.e., doesn't load `cygwin1.dll`). The only work-around for this is to compile a JVM under Cygwin (at your own risk). See for instance sun jdk 6 build instructions for cygwin.

OpenVMS Users

The command specified using `executable` and `<arg>` elements is executed exactly as specified inside a temporary

DCL script. This has some implications:

- paths have to be written in VMS style
- if your `executable` points to a DCL script remember to prefix it with an `@`-sign (e.g. `executable="@[FOO]BAR.COM"`), just as you would in a DCL script

For `<exec>` to work in an environment with a Java VM older than version 1.4.1-2 it is also *required* that the logical `JAVA$FORK_SUPPORT_CHDIR` is set to `TRUE` in the job table (see the *JDK Release Notes*).

Please note that the Java VM provided by HP doesn't follow OpenVMS' conventions of exit codes. If you run a Java VM with this task, the task may falsely claim that an error occurred (or silently ignore an error). Don't use this task to run `JAVA.EXE`, use a `<java>` task with the `fork` attribute set to `true` instead as this task will follow the VM's interpretation of exit codes.

RedHat S/390 Users

It has been reported on the VMESA-LISTSERV that shell scripts invoked via the Ant Exec task must have their interpreter specified, i.e., the scripts must start with something like:

```
#!/bin/bash
```

or the task will fail as follows:

```
[exec] Warning: UNIXProcess.forkAndExec native error: Exec format error
[exec] Result: 255
```

Running Ant as a background process on Unix(-like) systems

If you run Ant as a background process (like `ant &`) and use the `<exec>` task with `spawn` set to `false`, you must provide explicit input to the forked process or Ant will be suspended because it tries to read from the standard input.

Parameters

Attribute	Description	Required
command	the command to execute with all command line arguments. **deprecated, use executable and nested `<arg>` elements instead**.	Exactly one of the two.
executable	the command to execute without any command line arguments.	
dir	the directory in which the command should be executed.	No
os	list of Operating Systems on which the command may be executed. If the current OS's name is contained in this list, the command will be executed. The OS's name is determined by the Java Virtual machine and is set in the "os.name" system property.	No
osfamily	OS family as used in the <os> condition. *since Ant 1.7*	No
spawn	whether or not you want the command to be spawned Default is false. If you spawn a command, its output will not be logged by ant. The input, output, error, and result property settings are not active when spawning a process. *since Ant 1.6*	No
output	Name of a file to which to write the output. If the error stream is not also redirected to a file or property, it will appear in this output.	No
error	The file to which the standard error of the command should be redirected. *since Ant 1.6*	No

logError	This attribute is used when you wish to see error output in Ant's log and you are redirecting output to a file/property. The error output will not be included in the output file/property. If you redirect error with the "error" or "errorProperty" attributes, this will have no effect. *since Ant 1.6*	No
append	Whether output and error files should be appended to or overwritten. Defaults to false.	No
outputproperty	The name of a property in which the output of the command should be stored. Unless the error stream is redirected to a separate file or stream, this property will include the error output.	No
errorproperty	The name of a property in which the standard error of the command should be stored. *since Ant 1.6*	No
input	A file from which the executed command's standard input is taken. This attribute is mutually exclusive with the inputstring attribute. *since Ant 1.6*	No
inputstring	A string which serves as the input stream for the executed command. This attribute is mutually exclusive with the input attribute. *since Ant 1.6*	No
resultproperty	the name of a property in which the return code of the command should be stored. Only of interest if failonerror=false.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a return code signaling failure. Defaults to false.	No
failifexecutionfails	Stop the build if we can't start the program. Defaults to true.	No
newenvironment	Do not propagate old environment when new environment variables are specified.	No, default is *false*
vmlauncher	Run command using the Java VM's execution facilities where available. If set to false the underlying OS's shell, either directly or through the antRun scripts, will be used. Under some operating systems, this gives access to facilities not normally available through the VM including, under Windows, being able to execute scripts, rather than their associated interpreter. If you want to specify the name of the executable as a relative path to the directory given by the dir attribute, it may become necessary to set vmlauncher to false as well.	No, default is *true*
resolveexecutable	When this attribute is true, the name of the executable is resolved firstly against the project basedir and if that does not exist, against the execution directory if specified. On Unix systems, if you only want to allow execution of commands in the user's path, set this to false. *since Ant 1.6*	No, default is *false*
searchpath	When this attribute is true, then system path environment variables will be searched when resolving the location of the executable. *since Ant 1.6.3*	No, default is *false*

Examples

```
<exec dir="${src}" executable="cmd.exe" os="Windows 2000" output="dir.txt">
  <arg line="/c dir"/>
</exec>
```

Parameters specified as nested elements

arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments.

env

It is possible to specify environment variables to pass to the system command via nested <env> elements.

Attribute	Description	Required
key	The name of the environment variable. *Note: (Since Ant 1.7)* For windows, the name is case-insensitive.	Yes
value	The literal value for the environment variable.	Exactly one of these.
path	The value for a PATH like environment variable. You can use ; or : as path separators and Ant will convert it to the platform's local conventions.	
file	The value for the environment variable. Will be replaced by the absolute filename of the file by Ant.	

redirector

Since Ant 1.6.2

A nested I/O Redirector can be specified. In general, the attributes of the redirector behave as the corresponding attributes available at the task level. The most notable peculiarity stems from the retention of the <exec> attributes for backwards compatibility. Any file mapping is done using a `null` sourcefile; therefore not all Mapper types will return results. When no results are returned, redirection specifications will fall back to the task level attributes. In practice this means that defaults can be specified for input, output, and error output files.

Errors and return codes

By default the return code of a <exec> is ignored; when you set `failonerror="true"` then any return code signaling failure (OS specific) causes the build to fail. Alternatively, you can set `resultproperty` to the name of a property and have it assigned to the result code (barring immutability, of course).

If the attempt to start the program fails with an OS dependent error code, then <exec> halts the build unless `failifexecutionfails` is set to `false`. You can use that to run a program if it exists, but otherwise do nothing.

What do those error codes mean? Well, they are OS dependent. On Windows boxes you have to look at the documentation; error code 2 means 'no such program', which usually means it is not on the path. Any time you see such an error from any Ant task, it is usually not an Ant bug, but some configuration problem on your machine.

Examples

```
<exec executable="emacs">
  <env key="DISPLAY" value=":1.0"/>
</exec>
```

starts `emacs` on display 1 of the X Window System.

```
<property environment="env"/>
<exec ... >
  <env key="PATH" path="${env.PATH}:${basedir}/bin"/>
</exec>
```

adds `${basedir}/bin` to the `PATH` of the system command.

```
<property name="browser" location="C:/Program Files/Internet Explorer/iexplore.exe"/>
<property name="file" location="ant/docs/manual/index.html"/>
```

```
<exec executable="${browser}" spawn="true">
    <arg value="${file}"/>
</exec>
```

Starts the *${browser}* with the specified *${file}* and end the Ant process. The browser will remain.

```
<exec executable="cat">
    <redirector outputproperty="redirector.out"
                errorproperty="redirector.err"
                inputstring="blah before blah">
        <inputfilterchain>
            <replacestring from="before" to="after"/>
        </inputfilterchain>
        <outputmapper type="merge" to="redirector.out"/>
        <errormapper type="merge" to="redirector.err"/>
    </redirector>
</exec>
```

Sends the string "blah before blah" to the "cat" executable, using an <inputfilterchain> to replace "before" with "after" on the way in. Output is sent to the file "redirector.out" and stored in a property of the same name. Similarly, error output is sent to a file and a property, both named "redirector.err".

Note: do not try to specify arguments using a simple arg-element and separate them by spaces. This results in only a single argument containing the entire string.

Timeouts: If a timeout is specified, when it is reached the sub process is killed and a message printed to the log. The return value of the execution will be "-1", which will halt the build if `failonerror=true`, but be ignored otherwise.

Cvs

Description

Handles packages/modules retrieved from a CVS repository.

Important: This task needs "cvs" on the path. If it isn't, you will get an error (such as error 2 on windows). If <cvs> doesn't work, try to execute cvs.exe from the command line in the target directory in which you are working. Also note that this task assumes that the cvs exceutable is compatible with the Unix version from cvshome.org, this is not completely true for certain other cvs clients - like CVSNT for example - and some operation may fail when using such an incompatible client.

CVSNT Note: CVSNT prefers users to store the passwords inside the registry. If the cvspass task and the passfile attribute don't seem to work for you, the most likely reason is that CVSNT ignores your .cvspass file completely. See bugzilla report 21657 for recommended workarounds.

Parameters

Attribute	Description	Required
command	the CVS command to execute.	No, default "checkout".
compression	true or false - if set to true, this is the same as compressionlevel="3"	No. Defaults to false.
compressionlevel	A number between 1 and 9 (corresponding to possible values for CVS' -z# argument). Any other value is treated as compression="false"	No. Defaults to no compression.
cvsRoot	the CVSROOT variable.	No
cvsRsh	the CVS_RSH variable.	No
dest	the directory where the checked out files should be placed. Note that this is different from CVS's -d command line switch as Ant will never shorten pathnames to avoid empty directories.	No, default is project's basedir.
package	the package/module to check out. **Note:** multiple attributes can be split using spaces. Use a nested <module> element if you want to specify a module with spaces in its name.	No
tag	the tag of the package/module to check out.	No
date	Use the most recent revision no later than the given date	No
quiet	suppress informational messages. This is the same as -q on the command line.	No, default "false"
reallyquiet	suppress all messages. This is the same as -Q on the command line. *since Ant 1.6.*	No, default "false"
noexec	report only, don't change any files.	No, default to "false"
output	the file to direct standard output from the command.	No, default output to ANT Log as MSG_INFO.
error	the file to direct standard error from the command.	No, default error to ANT Log as MSG_WARN.

append	whether to append output/error when redirecting to a file.	No, default to "false".
port	Port used by CVS to communicate with the server.	No, default port `2401`.
passfile	Password file to read passwords from.	No, default file `~/.cvspass`.
failonerror	Stop the build process if the command exits with a return code other than `0`. Defaults to "false"	No

Parameters specified as nested elements

module

Specifies a package/module to work on, unlike the package attribute modules specified using this attribute can contain spaces in their name.

Attribute	Description	Required
name	The module's/package's name.	Yes.

Examples

```
  <cvs cvsRoot=":pserver:anoncvs@cvs.apache.org:/home/cvspublic"
       package="ant"
       dest="${ws.dir}"
  />
```

checks out the package/module "ant" from the CVS repository pointed to by the `cvsRoot` attribute, and stores the files in "`${ws.dir}`".

```
  <cvs dest="${ws.dir}" command="update"/>
```

updates the package/module that has previously been checked out into "`${ws.dir}`".

```
  <cvs command="-q diff -u -N" output="patch.txt"/>
```

silently (`-q`) creates a file called `patch.txt` which contains a unified (`-u`) diff which includes new files added via "cvs add" (`-N`) and can be used as input to patch. The equivalent, using `<commandline>` elements, is:

```
<cvs output="patch">
    <commandline>
        <argument value="-q"/>
        <argument value="diff"/>
        <argument value="-u"/>
        <argument value="-N"/>
    </commandline>
</cvs>
```

or:

```
<cvs output="patch">
    <commandline>
        <argument line="-q diff -u -N"/>
    </commandline>
</cvs>
```

You may include as many `<commandline>` elements as you like. Each will inherit the `failonerror`, `compression`, and other "global" parameters from the `<cvs>` element.

```
  <cvs command="update -A -d"/>
```

Updates from the head of repository ignoring sticky bits (`-A`) and creating any new directories as necessary (`-d`).

Note: the text of the command is passed to cvs "as-is" so any cvs options should appear before the command, and any command options should appear after the command as in the diff example above. See the cvs manual for details, specifically the Guide to CVS commands

Setproxy Task

Sets Java's web proxy properties, so that tasks and code run in the same JVM can have through-the-firewall access to remote web sites, and remote ftp sites.

Description

Sets Java's web proxy properties, so that tasks and code run in the same JVM can have through-the-firewall access to remote web sites, and remote ftp sites. You can nominate an http and ftp proxy, or a socks server, reset the server settings, or do nothing at all.

Examples

```
<setproxy/>
```

do nothing

```
<setproxy proxyhost="firewall"/>
```

set the proxy to firewall:80

```
<setproxy proxyhost="firewall" proxyport="81"/>
```

set the proxy to firewall:81

```
<setproxy proxyhost=""/>
```

stop using the http proxy; don't change the socks settings

```
<setproxy socksproxyhost="socksy"/>
```

use socks via socksy:1080

```
<setproxy socksproxyhost=""/>
```

stop using the socks server.

You can set a username and password for http with the `proxyHost` and `proxyPassword` attributes. On Java1.4 and above these can also be used against SOCKS5 servers.

Parameters

Attribute	Description	Type	Requirement
nonproxyhosts	A list of hosts to bypass the proxy on. These should be separated with the vertical bar character '|'. Only in Java 1.4 does ftp use this list. e.g. fozbot.corp.sun.com|*.eng.sun.com	String	Optional
proxyhost	the HTTP/ftp proxy host. Set this to "" for the http proxy option to be disabled	String	
proxypassword	Set the password for the proxy. Used only if the proxyUser is set.	String	
proxyport	the HTTP/ftp proxy port number; default is 80	int	
proxyuser	set the proxy user. Probably requires a password to accompany this setting. Default=""	String	
socksproxyhost	The name of a Socks server. Set to "" to turn socks proxying off.	String	
socksproxyport	Set the ProxyPort for socks connections. The default value is 1080	int	

Parameters as nested elements

Property

Description

Sets a property (by name and value), or set of properties (from file or resource) in the project. Properties are case sensitive.

Properties are immutable: whoever sets a property first freezes it for the rest of the build; they are most definitely not variables.

There are seven ways to set properties:

- By supplying both the *name* and one of *value* or *location* attribute.
- By supplying the *name* and nested text.
- By supplying both the *name* and *refid* attribute.
- By setting the *file* attribute with the filename of the property file to load. This property file has the format as defined by the file used in the class java.util.Properties, with the same rules about how non-ISO8859-1 characters must be escaped.
- By setting the *url* attribute with the url from which to load the properties. This url must be directed to a file that has the format as defined by the file used in the class java.util.Properties.
- By setting the *resource* attribute with the resource name of the property file to load. A resource is a property file on the current classpath, or on the specified classpath.
- By setting the *environment* attribute with a prefix to use. Properties will be defined for every environment variable by prefixing the supplied name and a period to the name of the variable.

Although combinations of these ways are possible, only one should be used at a time. Problems might occur with the order in which properties are set, for instance.

The value part of the properties being set, might contain references to other properties. These references are resolved at the time these properties are set. This also holds for properties loaded from a property file.

A list of predefined properties can be found here.

Since Ant 1.7.1 it is possible to load properties defined in xml according to Suns DTD, if Java5+ is present. For this the name of the file, resource or url has to end with `.xml`.

OpenVMS Users

With the `environment` attribute this task will load all defined logicals on an OpenVMS system. Logicals with multiple equivalence names get mapped to a property whose value is a comma separated list of all equivalence names. If a logical is defined in multiple tables, only the most local definition is available (the table priority order being PROCESS, JOB, GROUP, SYSTEM).

Parameters

Attribute	Description	Required
name	the name of the property to set.	No
value	the value of the property.	One of these or
location	Sets the property to the absolute filename of the given file. If the value of this attribute is an absolute path, it is left unchanged (with / and \ characters	

	converted to the current platforms conventions). Otherwise it is taken as a path relative to the project's basedir and expanded.	nested text, when using the name attribute
refid	Reference to an object defined elsewhere. Only yields reasonable results for references to PATH like structures or properties.	
resource	the name of the classpath resource containing properties settings in properties file format.	One of these, when **not** using the name attribute
file	the location of the properties file to load.	
url	a url containing properties-format settings.	
environment	the prefix to use when retrieving environment variables. Thus if you specify environment="myenv" you will be able to access OS-specific environment variables via property names "myenv.PATH" or "myenv.TERM". Note that if you supply a property name with a final "." it will not be doubled; i.e. environment="myenv." will still allow access of environment variables through "myenv.PATH" and "myenv.TERM". This functionality is currently only implemented on select platforms. Feel free to send patches to increase the number of platforms on which this functionality is supported ;). Note also that properties are case-sensitive, even if the environment variables on your operating system are not; e.g. Windows 2000's system path variable is set to an Ant property named "env.Path" rather than "env.PATH".	
classpath	the classpath to use when looking up a resource.	No
classpathref	the classpath to use when looking up a resource, given as reference to a `<path>` defined elsewhere..	No
prefix	Prefix to apply to properties loaded using `file`, `resource`, or `url`. A "." is appended to the prefix if not specified.	No
relative	If set to `true` the relative path to `basedir` is set. *Since Ant 1.8.0*	No (default=`false`)
basedir	The basedir to calculate the relative path from. *Since Ant 1.8.0*	No (default=`${basedir}`)

Parameters specified as nested elements

classpath

`Property`'s *classpath* attribute is a PATH like structure and can also be set via a nested *classpath* element.

Examples

```
  <property name="foo.dist" value="dist"/>
```

sets the property `foo.dist` to the value "dist".

```
  <property name="foo.dist">dist</property>
```

sets the property `foo.dist` to the value "dist".

```
  <property file="foo.properties"/>
```

reads a set of properties from a file called "foo.properties".

```
  <property url="http://www.mysite.com/bla/props/foo.properties"/>
```

reads a set of properties from the address "http://www.mysite.com/bla/props/foo.properties".

```
  <property resource="foo.properties"/>
```

reads a set of properties from a resource called "foo.properties".

Note that you can reference a global properties file for all of your Ant builds using the following:

```
  <property file="${user.home}/.ant-global.properties"/>
```

since the "user.home" property is defined by the Java virtual machine to be your home directory. Where the "user.home" property resolves to in the file system depends on the operating system version and the JVM implementation. On Unix based systems, this will map to the user's home directory. On modern Windows variants, this will most likely resolve to the user's directory in the "Documents and Settings" folder. Older windows variants such as Windows 98/ME are less predictable, as are other operating system/JVM combinations.

```
  <property environment="env"/>
  <echo message="Number of Processors = ${env.NUMBER_OF_PROCESSORS}"/>
  <echo message="ANT_HOME is set to = ${env.ANT_HOME}"/>
```

reads the system environment variables and stores them in properties, prefixed with "env". Note that this only works on *select* operating systems. Two of the values are shown being echoed.

```
  <property environment="env"/>
  <property file="${user.name}.properties"/>
  <property file="${env.STAGE}.properties"/>
  <property file="build.properties"/>
```

This buildfile uses the properties defined in `build.properties`. Regarding to the environment variable `STAGE` some or all values could be overwritten, e.g. having `STAGE=test` and a `test.properties` you have special values for that (like another name for the test server). Finally all these values could be overwritten by personal settings with a file per user.

```
  <property name="foo" location="my/file.txt" relative="true" basedir=".."/>
```

Stores the relative path in `foo`: projectbasedir/my/file.txt

```
  <property name="foo" location="my/file.txt" relative="true" basedir="cvs"/>
```

Stores the relative path in `foo`: ../my/file.txt

Property Files

As stated, this task will load in a properties file stored in the file system, or as a resource on a classpath. Here are some interesting facts about this feature

1. If the file is not there, nothing is printed except at -verbose log level. This lets you have optional configuration files for every project, that team members can customize.
2. The rules for this format are laid down by Sun. This makes it hard for Team Ant to field bug reports about it.
3. Trailing spaces are not stripped. It may have been what you wanted.
4. Want unusual characters? Escape them \u0456 or \" style.
5. Ant Properties are expanded in the file.

In-file property expansion is very cool. Learn to use it.

Example:

```
build.compiler=jikes
deploy.server=lucky
deploy.port=8080
deploy.url=http://${deploy.server}:${deploy.port}/
```

Notes about environment variables

Ant runs on Java 1.2 therefore it cant use Java5 features for accessing environment variables. So it starts a command in a new process which prints the environment variables, analyzes the output and creates the properties.
There are commands for the following operating systems implemented in Execute.java (method `getProcEnvCommand()`):

OS	command
os/2	cmd /c set
windows	
* win9x	command.com /c set
* other	cmd /c set
z/os	/bin/env **OR** /usr/bin/env **OR** env *(depending on read rights)*
unix	/bin/env **OR** /usr/bin/env **OR** env *(depending on read rights)*
netware	env
os/400	env
openvms	show logical

Filter

Description

Sets a token filter for this project or read multiple token filter from an input file and sets these as filters. Token filters are used by all tasks that perform file copying operations through the Project commodity methods. See the warning *here* before using.

Note 1: the token string must not contain the separators chars (@).
Note 2: Either token and value attributes must be provided, or only the filtersfile attribute.

Parameters

Attribute	Description	Required
token	the token string without @	Yes*
value	the string that should be put to replace the token when the file is copied	Yes*
filtersfile	The file from which the filters must be read. This file must be a formatted as a property file.	Yes*

* see notes 1 and 2 above parameters table.

Examples

```
<filter token="year" value="2000"/>
<copy todir="${dest.dir}" filtering="true">
  <fileset dir="${src.dir}"/>
</copy>
```

will copy recursively all the files from the *src.dir* directory into the *dest.dir* directory replacing all the occurrences of the string *@year@* with *2000*.

```
<filter filtersfile="deploy_env.properties"/>
```

will read all property entries from the *deploy_env.properties* file and set these as filters.

Taskdef

Description

Adds a task definition to the current project, such that this new task can be used in the current project.

This task is a form of Typedef with the attributes "adapter" and "adaptto" set to the values "org.apache.tools.ant.TaskAdapter" and "org.apache.tools.ant.Task" respectively.

Examples

```
  <taskdef name="myjavadoc" classname="com.mydomain.JavadocTask"/>
```

makes a task called `myjavadoc` available to Ant. The class `com.mydomain.JavadocTask` implements the task.

Tstamp

Description

Sets the `DSTAMP`, `TSTAMP`, and `TODAY` properties in the current project. By default, the `DSTAMP` property is in the format "yyyyMMdd", `TSTAMP` is in the format "hhmm", and `TODAY` is in the format "MMMM dd yyyy". Use the nested `<format>` element to specify a different format.

These properties can be used in the build-file, for instance, to create time-stamped filenames, or used to replace placeholder tags inside documents to indicate, for example, the release date. The best place for this task is probably in an initialization target.

Parameters

Attribute	Description	Required
prefix	Prefix used for all properties set. The default is no prefix.	No

Nested Elements

The Tstamp task supports a `<format>` nested element that allows a property to be set to the current date and time in a given format. The date/time patterns are as defined in the Java SimpleDateFormat class. The format element also allows offsets to be applied to the time to generate different time values.

Attribute	Description	Required
property	The property to receive the date/time string in the given pattern.	Yes
pattern	The date/time pattern to be used. The values are as defined by the Java SimpleDateFormat class.	Yes
timezone	The timezone to use for displaying time. The values are as defined by the Java TimeZone class.	No
offset	The numeric offset to the current time	No
unit	The unit of the offset to be applied to the current time. Valid Values are • millisecond • second • minute • hour • day • week • month • year	No
locale	The locale used to create date/time string. The general form is "language, country, variant" but either variant or variant and country may be	No

	omitted. For more information please refer to documentation for the Locale class.	

Examples

```
<tstamp/>
```

sets the standard `DSTAMP`, `TSTAMP`, and `TODAY` properties according to the default formats.

```
<tstamp>
  <format property="TODAY_UK" pattern="d-MMMM-yyyy" locale="en,UK"/>
</tstamp>
```

sets the standard properties as well as the property `TODAY_UK` with the date/time pattern "d-MMMM-yyyy" using English locale (eg. 21-May-2001).

```
<tstamp>
    <format property="touch.time" pattern="MM/dd/yyyy hh:mm aa"
            offset="-5" unit="hour"/>
</tstamp>
```

Creates a timestamp, in the property touch.time, 5 hours before the current time. The format in this example is suitable for use with the `<touch>` task. The standard properties are set also.

```
<tstamp prefix="start"/>
```

Sets three properties with the standard formats, prefixed with "start.": `start.DSTAMP`, `start.TSTAMP`, and `start.TODAY`.

Import

Description

Imports another build file into the current project.

Note this task heavily relies on the ProjectHelper implementation and doesn't really perform any work of its own. If you have configured Ant to use a ProjectHelper other than Ant's default, this task may or may not work.

On execution it will read another Ant file into the same Project. This means that it basically works like the Entity Includes as explained in the Ant FAQ, as if the imported file was contained in the importing file, minus the top `<project>` tag.

The import task may only be used as a top-level task. This means that it may not be used in a target.

There are two further functional aspects that pertain to this task and that are not possible with entity includes:

- target overriding
- special properties

Target overriding

If a target in the main file is also present in at least one of the imported files, the one from the main file takes precedence.

So if I import for example a *docsbuild.xml* file named **builddocs**, that contains a "**docs**" target, I can redefine it in my main buildfile and that is the one that will be called. This makes it easy to keep the same target name, so that the overriding target is still called by any other targets--in either the main or imported buildfile(s)--for which it is a dependency, with a different implementation. The target from *docsbuild.xml* is made available by the name "**builddocs.docs**". This enables the new implementation to call the old target, thus *enhancing* it with tasks called before or after it.

If you use the *as* attribute of the task, its value will be used to prefix the overriden target's name instead of the name attribute of the project tag.

Special Properties

Imported files are treated as they are present in the main buildfile. This makes it easy to understand, but it makes it impossible for them to reference files and resources relative to their path. Because of this, for every imported file, Ant adds a property that contains the path to the imported buildfile. With this path, the imported buildfile can keep resources and be able to reference them relative to its position.

So if I import for example a *docsbuild.xml* file named **builddocs**, I can get its path as **ant.file.builddocs**, similarly to the **ant.file** property of the main buildfile.

Note that "builddocs" is not the filename, but the name attribute present in the imported project tag.

If the imported file does not have a name attribute, the ant.file.projectname property will not be set.

Since Ant 1.8.0 the task can also import resources from URLs or classpath resources (which are URLs, really). If you need to know whether the current build file's source has been a file or an URL you can consult the property **ant.file.type.*projectname*** (using the same example as above **ant.file.type.builddocs**) which either have the value

"file" or "url".

Resolving files against the imported file

Suppose your main build file called `importing.xml` imports a build file `imported.xml`, located anywhere on the file system, and `imported.xml` reads a set of properties from `imported.properties`:

```
<!-- importing.xml -->
<project name="importing" basedir="." default="...">
  <import file="${path_to_imported}/imported.xml"/>
</project>

<!-- imported.xml -->
<project name="imported" basedir="." default="...">
  <property file="imported.properties"/>
</project>
```

This snippet however will resolve `imported.properties` against the basedir of `importing.xml`, because the basedir of `imported.xml` is ignored by Ant. The right way to use `imported.properties` is:

```
<!-- imported.xml -->
<project name="imported" basedir="." default="...">
  <dirname property="imported.basedir" file="${ant.file.imported}"/>
  <property file="${imported.basedir}/imported.properties"/>
</project>
```

As explained above `${ant.file.imported}` stores the path of the build script, that defines the project called `imported`, (in short it stores the path to `imported.xml`) and `<dirname>` takes its directory. This technique also allows `imported.xml` to be used as a standalone file (without being imported in other project).

The above description only works for imported files that actually are imported from files and not from URLs. For files imported from URLs using resources relative to the imported file requires you to use tasks that can work on non-file resources in the first place. To create a relative resource you'd use something like:

```
  <loadproperties>
    <url baseUrl="${ant.file.imported}"
         relativePath="imported.properties"/>
  </loadproperties>
```

Parameters

Attribute	Description	Required
file	The file to import. If this is a relative file name, the file name will be resolved relative to the *importing* file. **Note**, this is unlike most other ant file attributes, where relative files are resolved relative to ${basedir}.	Yes or a nested resource collection
optional	If true, do not stop the build if the file does not exist, default is false.	No
as	Specifies the prefix prepended to the target names. If ommitted, the name attribute of the project tag of the imported file will be used.	No
prefixSeparator	Specifies the separator to be used between the prefix and the target name. Defaults to ".".	No

Parameters specified as nested elements

any resource or resource collection

The specified resources will be imported. *Since Ant 1.8.0*

Examples

```
  <import file="../common-targets.xml"/>
```

Imports targets from the common-targets.xml file that is in a parent directory.

```
  <import file="${deploy-platform}.xml"/>
```

Imports the project defined by the property deploy-platform

```
  <import>
    <javaresource name="common/targets.xml">
      <classpath location="common.jar"/>
    </javaresource>
  </import>
```

Imports targets from the targets.xml file that is inside the directory common inside the jar file common.jar.

How is <import> different from <include>?

The short version: Use import if you intend to override a target, otherwise use include.

When using import the imported targets are available by up to two names. Their "normal" name without any prefix and potentially with a prefixed name (the value of the as attribute or the imported project's name attribute, if any).

When using include the included targets are only available in the prefixed form.

When using import, the imported target's depends attribute remains unchanged, i.e. it uses "normal" names and allows you to override targets in the dependency list.

When using include, the included targets cannot be overridden and their depends attributes are rewritten so that prefixed names are used. This allows writers of the included file to control which target is invoked as part of the dependencies.

It is possible to include the same file more than once by using different prefixes, it is not possible to import the same file more than once.

Examples

nested.xml shall be:

```
<project>
  <target name="setUp">
    <property name="prop" value="in nested"/>
  </target>

  <target name="echo" depends="setUp">
    <echo>prop has the value ${prop}</echo>
  </target>
</project>
```

When using import like in

```
<project default="test">
  <target name="setUp">
    <property name="prop" value="in importing"/>
  </target>

  <import file="nested.xml" as="nested"/>

  <target name="test" depends="nested.echo"/>
```

```
</project>
```

Running the build file will emit:

```
setUp:

nested.echo:
     [echo] prop has the value in importing

test:
```

When using include like in

```
<project default="test">
  <target name="setUp">
    <property name="prop" value="in importing"/>
  </target>

  <include file="nested.xml" as="nested"/>

  <target name="test" depends="nested.echo"/>
</project>
```

Running the target build file will emit:

```
nested.setUp:

nested.echo:
     [echo] prop has the value in nested

test:
```

and there won't be any target named "echo" on the including build file.

Ant

Description

Runs Ant on a supplied buildfile. This can be used to build subprojects. **This task must not be used outside of a `target` if it invokes the same build file it is part of.**

When the *antfile* attribute is omitted, the file "build.xml" in the supplied directory (*dir* attribute) is used.

If no target attribute is supplied, the default target of the new project is used.

By default, all of the properties of the current project will be available in the new project. Alternatively, you can set the *inheritAll* attribute to `false` and only "user" properties (i.e., those passed on the command-line) will be passed to the new project. In either case, the set of properties passed to the new project will override the properties that are set in the new project (See also the property task).

You can also set properties in the new project from the old project by using nested property tags. These properties are always passed to the new project and any project created in that project regardless of the setting of *inheritAll*. This allows you to parameterize your subprojects. Properties defined on the command line cannot be overridden by nested `<property>` elements.

When more than one nested `<property>` element would set a property of the same name, the one declared last will win. This is for backwards compatibility reasons even so it is different from the way `<property>` tasks in build files behave.

References to data types can also be passed to the new project, but by default they are not. If you set the inheritrefs attribute to true, all references will be copied, but they will not override references defined in the new project.

Nested *`<reference>`* elements can also be used to copy references from the calling project to the new project, optionally under a different id. References taken from nested elements will override existing references that have been defined outside of targets in the new project - but not those defined inside of targets.

Parameters

Attribute	Description	Required
antfile	the buildfile to use. Defaults to "build.xml". This file is expected to be a filename relative to the dir attribute given.	No
dir	the directory to use as a basedir for the new Ant project (unless useNativeBasedir is set to true). Defaults to the current project's basedir, unless inheritall has been set to false, in which case it doesn't have a default value. This will override the basedir setting of the called project. Also serves as the directory to resolve the antfile and output attribute's values (if any).	No
target	the target of the new Ant project that should be executed. Defaults to the new project's default target.	No
output	Filename to write the ant output to. This is relative to the value of the dir attribute if it has been set or to the base directory of the current project otherwise.	No
inheritAll	If `true`, pass all properties to the new Ant project. Defaults to `true`.	No
inheritRefs	If `true`, pass all references to the new Ant project. Defaults to `false`.	No
useNativeBasedir	If set to true, the child build will use the same basedir as it would have used when run	No

	from the command line (i.e. the basedir one would expect when looking at the child build's buildfile). Defaults to `false`. *since Ant 1.8.0*	

Parameters specified as nested elements

property

See the description of the property task.
These properties become equivalent to properties you define on the command line. These are special properties and they will always get passed down, even through additional `<*ant*>` tasks with inheritall set to false (see above).
Note that the `refid` attribute points to a reference in the calling project, not in the new one.

reference

Used to choose references that shall be copied into the new project, optionally changing their id.

Attribute	Description	Required
refid	The id of the reference in the calling project.	Yes
torefid	The id of the reference in the new project.	No, defaults to the value of refid.

propertyset

You can specify a set of properties to be copied into the new project with propertysets.

since Ant 1.6.

target

You can specify multiple targets using nested `<target>` elements instead of using the target attribute. These will be executed as if Ant had been invoked with a single target whose dependencies are the targets so specified, in the order specified.

Attribute	Description	Required
name	The name of the called target.	Yes

since Ant 1.6.3.

Basedir of the new project

If you set `useNativeBasedir` to true, the basedir of the new project will be whatever the basedir attribute of the `<project>` element of the new project says (or the new project's directory if the there is no basedir attribute) - no matter what any other attribute of this task says and no matter how deeply nested into levels of `<ant>` invocations this task lives.

If you haven't set `useNativeBasedir` or set it to false, the following rules apply:

The basedir value of the new project is affected by the two attributes dir and inheritall as well as the `<ant>` task's history. The current behaviour is known to be confusing but cannot be changed without breaking backwards compatibility in subtle ways.

If the <ant> task is in a "top level" build file, i.e. the project containing the <ant> task has not itself been invoked as part of a different <ant> (or <antcall>) task "higher up", the following table shows the details:

dir attribute	inheritAll attribute	new project's basedir
value provided	true	value of dir attribute
value provided	false	value of dir attribute
omitted	true	basedir of calling project (the one whose build file contains the <ant> task).
omitted	false	basedir attribute of the <project> element of the new project

If on the other hand the <ant> task is already nested into another invocation, the parent invocation's settings affect the outcome of the basedir value. The current task's dir attribute will always win, but if the dir attribute has been omitted an even more complex situation arises:

parent dir attribute	parent inheritAll attribute	current inheritAll attribute	new project's basedir
value provided	any	any	value of parent's dir attribute
omitted	true	true	basedir of parent project (the one whose build file called the build file that contains the current <ant> task).
omitted	true	false	basedir of parent project (the one whose build file called the build file that contains the current <ant> task).
omitted	false	true	basedir of calling project (the one whose build file contains the current <ant> task).
omitted	false	false	basedir attribute of the <project> element of the new project

If you add even deeper levels of nesting, things get even more complicated and you need to apply the above table recursively.

If the basedir of the outer most build has been specified as a property on the command line (i.e. `-Dbasedir=some-value` or a `-propertyfile` argument) the value provided will get an even higher priority. For any <ant> task that doesn't specify a dir attribute, the new project's basedir will be the value specified on the command line - no matter how deeply nested into layers of build files the task may be.

The same happens if the basedir is specified as a nested <property> of an <ant> task. The basedir of build files started at deeper levels will be set to the specified value of the property element unless the corresponding Ant tasks set the dir attribute explicitly.

Examples

```
<ant antfile="subproject/subbuild.xml" target="compile"/>

<ant dir="subproject"/>

<ant antfile="subproject/property_based_subbuild.xml">
  <property name="param1" value="version 1.x"/>
  <property file="config/subproject/default.properties"/>
</ant>

<ant inheritAll="false" antfile="subproject/subbuild.xml">
  <property name="output.type" value="html"/>
</ant>
```

These lines invoke the same build file:

```
<ant antfile="sub1/sub2/build.xml" />
<ant antfile="sub2/build.xml" dir="sub1" />
<ant antfile="build.xml" dir="sub1/sub2" />
```

The build file of the calling project defines some `<path>` elements like this:

```
<path id="path1">
    ...
</path>
<path id="path2">
    ...
</path>
```

and the called build file (`subbuild.xml`) also defines a `<path>` with the id `path1`, but `path2` is not defined:

```
<ant antfile="subbuild.xml" inheritrefs="true"/>
```

will not override `subbuild`'s definition of `path1`, but make the parent's definition of `path2` available in the subbuild.

```
<ant antfile="subbuild.xml"/>
```

as well as

```
<ant antfile="subbuild.xml" inheritrefs="false"/>
```

will neither override `path1` nor copy `path2`.

```
<ant antfile="subbuild.xml" inheritrefs="false">
  <reference refid="path1"/>
</ant>
```

will override `subbuild`'s definition of `path1`.

```
<ant antfile="subbuild.xml" inheritrefs="false">
  <reference refid="path1" torefid="path2"/>
</ant>
```

will copy the parent's definition of `path1` into the new project using the id `path2`.

Local

Description

Adds a local property to the current scope. Property scopes exist at Ant's various "block" levels. These include targets as well as the Parallel and Sequential task containers (including Macrodef bodies). A local property at a given scope "shadows" properties of the same name at higher scopes, including the global scope. Note that using the Local task at the global level effectively makes the property local to the "anonymous target" in which top-level operations are carried out; it will not be defined for other targets in the buildfile. **Since Ant 1.8**

A property is made local if the `<local>` task preceedes its definition. See the examples section.

Parameters

Attribute	Description	Required
name	The property to declare in the current scope	Yes

Examples

Temporarily shadow a global property's value

```
    <property name="foo" value="foo"/>

    <target name="step1">
        <echo>Before local: foo is ${foo}</echo>
        <local name="foo"/>
        <property name="foo" value="bar"/>
        <echo>After local: foo is ${foo}</echo>
    </target>

    <target name="step2" depends="step1">
        <echo>In step2: foo is ${foo}</echo>
    </target>
```

outputs

```
step1:
     [echo] Before local: foo is foo
     [echo] After local: foo is bar

step2:
     [echo] In step2: foo is foo
```

here the local-task shadowed the global definition of `foo` for the remainder of the target step1.

Creating thread local properties

```
    <property name="foo" value="foo"/>

    <parallel>
        <echo>global 1: foo is ${foo}</echo>
        <sequential>
            <local name="foo"/>
            <property name="foo" value="bar.1"/>
            <echo>First sequential: foo is ${foo}</echo>
        </sequential>
        <sequential>
            <sleep seconds="1"/>
            <echo>global 2: foo is ${foo}</echo>
```

```
        </sequential>
        <sequential>
            <local name="foo"/>
            <property name="foo" value="bar.2"/>
            <echo>Second sequential: foo is ${foo}</echo>
        </sequential>
        <echo>global 3: foo is ${foo}</echo>
    </parallel>
```

outputs something similar to

```
     [echo] global 3: foo is foo
     [echo] global 1: foo is foo
     [echo] First sequential: foo is bar.1
     [echo] Second sequential: foo is bar.2
     [echo] global 2: foo is foo
```

Use inside macrodef

This probably is where local can be applied in the most useful way. If you needed a "temporary property" inside a macrodef in Ant prior to Ant 1.8.0 you had to try to come up with a property name that would be unique across macro invocations.

Say you wanted to write a macro that created the parent directory of a given file. A naive approach would be:

```
    <macrodef name="makeparentdir">
        <attribute name="file"/>
        <sequential>
            <dirname property="parent" file="@{file}"/>
            <mkdir dir="${parent}"/>
        </sequential>
    </macrodef>
    <makeparentdir file="some-dir/some-file"/>
```

but this would create a global property "parent" on the first invocation - and since properties are not mutable, any subsequent invocation will see the same value and try to create the same directory as the first invocation.

The recommendation prior to Ant 1.8.0 was to use a property name based on one of the macro's attributes, like

```
    <macrodef name="makeparentdir">
        <attribute name="file"/>
        <sequential>
            <dirname property="parent.@{file}" file="@{file}"/>
            <mkdir dir="${parent.@{file}}"/>
        </sequential>
    </macrodef>
```

Now invocations for different files will set different properties and the directories will get created. Unfortunately this "pollutes" the global properties space. In addition it may be hard to come up with unique names in some cases.

Enter `<local>`:

```
    <macrodef name="makeparentdir">
        <attribute name="file"/>
        <sequential>
            <local name="parent"/>
            <dirname property="parent" file="@{file}"/>
            <mkdir dir="${parent}"/>
        </sequential>
    </macrodef>
```

Each invocation gets its own property name "parent" and there will be no global property of that name at all.

Sequential

Description

Sequential is a container task - it can contain other Ant tasks. The nested tasks are simply executed in sequence. Sequential's primary use is to support the sequential execution of a subset of tasks within the parallel task

The sequential task has no attributes and does not support any nested elements apart from Ant tasks. Any valid Ant task may be embedded within the sequential task.

Example

```
<parallel>
  <wlrun ... >
  <sequential>
    <sleep seconds="30"/>
    <junit ... >
    <wlstop/>
  </sequential>
</parallel>
```

This example shows how the sequential task is used to execute three tasks in sequence, while another task is being executed in a separate thread.

MacroDef

Description

This defines a new task using a `<sequential>` nested task as a template. Nested elements `<attribute>` and `<element>` are used to specify attributes and elements of the new task. These get substituted into the `<sequential>` task when the new task is run.

Note

You can also use *prior defined* attributes for default-values in other attributes. See the examples.

since Ant 1.6

Parameters

Attribute	Description	Required
name	The name of the new definition.	Yes
uri	The uri that this definition should live in.	No
description	A description of the macrodef (for documentation purposes only).	No
backtrace	This controls the error traceback if they is an error detected when running the macro. If this is set to true, there will be an error trackback, if false there will not be one. *Since Ant 1.7.*	No; default *true*

Parameters specified as nested elements

attribute

This is used to specify attributes of the new task. The values of the attributes get substituted into the templated task. The attributes will be required attributes unless a default value has been set.

This attribute is placed in the body of the templated task using a notation similar to the ant property notation - @{attribute name}. (May be remembered as "put the substitution AT this location").

The escape sequence @@ is used to escape @. This allows @{x} to be placed in the text without substitution of x by using @@{x}. This corresponds to the $$ escape sequence for properties.

The case of the attribute is ignored, so @{myAttribute} is treated the same as @{MyAttribute}.

Parameters

Attribute	Description	Required
name	The name of the new attribute	Yes
default	The default value of the attribute.	No
description	This contains a description of the attribute. *since ant 1.6.1*	No

element

This is used to specify nested elements of the new task. The contents of the nested elements of the task instance are placed in the templated task at the tag name.

The case of the element name is ignored.

Parameters

Attribute	Description	Required
name	The name of the element	Yes
optional	If true this nested element is optional. Default is false - i.e the nested element is required in the new task.	No
implicit	If true this nested element is implicit. This means that any nested elements of the macrodef instance will be placed in the element indicated by the name of this element. There can only be one element if an element is implicit. The default value is false. *since ant 1.6.2*	No
description	This contains a description informing the user what the contents of the element are expected to be. *since ant 1.6.1*	No

text

This is used to specify the treatment of text contents of the macro invocation. If this element is not present, then any nested text in the macro invocation will be an error. If the text element is present, then the name becomes an attribute that gets set to the nested text of the macro invocation. *Since ant 1.6.1.*

The case of the text name is ignored.

Parameters

Attribute	Description	Required
name	The name of the text attribute	Yes
optional	If true nested text in the macro is optional, default is "false".	No
trim	If true, the nested text is trimmed of white space, default is "false".	No
description	This contains a description informing the user what the nested text of the macro is expected to be.	No

Examples

The following example defined a task called testing and runs it.

```
<macrodef name="testing">
   <attribute name="v" default="NOT SET"/>
   <element name="some-tasks" optional="yes"/>
   <sequential>
      <echo>v is @{v}</echo>
      <some-tasks/>
   </sequential>
</macrodef>

<testing v="This is v">
   <some-tasks>
      <echo>this is a test</echo>
```

```
   </some-tasks>
</testing>
```

The following fragment defines a task called `<call-cc>` which take the attributes "target", "link" and "target.dir" and the nested element "cc-elements". The body of the task uses the `<cc>` task from the ant-contrib project.

```
<macrodef name="call-cc">
   <attribute name="target"/>
   <attribute name="link"/>
   <attribute name="target.dir"/>
   <element name="cc-elements"/>
   <sequential>
      <mkdir dir="${obj.dir}/@{target}"/>
      <mkdir dir="@{target.dir}"/>
         <cc link="@{link}" objdir="${obj.dir}/@{target}"
             outfile="@{target.dir}/@{target}">
            <compiler refid="compiler.options"/>
            <cc-elements/>
         </cc>
      </sequential>
</macrodef>
```

This then can be used as follows:

```
<call-cc target="unittests" link="executable"
         target.dir="${build.bin.dir}">
   <cc-elements>
      <includepath location="${gen.dir}"/>
      <includepath location="test"/>
      <fileset dir="test/unittest" includes = "**/*.cpp"/>
      <fileset dir="${gen.dir}" includes = "*.cpp"/>
      <linker refid="linker-libs"/>
   </cc-elements>
</call-cc>
```

The following fragment shows <call-cc>, but this time using an implicit element and with the link and target.dir arguments having default values.

```
<macrodef name="call-cc">
   <attribute name="target"/>
   <attribute name="link" default="executable"/>
   <attribute name="target.dir" default="${build.bin.dir}"/>
   <element name="cc-elements" implicit="yes"/>
   <sequential>
      <mkdir dir="${obj.dir}/@{target}"/>
      <mkdir dir="@{target.dir}"/>
         <cc link="@{link}" objdir="${obj.dir}/@{target}"
             outfile="@{target.dir}/@{target}">
            <compiler refid="compiler.options"/>
            <cc-elements/>
         </cc>
      </sequential>
</macrodef>
```

This then can be used as follows, note that <cc-elements> is not specified.

```
<call-cc target="unittests">
   <includepath location="${gen.dir}"/>
   <includepath location="test"/>
   <fileset dir="test/unittest" includes = "**/*.cpp"/>
   <fileset dir="${gen.dir}" includes = "*.cpp"/>
   <linker refid="linker-libs"/>
</call-cc>
```

The following shows the use of the `text` element.

```
<macrodef name="echotest">
   <text name="text"/>
   <sequential>
      <echo>@{text}</echo>
   </sequential>
</macrodef>
<echotest>
   Hello world
```

```
</echotest>
```

The following uses a prior defined attribute for setting the default value of another. The output would be `one=test two=test`. If you change the order of lines *1 and *2 the output would be `one=test two=@{one}`, because while processing the *two*-line the value for *one* is not set.

```
<macrodef name="test">
   <attribute name="one"/>                        *1
   <attribute name="two" default="@{one}"/>     *2
   <sequential>
      <echo>one=@{one}   two=@{two}</echo>
   </sequential>
</macrodef>
<test one="test"/>
```

PropertyHelper

Description

This task is provided for the purpose of allowing the user to **(a)** install a different PropertyHelper at runtime, or **(b)** (hopefully more often) install one or more PropertyHelper Delegates into the PropertyHelper active on the current Project. This is somewhat advanced Ant usage and assumes a working familiarity with the modern Ant APIs. See the description of Ant's Property Helper for more information. **Since Ant 1.8.0**

Parameters specified as nested elements

PropertyHelper

You may specify exactly one configured `org.apache.tools.ant.PropertyHelper` instance.

PropertyHelper.Delegate

You may specify, either in conjunction with a new `PropertyHelper` or not, one or more configured implementations of the `org.apache.tools.ant.PropertyHelper.Delegate` interface. A deeper understanding of the API is required here, however, as `Delegate` is a marker interface only: the nested arguments must implement a `Delegate` subinterface in order to do anything meaningful.

delegate

A generic <delegate> element which can use project references is also provided:

Parameters

Attribute	Description	Required
refid	The *id* of a `PropertyHelper.Delegate` to install.	Yes

Examples

Install a completely different PropertyHelper implementation (assuming `MyPropertyHelper extends PropertyHelper`):

```
<componentdef classname="org.example.MyPropertyHelper"
              name="mypropertyhelper"/>
<propertyhelper>
  <mypropertyhelper/>
</propertyhelper>
```

Add a new PropertyEvaluator delegate (assuming `MyPropertyEvaluator implements PropertyHelper.PropertyEvaluator`). Note that PropertyHelper uses the configured delegates in LIFO order. I.e. the delegate added by this task will be consulted before any previously defined delegate and in particular before the built-in ones.

```
<componentdef classname="org.example.MyPropertyEvaluator"
              name="mypropertyevaluator"/>
<propertyhelper>
  <mypropertyevaluator/>
</propertyhelper>
```

Add a new PropertyEvaluator delegate using the refid syntax:

```
<typedef classname="org.example.MyPropertyEvaluator"
         name="mypropertyevaluator"/>
<mypropertyevaluator id="evaluator"/>
<propertyhelper>
  <delegate refid="evaluator"/>
</propertyhelper>
```

NetRexxC

Description

Compiles a NetRexx source tree within the running (Ant) VM.

The source and destination directory will be recursively scanned for NetRexx source files to compile. Only NetRexx files that have no corresponding class file or where the class file is older than the java file will be compiled.

Files in the source tree are copied to the destination directory, allowing support files to be located properly in the classpath. The source files are copied because the NetRexx compiler cannot produce class files in a specific directory via parameters

The directory structure of the source tree should follow the package hierarchy.

It is possible to refine the set of files that are being compiled/copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

All properties except classpath, srcdir and destDir are also available as properties in the form `ant.netrexxc.`*`attributename`*, eg.

```
<property name="ant.netrexxc.verbose" value="noverbose"/>
```

or from the command line as

```
ant -Dant.netrexxc.verbose=noverbose ...
```

Parameters

Attribute	Description	Required
binary	Whether literals are treated as the java binary type rather than the NetRexx types	No
classpath	The classpath to use during compilation	No
comments	Whether comments are passed through to the generated java source	No
compact	Whether error messages come out in compact or verbose format. Default is the compact format.	No
compile	Whether the NetRexx compiler should compile the generated java code	No
console	Whether or not messages should be displayed on the 'console'. Note that this task will rely on the default value for filtering compile messages.	No
crossref	Whether variable cross references are generated	No
decimal	Whether decimal arithmetic should be used for the NetRexx code. Setting this to off will report decimal arithmetic as an error, for performance critical applications.	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no").	No

	Default excludes are used when omitted.	
destDir	the destination directory into which the NetRexx source files should be copied and then compiled	Yes
diag	Whether diagnostic information about the compile is generated	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
explicit	Whether variables must be declared explicitly before use	No
format	Whether the generated java code is formatted nicely or left to match NetRexx line numbers for call stack debugging	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
java	Whether the generated java code is produced	No
keep	Sets whether the generated java source file should be kept after compilation. The generated files will have an extension of .java.keep, **not** .java. Use removeKeepExtension to change that.	No
logo	Whether the compiler text logo is displayed when compiling	No
removeKeepExtension	Tells wether the trailing .keep in nocompile-mode should be removed so that the resulting java source really ends on .java. This facilitates the use of the javadoc tool lateron.	No
replace	Whether the generated .java file should be replaced when compiling	No
savelog	Whether the compiler messages will be written to NetRexxC.log as well as to the console	No
sourcedir	Tells the NetRexx compiler to store the class files in the same directory as the source files. The alternative is the working directory	No
srcDir	Set the source dir to find the source NetRexx files	Yes
strictargs	Tells the NetRexx compiler that method calls always need parentheses, even if no arguments are needed, e.g. `aStringVar.getBytes` vs. `aStringVar.getBytes()`	No
strictassign	Tells the NetRexx compile that assignments must match exactly on type	No
strictcase	Specifies whether the NetRexx compiler should be case sensitive or not	No
strictimport	Whether classes need to be imported explicitly using an `import` statement. By default the NetRexx compiler will import certain packages automatically	No
strictprops	Whether local properties need to be qualified explicitly using `this`	No
strictsignal	Whether the compiler should force catching of exceptions by explicitly named types	No
symbols	Whether debug symbols should be generated into the class file	No
time	Asks the NetRexx compiler to print compilation times to the console	No
trace	Turns on or off tracing and directs the resultant trace output	No

utf8	Tells the NetRexx compiler that the source is in UTF8	No
verbose	Whether lots of warnings and error messages should be generated	No
suppressMethodArgumentNotUsed	Tells whether we should filter out the &Method argument not used& messages in strictargs mode.	No
suppressPrivatePropertyNotUsed	Tells whether we should filter out the &Private Property defined, but not used& messages in strictargs mode.	No
suppressVariableNotUsed	Tells whether we should filter out the &Variable set but not used& messages in strictargs mode. Please be careful with this one, as you can hide errors behind it!	No
suppressExceptionNotSignalled	Tells whether we should filter out the &Exception is declared, but not signaled within the method& messages in strictsignal mode.	No
suppressDeprecation	Tells whether we should filter out any deprecation-messages of the compiler out.	No

Examples

```
<netrexxc srcDir="/source/project" includes="vnr/util/*" destDir="/source/project/build"
classpath="/source/project2/proj.jar" comments="true" crossref="false" replace="true"
keep="true"/>
```

Ant EJB Tasks User Manual

by

- Paul Austin (p_d_austin@yahoo.com)
- Holger Engels (hengels@innovidata.com)
- Tim Fennell (tfenne@rcn.com)
- Martin Gee (martin.gee@icsynergy.com)
- Conor MacNeill
- Cyrille Morvan (cmorvan@ingenosya.com)
- Greg Nelson (gn@sun.com)
- Rob van Oostrum(rob@springwellfarms.ca)

Table of Contents

Introduction

Ant provides a number of optional tasks for developing 1.x and 2.x Enterprise Java Beans (EJBs). In general these tasks are specific to the particular vendor's EJB Server.

The tasks support:

- Borland Application Server 4.5
- iPlanet Application Server 6.0
- JBoss 2.1 and above EJB servers
- Weblogic 4.5.1 through to 7.0 EJB servers
- JOnAS 2.4.x and 2.5 Open Source EJB server
- IBM WebSphere 4.0

Vendors such as BEA and IBM now provide custom Ant tasks to work with their particular products. More importantly, EJB3.0 renders this whole process obsolete. Accordingly, developement of these tasks is effectively frozen. Bug reports and especially patches are welcome, but there is no pressing need to add support for new application servers. Nobody should be writing new EJB2.x applications and definitely not new EJB2.x servers.

EJB Tasks

Task	Application Servers
blgenclient	Borland Application Server 4.5 and 5.x

ddcreator	Weblogic 4.5.1	
ejbc	Weblogic 4.5.1	
iplanet-ejbc	iPlanet Application Server 6.0	
ejbjar	**Nested Elements**	
	borland	Borland Application Server 4.5 and 5.x
	iPlanet	iPlanet Application Server 6.0
	jboss	JBoss
	jonas	JOnAS 2.4.x and 2.5
	weblogic	Weblogic 5.1 to 7.0
	websphere	IBM WebSphere 4.0
wlrun	Weblogic 4.5.1 to 7.0	
wlstop	Weblogic 4.5.1 to 7.0	

---

ddcreator

Description:

ddcreator will compile a set of Weblogic text-based deployment descriptors into a serialized EJB deployment descriptor. The selection of which of the text-based descriptors are to be compiled is based on the standard Ant include and exclude selection mechanisms.

Parameters:

Attribute	Description	Required
descriptors	This is the base directory from which descriptors are selected.	Yes
dest	The directory where the serialized deployment descriptors will be written	Yes
classpath	This is the classpath to use to run the underlying weblogic ddcreator tool. This must include the `weblogic.ejb.utils.DDCreator` class	No

Examples

```
<ddcreator descriptors="${dd.dir}"
           dest="${gen.classes}"
           classpath="${descriptorbuild.classpath}">
  <include name="*.txt"/>
</ddcreator>
```

---

ejbc

Description:

The ejbc task will run Weblogic's ejbc tool. This tool will take a serialized deployment descriptor, examine the various EJB interfaces and bean classes and then generate the required support classes necessary to deploy the bean in a Weblogic EJB container. This will include the RMI stubs and skeletons as well as the classes which implement the bean's home and remote interfaces.

The ant task which runs this tool is able to compile several beans in a single operation. The beans to be compiled are selected by including their serialized deployment descriptors. The standard ant `include` and `exclude` constructs can be used to select the deployment descriptors to be included.

Each descriptor is examined to determine whether the generated classes are out of date and need to be regenerated. The deployment descriptor is de-serialized to discover the home, remote and implementation classes. The corresponding source files are determined and checked to see their modification times. These times and the modification time of the serialized descriptor itself are compared with the modification time of the generated classes. If the generated classes are not present or are out of date, the ejbc tool is run to generate new versions.

Parameters:

Attribute	Description	Required
descriptors	This is the base directory from which the serialized deployment descriptors are selected.	Yes
dest	The base directory where the generated classes, RIM stubs and RMI skeletons are written	Yes
manifest	The name of a manifest file to be written. This manifest will contain an entry for each EJB processed	Yes
src	The base directory of the source tree containing the source files of the home interface, remote interface and bean implementation classes.	Yes
classpath	This classpath must include both the `weblogic.ejbc` class and the class files of the bean, home interface, remote interface, etc of the bean being processed.	No
keepgenerated	Controls whether ejbc will keep the intermediate Java files used to build the class files. This can be useful when debugging.	No, defaults to false.

Examples

```
<ejbc descriptors="${gen.classes}"
           src="${src.dir}"
           dest="${gen.classes}"
           manifest="${build.manifest}"
           classpath="${descriptorbuild.classpath}">
  <include name="*.ser"/>
</ejbc>
```

---

iplanet-ejbc

Description:

Task to compile EJB stubs and skeletons for the iPlanet Application Server 6.0. Given a standard EJB 1.1 XML descriptor as well as an iAS-specific EJB descriptor, this task will generate the stubs and skeletons required to deploy

the EJB to iAS. Since the XML descriptors can include multiple EJBs, this is a convenient way of specifying many EJBs in a single Ant task.

For each EJB specified, the task will locate the three classes that comprise the EJB in the destination directory. If these class files cannot be located in the destination directory, the task will fail. The task will also attempt to locate the EJB stubs and skeletons in this directory. If found, the timestamps on the stubs and skeletons will be checked to ensure they are up to date. Only if these files cannot be found or if they are out of date will the iAS ejbc utility be called to generate new stubs and skeletons.

Parameters:

Attribute	Description	Required
ejbdescriptor	Standard EJB 1.1 XML descriptor (typically titled "ejb-jar.xml").	Yes
iasdescriptor	iAS-specific EJB XML descriptor (typically titled "ias-ejb-jar.xml").	Yes
dest	The is the base directory where the RMI stubs and skeletons are written. In addition, the class files for each bean (home interface, remote interface, and EJB implementation) must be found in this directory.	Yes
classpath	The classpath used when generating EJB stubs and skeletons. If omitted, the classpath specified when Ant was started will be used. Nested "classpath" elements may also be used.	No
keepgenerated	Indicates whether or not the Java source files which are generated by ejbc will be saved or automatically deleted. If "yes", the source files will be retained. If omitted, it defaults to "no".	No
debug	Indicates whether or not the ejbc utility should log additional debugging statements to the standard output. If "yes", the additional debugging statements will be generated. If omitted, it defaults to "no".	No
iashome	May be used to specify the "home" directory for this iAS installation. This is used to find the ejbc utility if it isn't included in the user's system path. If specified, it should refer to the "[install-location]/iplanet/ias6/ias" directory. If omitted, the ejbc utility must be on the user's system path.	No

Examples

```
<iplanet-ejbc ejbdescriptor="ejb-jar.xml"
              iasdescriptor="ias-ejb-jar.xml"
              dest="${build.classesdir}"
              classpath="${ias.ejbc.cpath}"/>


<iplanet-ejbc ejbdescriptor="ejb-jar.xml"
              iasdescriptor="ias-ejb-jar.xml"
              dest="${build.classesdir}"
              keepgenerated="yes"
              debug="yes"
              iashome="${ias.home}">
              <classpath>
                  <pathelement path="."/>
                  <pathelement path="${build.classpath}"/>
              </classpath>
</iplanet-ejbc>
```

---

wlrun

Description:

The `wlrun` task is used to start a weblogic server. The task runs a weblogic instance in a separate Java Virtual Machine. A number of parameters are used to control the operation of the weblogic instance. Note that the task, and hence ant, will not complete until the weblogic instance is stopped.

Parameters:

Attribute	Description	Required for 4.5.1 and 5.1	Required for 6.0
BEA Home	The location of the BEA Home where the server's config is defined. If this attribute is present, wlrun assumes that the server will be running under Weblogic 6.0	N/A	Yes
home	The location of the weblogic home that is to be used. This is the location where weblogic is installed.	Yes	Yes. Note this is the absolute location, not relative to BEA home.
Domain	The domain to which the server belongs.	N/A	Yes
classpath	The classpath to be used with the Java Virtual Machine that runs the Weblogic Server. Prior to Weblogic 6.0, this is typically set to the Weblogic boot classpath. Under Weblogic 6.0 this should include all the weblogic jars	Yes	Yes
wlclasspath	The weblogic classpath used by the Weblogic Server.	No	N/A
properties	The name of the server's properties file within the weblogic home directory used to control the weblogic instance.	Yes	N/A
name	The name of the weblogic server within the weblogic home which is to be run. This defaults to "myserver"	No	No
policy	The name of the security policy file within the weblogic home directory that is to be used. If not specified, the default policy file `weblogic.policy` is used.	No	No
username	The management username used to manage the server	N/A	No
password	The server's management password	N/A	Yes
pkPassword	The private key password so the server can decrypt the SSL private key file	N/A	No
jvmargs	Additional argument string passed to the Java Virtual Machine used to run the Weblogic instance.	No	No
weblogicMainClass	name of the main class for weblogic	No	No

Nested Elements

The wlrun task supports nested `<classpath>` and `<wlclasspath>` elements to set the respective classpaths.

Examples

This example shows the use of wlrun to run a server under Weblogic 5.1

```
    <wlrun taskname="myserver"
           classpath="${weblogic.boot.classpath}"
           wlclasspath="${weblogic.classes}:${code.jars}"
           name="myserver"
           home="${weblogic.home}"
           properties="myserver/myserver.properties"/>
```

This example shows wlrun being used to run the petstore server under Weblogic 6.0

```
    <wlrun taskname="petstore"
           classpath="${weblogic.classes}"
           name="petstoreServer"
           domain="petstore"
           home="${weblogic.home}"
           password="petstorePassword"
           beahome="${bea.home}"/>
```

---

wlstop

Description:

The `wlstop` task is used to stop a weblogic instance which is currently running. To shut down an instance you must supply both a username and a password. These will be stored in the clear in the build script used to stop the instance. For security reasons, this task is therefore only appropriate in a development environment.

This task works for most version of Weblogic, including 6.0. You need to specify the BEA Home to have this task work correctly under 6.0

Parameters:

Attribute	Description	Required
BEAHome	This attribute selects Weblogic 6.0 shutdown.	No
classpath	The classpath to be used with the Java Virtual Machine that runs the Weblogic Shutdown command.	Yes
user	The username of the account which will be used to shutdown the server	Yes
password	The password for the account specified in the user parameter.	Yes
url	The URL which describes the port to which the server is listening for T3 connections. For example, t3://localhost:7001	Yes
delay	The delay in seconds after which the server will stop. This defaults to an immediate shutdown.	No

Nested Element

The classpath of the wlstop task can be set by a `<classpath>` nested element.

Examples

This example show the shutdown for a Weblogic 6.0 server

```
    <wlstop classpath="${weblogic.classes}"
            user="system"
```

```
            url="t3://localhost:7001"
            password="foobar"
            beahome="${bea.home}"/>
```

---

ejbjar

Description:

This task is designed to support building of EJB jar files (EJB 1.1 & 2.0). Support is currently provided for 'vanilla' EJB jar files - i.e. those containing only the user generated class files and the standard deployment descriptor. Nested elements provide support for vendor specific deployment tools. These currently include:

- Borland Application Server 4.5
- iPlanet Application Server 6.0
- JBoss 2.1 and above
- Weblogic 5.1/6.0 session/entity beans using the weblogic.ejbc tool
- IBM WebSphere 4.0
- TOPLink for WebLogic 2.5.1-enabled entity beans
- JOnAS 2.4.x and 2.5 Open Source EJB server

The task works as a directory scanning task, and performs an action for each deployment descriptor found. As such the includes and excludes should be set to ensure that all desired EJB descriptors are found, but no application server descriptors are found. For each descriptor found, ejbjar will parse the deployment descriptor to determine the necessary class files which implement the bean. These files are assembled along with the deployment descriptors into a well formed EJB jar file. Any support files which need to be included in the generated jar can be added with the `<support>` nested element. For each class included in the jar, ejbjar will scan for any super classes or super interfaces. These will be added to the generated jar.

If no nested vendor-specific deployment elements are present, the task will simply generate a generic EJB jar. Such jars are typically used as the input to vendor-specific deployment tools. For each nested deployment element, a vendor specific deployment tool is run to generate a jar file ready for deployment in that vendor's EJB container.

The jar files are only built if they are out of date. Each deployment tool element will examine its target jar file and determine if it is out of date with respect to the class files and deployment descriptors that make up the bean. If any of these files are newer than the jar file the jar will be rebuilt otherwise a message is logged that the jar file is up to date.

The task uses the jakarta-BCEL framework to extract all dependent classes. This means that, in addition to the classes that are mentioned in the deployment descriptor, any classes that these depend on are also automatically included in the jar file.

Naming Convention

Ejbjar handles the processing of multiple beans, and it uses a set of naming conventions to determine the name of the generated EJB jars. The naming convention that is used is controlled by the "naming" attribute. It supports the following values

- descriptor

  This is the default naming scheme. The name of the generated bean is derived from the name of the deployment descriptor. For an Account bean, for example, the deployment descriptor would be named `Account-ejb-jar.xml`. Vendor specific descriptors are located using the same naming convention. The weblogic bean, for

  `Account-weblogic-ejb-jar.xml`

example, would be named . Under this arrangement, the deployment descriptors can be separated from the code implementing the beans, which can be useful when the same bean code is deployed in separate beans.

This scheme is useful when you are using one bean per EJB jar and where you may be deploying the same bean classes in different beans, with different deployment characteristics.

- ejb-name

  This naming scheme uses the `<ejb-name>` element from the deployment descriptor to determine the bean name. In this situation, the descriptors normally use the generic descriptor names, such as `ejb-jar.xml` along with any associated vendor specific descriptor names. For example, If the value of the `<ejb-name>` were to be given in the deployment descriptor as follows:

```
<ejb-jar>
    <enterprise-beans>
        <entity>
            <ejb-name>Sample</ejb-name>
            <home>org.apache.ant.ejbsample.SampleHome</home>
```

  then the name of the generated bean would be `Sample.jar`

  This scheme is useful where you want to use the standard deployment descriptor names, which may be more compatible with other EJB tools. This scheme must have one bean per jar.

- directory

  In this mode, the name of the generated bean jar is derived from the directory containing the deployment descriptors. Again the deployment descriptors typically use the standard filenames. For example, if the path to the deployment descriptor is `/home/user/dev/appserver/dd/sample`, then the generated bean will be named `sample.jar`

  This scheme is also useful when you want to use standard style descriptor names. It is often most useful when the descriptors are located in the same directory as the bean source code, although that is not mandatory. This scheme can handle multiple beans per jar.

- basejarname

  The final scheme supported by the `<ejbjar>` task is used when you want to specify the generated bean jar name directly. In this case the name of the generated jar is specified by the "basejarname" attribute. Since all generated beans will have the same name, this task should be only used when each descriptor is in its own directory.

  This scheme is most appropriate when you are using multiple beans per jar and only process a single deployment descriptor. You typically want to specify the name of the jar and not derive it from the beans in the jar.

Dependencies

In addition to the bean classes, ejbjar is able to ad additional classes to the generated ejbjar. These classes are typically the support classes which are used by the bean's classes or as parameters to the bean's methods.

In versions of Ant prior to 1.5, ejbjar used reflection and attempted to add the super classes and super interfaces of the bean classes. For this technique to work the bean classes had to be loaded into Ant's JVM. This was not always possible due to class dependencies.

The ejbjar task in Ant releases 1.5 and later uses the jakarta-BCEL library to analyze the bean's class files directly, rather than loading them into the JVM. This also allows ejbjar to add all of the required support classes for a bean and not just super classes.

In Ant 1.5, a new attribute, `dependency` has been introduced to allow the buildfile to control what additional classes are added to the generated jar. It takes three possible values

- `none` - only the bean classes and interfaces described in the bean's descriptor are added to the jar.
- `super` - this is the default value and replicates the original ejbjar behaviour where super classes and super interfaces are added to the jar
- `full` - In this mode all classes used by the bean's classes and interfaces are added to the jar

The `super` and `full` values require the jakarta-BCEL library to be available. If it is not, ejbjar will drop back to the behaviour corresponding to the value `none`.

Parameters:

Attribute	Description	Required
descriptordir	The base directory under which to scan for EJB deployment descriptors. If this attribute is not specified, then the deployment descriptors must be located in the directory specified by the 'srcdir' attribute.	No
srcdir	The base directory containing the .class files that make up the bean. Included are the home- remote- pk- and implementation- classes and all classes, that these depend on. Note that this can be the same as the descriptordir if all files are in the same directory tree.	Yes
destdir	The base directory into which generated jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace. Note that this attribute is only used if the task is generating generic jars (i.e. no vendor-specific deployment elements have been specified).	Yes, unless vendor-specific deployment elements have been specified.
cmpversion	Either `1.0` or `2.0`. Default is `1.0`. A CMP 2.0 implementation exists currently only for JBoss.	No
naming	Controls the naming convention used to name generated EJB jars. Please refer to the description above.	No
basejarname	The base name that is used for the generated jar files. If this attribute is specified, the generic jar file name will use this value as the prefix (followed by the value specified in the 'genericjarsuffix' attribute) and the resultant ejb jar file (followed by any suffix specified in the nested element).	No
basenameterminator	String value used to substring out a string from the name of each deployment descriptor found, which is then used to locate related deployment descriptors (e.g. the WebLogic descriptors). For example, a basename of '.' and a deployment descriptor called 'FooBean.ejb-jar.xml' would result in a basename of 'FooBean' which would then be used to find FooBean.weblogic-ejb-jar.xml and FooBean.weblogic-cmp-rdbms-jar.xml, as well as to create the filenames of the jar files as FooBean-generic.jar and FooBean-wl.jar. This attribute is not used if the 'basejarname' attribute is specified.	No, defaults to '-'.
genericjarsuffix	String value appended to the basename of the deployment descriptor to create the filename of the generic EJB jar file.	No, defaults to '-generic.jar'.

classpath	This classpath is used when resolving classes which are to be added to the jar. Typically nested deployment tool elements will also support a classpath which will be combined with this classpath when resolving classes	No.
flatdestdir	Set this attribute to true if you want all generated jars to be placed in the root of the destdir, rather than according to the location of the deployment descriptor within the descriptor dir hierarchy.	No.
dependency	This attribute controls which additional classes and interfaces are added to the jar. Please refer to the description above	No.
manifest	the manifest file to use, if any.	No

Nested Elements

In addition to the vendor specific nested elements, the ejbjar task provides three nested elements.

Classpath

The `<classpath>` nested element allows the classpath to be set. It is useful when setting the classpath from a reference path. In all other respects the behaviour is the same as the classpath attribute.

dtd

The `<dtd>` element is used to specify the local location of DTDs to be used when parsing the EJB deployment descriptor. Using a local DTD is much faster than loading the DTD across the net. If you are running ejbjar behind a firewall you may not even be able to access the remote DTD. The supported vendor-specific nested elements know the location of the required DTDs within the vendor class hierarchy and, in general, this means `<dtd>` elements are not required. It does mean, however, that the vendor's class hierarchy must be available in the classpath when Ant is started. If your want to run Ant without requiring the vendor classes in the classpath, you would need to use a `<dtd>` element.

Attribute	Description	Required
publicId	The public Id of the DTD for which the location is being provided	Yes
location	The location of the local copy of the DTD. This can either be a file or a resource loadable from the classpath.	Yes

support

The `<support>` nested element is used to supply additional classes (files) to be included in the generated jars. The `<support>` element is a FileSet, so it can either reference a fileset declared elsewhere or it can be defined in-place with the appropriate `<include>` and `<exclude>` nested elements. The files in the support fileset are added into the generated EJB jar in the same relative location as their location within the support fileset. Note that when ejbjar generates more than one jar file, the support files are added to each one.

Vendor-specific deployment elements

Each vendor-specific nested element controls the generation of a deployable jar specific to that vendor's EJB container. The parameters for each supported deployment element are detailed here.

Jboss element

The jboss element searches for the JBoss specific deployment descriptors and adds them to the final ejb jar file. JBoss has two deployment descriptors:

- jboss.xml
- for container manager persistence:

CMP version	File name
CMP 1.0	jaws.xml
CMP 2.0	jbosscmp-jdbc.xml

. The JBoss server uses hot deployment and does not require compilation of additional stubs and skeletons.

Attribute	Description	Required
destdir	The base directory into which the generated weblogic ready jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace.	Yes
genericjarsuffix	A generic jar is generated as an intermediate step in build the weblogic deployment jar. The suffix used to generate the generic jar file is not particularly important unless it is desired to keep the generic jar file. It should not, however, be the same as the suffix setting.	No, defaults to '-generic.jar'.
suffix	String value appended to the basename of the deployment descriptor to create the filename of the JBoss EJB jar file.	No, defaults to '.jar'.
keepgeneric	This controls whether the generic file used as input to ejbc is retained.	No, defaults to false

Weblogic element

The weblogic element is used to control the weblogic.ejbc compiler for generating weblogic EJB jars. Prior to Ant 1.3, the method of locating CMP descriptors was to use the ejbjar naming convention. So if your ejb-jar was called, Customer-ejb-jar.xml, your weblogic descriptor was called Customer- weblogic-ejb-jar.xml and your CMP descriptor had to be Customer-weblogic-cmp- rdbms-jar.xml. In addition, the `<type-storage>` element in the weblogic descriptor had to be set to the standard name META-INF/weblogic-cmp-rdbms- jar.xml, as that is where the CMP descriptor was mapped to in the generated jar.

There are a few problems with this scheme. It does not allow for more than one CMP descriptor to be defined in a jar and it is not compatible with the deployment descriptors generated by some tools.

In Ant 1.3, ejbjar parses the weblogic deployment descriptor to discover the CMP descriptors, which are then included automatically. This behaviour is controlled by the newCMP attribute. Note that if you move to the new method of determining CMP descriptors, you will need to update your weblogic deployment descriptor's `<type-storage>` element. In the above example, you would define this as META-INF/Customer-weblogic-cmp-rdbms-jar.xml.

Attribute	Description	Required
destdir	The base directory into which the generated weblogic ready jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace.	Yes
genericjarsuffix	A generic jar is generated as an intermediate step in build the weblogic deployment jar. The suffix used to generate the generic jar file is not particularly important unless it is	No, defaults to

	desired to keep the generic jar file. It should not, however, be the same as the suffix setting.	'-generic.jar'.
suffix	String value appended to the basename of the deployment descriptor to create the filename of the WebLogic EJB jar file.	No, defaults to '.jar'.
classpath	The classpath to be used when running the weblogic ejbc tool. Note that this tool typically requires the classes that make up the bean to be available on the classpath. Currently, however, this will cause the ejbc tool to be run in a separate VM	No
wlclasspath	Weblogic 6.0 will give a warning if the home and remote interfaces of a bean are on the system classpath used to run weblogic.ejbc. In that case, the standard weblogic classes should be set with this attribute (or equivalent nested element) and the home and remote interfaces located with the standard classpath attribute	No
keepgeneric	This controls whether the generic file used as input to ejbc is retained.	No, defaults to false
compiler	This allows for the selection of a different compiler to be used for the compilation of the generated Java files. This could be set, for example, to Jikes to compile with the Jikes compiler. If this is not set and the `build.compiler` property is set to jikes, the Jikes compiler will be used. If this is not desired, the value "`default`" may be given to use the default compiler	No
rebuild	This flag controls whether weblogic.ejbc is always invoked to build the jar file. In certain circumstances, such as when only a bean class has been changed, the jar can be generated by merely replacing the changed classes and not rerunning ejbc. Setting this to false will reduce the time to run ejbjar.	No, defaults to true.
keepgenerated	Controls whether weblogic will keep the generated Java files used to build the class files added to the jar. This can be useful when debugging	No, defaults to false.
args	Any additional arguments to be passed to the weblogic.ejbc tool.	No.
weblogicdtd	**Deprecated**. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested `<dtd>` element, described above. If you do choose to use an attribute, you should use a nested `<dtd>` element.	No.
wldtd	**Deprecated**. Defines the location of the weblogic-ejb-jar DTD which covers the Weblogic specific deployment descriptors. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested `<dtd>` element, described above.	No.
ejbdtd	**Deprecated**. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested `<dtd>` element, described above.	No.
newCMP	If this is set to true, the new method for locating CMP descriptors will be used.	No. Defaults to false
oldCMP	**Deprecated** This is an antonym for newCMP which should be used instead.	No.
noEJBC	If this attribute is set to true, Weblogic's ejbc will not be run on the EJB jar. Use this if you prefer to run ejbc at deployment time.	No.
ejbcclass	Specifies the classname of the ejbc compiler. Normally ejbjar determines the appropriate class based on the DTD used for the EJB. The EJB 2.0 compiler featured in weblogic 6 has, however, been deprecated in version 7. When using with version 7 this attribute should be set to "weblogic.ejbc" to avoid the deprecation warning.	No.

jvmargs	Any additional arguments to be passed to the Virtual Machine running weblogic.ejbc tool. For example to set the memory size, this could be jvmargs="-Xmx128m"	No.
jvmdebuglevel	Sets the weblogic.StdoutSeverityLevel to use when running the Virtual Machine that executes ejbc. Set to 16 to avoid the warnings about EJB Home and Remotes being in the classpath	No.
outputdir	If set ejbc will be given this directory as the output destination rather than a jar file. This allows for the generation of "exploded" jars.	No.

The weblogic nested element supports three nested elements. The first two, `<classpath>` and `<wlclasspath>`, are used to set the respective classpaths. These nested elements are useful when setting up class paths using reference Ids. The last, `<sysproperty>`, allows Java system properties to be set during the compiler run. This turns out to be necessary for supporting CMP EJB compilation in all environments.

TOPLink for Weblogic element

Deprecated

The toplink element is no longer required. Toplink beans can now be built with the standard weblogic element, as long as the newCMP attribute is set to "true"

The TopLink element is used to handle beans which use Toplink for the CMP operations. It is derived from the standard weblogic element so it supports the same set of attributes plus these additional attributes

Attribute	Description	Required
toplinkdescriptor	This specifies the name of the TOPLink deployment descriptor file contained in the 'descriptordir' directory.	Yes
toplinkdtd	This specifies the location of the TOPLink DTD file. This can be a file path or a file URL. This attribute is not required, but using a local DTD is recommended.	No, defaults to dtd file at www.objectpeople.com.

Examples

This example shows ejbjar being used to generate deployment jars using a Weblogic EJB container. This example requires the naming standard to be used for the deployment descriptors. Using this format will create a ejb jar file for each variation of '*-ejb-jar.xml' that is found in the deployment descriptor directory.

```
    <ejbjar srcdir="${build.classes}"
            descriptordir="${descriptor.dir}">
      <weblogic destdir="${deploymentjars.dir}"
                classpath="${descriptorbuild.classpath}"/>
      <include name="**/*-ejb-jar.xml"/>
      <exclude name="**/*weblogic*.xml"/>
    </ejbjar>
```

If weblogic is not in the Ant classpath, the following example shows how to specify the location of the weblogic DTDs. This example also show the use of a nested classpath element.

```
    <ejbjar descriptordir="${src.dir}" srcdir="${build.classes}">
       <weblogic destdir="${deployment.webshop.dir}"
                 keepgeneric="true"
                 args="-g -keepgenerated ${ejbc.compiler}"
                 suffix=".jar"
                 oldCMP="false">
         <classpath>
           <pathelement path="${descriptorbuild.classpath}"/>
         </classpath>
       </weblogic>
```

```
      <include name="**/*-ejb-jar.xml"/>
      <exclude name="**/*-weblogic-ejb-jar.xml"/>
      <dtd publicId="-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
           location="${weblogic.home}/classes/weblogic/ejb/deployment/xml/ejb-jar.dtd"/>
      <dtd publicId="-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN"
           location="${weblogic.home}/classes/weblogic/ejb/deployment/xml/weblogic-ejb-jar.dtd"/>
   </ejbjar>
```

This example shows ejbjar being used to generate a single deployment jar using a Weblogic EJB container. This example does not require the deployment descriptors to use the naming standard. This will create only one ejb jar file - 'TheEJBJar.jar'.

```
   <ejbjar srcdir="${build.classes}"
           descriptordir="${descriptor.dir}"
           basejarname="TheEJBJar">
     <weblogic destdir="${deploymentjars.dir}"
               classpath="${descriptorbuild.classpath}"/>
     <include name="**/ejb-jar.xml"/>
     <exclude name="**/weblogic*.xml"/>
   </ejbjar>
```

This example shows ejbjar being used to generate deployment jars for a TOPLink-enabled entity bean using a Weblogic EJB container. This example does not require the deployment descriptors to use the naming standard. This will create only one TOPLink-enabled ejb jar file - 'Address.jar'.

```
   <ejbjar srcdir="${build.dir}"
           destdir="${solant.ejb.dir}"
           descriptordir="${descriptor.dir}"
           basejarname="Address">
           <weblogictoplink destdir="${solant.ejb.dir}"
                   classpath="${java.class.path}"
                   keepgeneric="false"
                   toplinkdescriptor="Address.xml"
                   toplinkdtd="file:///dtdfiles/toplink-cmp_2_5_1.dtd"
                   suffix=".jar"/>
           <include name="**/ejb-jar.xml"/>
           <exclude name="**/weblogic-ejb-jar.xml"/>
   </ejbjar>
```

This final example shows how you would set-up ejbjar under Weblogic 6.0. It also shows the use of the `<support>` element to add support files

```
   <ejbjar descriptordir="${dd.dir}" srcdir="${build.classes.server}">
      <include name="**/*-ejb-jar.xml"/>
      <exclude name="**/*-weblogic-ejb-jar.xml"/>
      <support dir="${build.classes.server}">
           <include name="**/*.class"/>
      </support>
      <weblogic destdir="${deployment.dir}"
                keepgeneric="true"
                suffix=".jar"
                rebuild="false">
        <classpath>
           <pathelement path="${build.classes.server}"/>
        </classpath>
        <wlclasspath>
           <pathelement path="${weblogic.classes}"/>
        </wlclasspath>
      </weblogic>
   </ejbjar>
```

WebSphere element

The websphere element searches for the websphere specific deployment descriptors and adds them to the final ejb jar file. Websphere has two specific descriptors for session beans:

- ibm-ejb-jar-bnd.xmi
- ibm-ejb-jar-ext.xmi

and another two for container managed entity beans:

- Map.mapxmi
- Schema.dbxmi

In terms of WebSphere, the generation of container code and stubs is called `deployment`. This step can be performed by the websphere element as part of the jar generation process. If the switch `ejbdeploy` is on, the ejbdeploy tool from the websphere toolset is called for every ejb-jar. Unfortunately, this step only works, if you use the ibm jdk. Otherwise, the rmic (called by ejbdeploy) throws a ClassFormatError. Be sure to switch ejbdeploy off, if run ant with sun jdk.

For the websphere element to work, you have to provide a complete classpath, that contains all classes, that are required to reflect the bean classes. For ejbdeploy to work, you must also provide the classpath of the ejbdeploy tool and set the *websphere.home* property (look at the examples below).

Attribute	Description	Required
destdir	The base directory into which the generated weblogic ready jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace.	Yes
ejbdeploy	Decides whether ejbdeploy is called. When you set this to true, be sure, to run ant with the ibm jdk.	No, defaults to true
suffix	String value appended to the basename of the deployment descriptor to create the filename of the WebLogic EJB jar file.	No, defaults to '.jar'.
keepgeneric	This controls whether the generic file used as input to ejbdeploy is retained.	No, defaults to false
rebuild	This controls whether ejbdeploy is called although no changes have occurred.	No, defaults to false
tempdir	A directory, where ejbdeploy will write temporary files	No, defaults to '_ejbdeploy_temp'.
dbName dbSchema	These options are passed to ejbdeploy.	No
dbVendor	This option is passed to ejbdeploy. Valid options can be obtained by running the following command: `<WAS_HOME>/bin/EJBDeploy.[sh/bat] -help` This is also used to determine the name of the Map.mapxmi and Schema.dbxmi files, for example Account-DB2UDBWIN_V71-Map.mapxmi and Account-DB2UDBWIN_V71-Schema.dbxmi.	No
codegen quiet novalidate noinform trace use35MappingRules	These options are all passed to ejbdeploy. All options except 'quiet' default to false.	No
rmicOptions	This option is passed to ejbdeploy and will be passed on to rmic.	No

This example shows ejbjar being used to generate deployment jars for all deployment descriptors in the descriptor dir:

```
    <property name="webpshere.home" value="${was4.home}"/>
    <ejbjar srcdir="${build.class}" descriptordir="etc/ejb">
      <include name="*-ejb-jar.xml"/>
      <websphere dbvendor="DB2UDBOS390_V6"
                 ejbdeploy="true"
```

```
                oldCMP="false"
                tempdir="/tmp"
                destdir="${dist.server}">
        <wasclasspath>
          <pathelement
location="${was4.home}/deploytool/itp/plugins/org.eclipse.core.boot/boot.jar"/>
          <pathelement
location="${was4.home}/deploytool/itp/plugins/com.ibm.etools.ejbdeploy/runtime/batch.jar"/>
          <pathelement location="${was4.home}/lib/xerces.jar"/>
          <pathelement location="${was4.home}/lib/ivjejb35.jar"/>
          <pathelement location="${was4.home}/lib/j2ee.jar"/>
          <pathelement location="${was4.home}/lib/vaprt.jar"/>
        </wasclasspath>
        <classpath>
          <path refid="build.classpath"/>
        </classpath>
      </websphere>
      <dtd publicId="-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
           location="${lib}/dtd/ejb-jar_1_1.dtd"/>
    </ejbjar>
```

iPlanet Application Server (iAS) element

The <iplanet< nested element is used to build iAS-specific stubs and skeletons and construct a JAR file which may be deployed to the iPlanet Application Server 6.0. The build process will always determine if the EJB stubs/skeletons and the EJB-JAR file are up to date, and it will do the minimum amount of work required.

Like the WebLogic element, a naming convention for the EJB descriptors is most commonly used to specify the name for the completed JAR file. For example, if the EJB descriptor ejb/Account-ejb-jar.xml is found in the descriptor directory, the iplanet element will search for an iAS-specific EJB descriptor file named ejb/Account-ias-ejb-jar.xml (if it isn't found, the task will fail) and a JAR file named ejb/Account.jar will be written in the destination directory. Note that when the EJB descriptors are added to the JAR file, they are automatically renamed META-INF/ejb-jar.xml and META-INF/ias-ejb-jar.xml.

Of course, this naming behaviour can be modified by specifying attributes in the ejbjar task (for example, basejarname, basenameterminator, and flatdestdir) as well as the iplanet element (for example, suffix). Refer to the appropriate documentation for more details.

Parameters:

Attribute	Description	Required
destdir	The base directory into which the generated JAR files will be written. Each JAR file is written in directories which correspond to their location within the "descriptordir" namespace.	Yes
classpath	The classpath used when generating EJB stubs and skeletons. If omitted, the classpath specified in the "ejbjar" parent task will be used. If specified, the classpath elements will be prepended to the classpath specified in the parent "ejbjar" task. Note that nested "classpath" elements may also be used.	No
keepgenerated	Indicates whether or not the Java source files which are generated by ejbc will be saved or automatically deleted. If "yes", the source files will be retained. If omitted, it defaults to "no".	No
debug	Indicates whether or not the ejbc utility should log additional debugging statements to the standard output. If "yes", the additional debugging statements will be generated. If omitted, it defaults to "no".	No
iashome	May be used to specify the "home" directory for this iAS installation. This is used to find the ejbc utility if it isn't included in the user's system path. If specified, it should refer to the [install-location]/iplanet/ias6/ias directory. If omitted, the ejbc utility must be on the user's	No

	system path.	
suffix	String value appended to the JAR filename when creating each JAR. If omitted, it defaults to ".jar".	No

As noted above, the iplanet element supports additional `<classpath>` nested elements.

Examples

This example demonstrates the typical use of the `<iplanet>` nested element. It will name each EJB-JAR using the "basename" prepended to each standard EJB descriptor. For example, if the descriptor named "Account-ejb-jar.xml" is processed, the EJB-JAR will be named "Account.jar"

```
    <ejbjar srcdir="${build.classesdir}"
            descriptordir="${src}">

            <iplanet destdir="${assemble.ejbjar}"
                     classpath="${ias.ejbc.cpath}"/>
            <include name="**/*-ejb-jar.xml"/>
            <exclude name="**/*ias-*.xml"/>
    </ejbjar>
```

This example demonstrates the use of a nested classpath element as well as some of the other optional attributes.

```
    <ejbjar srcdir="${build.classesdir}"
            descriptordir="${src}">

            <iplanet destdir="${assemble.ejbjar}"
                     iashome="${ias.home}"
                     debug="yes"
                     keepgenerated="yes">
                     <classpath>
                         <pathelement path="."/>
                         <pathelement path="${build.classpath}"/>
                     </classpath>
            </iplanet>
            <include name="**/*-ejb-jar.xml"/>
            <exclude name="**/*ias-*.xml"/>
    </ejbjar>
```

This example demonstrates the use of basejarname attribute. In this case, the completed EJB-JAR will be named "HelloWorld.jar" If multiple EJB descriptors might be found, care must be taken to ensure that the completed JAR files don't overwrite each other.

```
    <ejbjar srcdir="${build.classesdir}"
            descriptordir="${src}"
            basejarname="HelloWorld">

            <iplanet destdir="${assemble.ejbjar}"
                     classpath="${ias.ejbc.cpath}"/>
            <include name="**/*-ejb-jar.xml"/>
            <exclude name="**/*ias-*.xml"/>
    </ejbjar>
```

This example demonstrates the use of the dtd nested element. If the local copies of the DTDs are included in the classpath, they will be automatically referenced without the nested elements. In iAS 6.0 SP2, these local DTDs are found in the [iAS-install-directory]/APPS directory. In iAS 6.0 SP3, these local DTDs are found in the [iAS-install-directory]/dtd directory.

```
    <ejbjar srcdir="${build.classesdir}"
            descriptordir="${src}">
            <iplanet destdir="${assemble.ejbjar}">
                     classpath="${ias.ejbc.cpath}"/>
            <include name="**/*-ejb-jar.xml"/>
            <exclude name="**/*ias-*.xml"/>

            <dtd publicId="-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
                 location="${ias.home}/APPS/ejb-jar_1_1.dtd"/>
            <dtd publicId="-//Sun Microsystems, Inc.//DTD iAS Enterprise JavaBeans 1.0//EN"
                 location="${ias.home}/APPS/IASEjb_jar_1_0.dtd"/>
```

```
    </ejbjar>
```

JOnAS (Java Open Application Server) element

The <jonas> nested element is used to build JOnAS-specific stubs and skeletons thanks to the GenIC specific tool, and construct a JAR file which may be deployed to the JOnAS Application Server. The build process will always determine if the EJB stubs/skeletons and the EJB-JAR file are up to date, and it will do the minimum amount of work required.

Like the WebLogic element, a naming convention for the EJB descriptors is most commonly used to specify the name for the completed JAR file. For example, if the EJB descriptor ejb/Account-ejb-jar.xml is found in the descriptor directory, the <jonas> element will search for a JOnAS-specific EJB descriptor file named ejb/Account-jonas-ejb-jar.xml and a JAR file named ejb/Account.jar will be written in the destination directory. But the <jonas> element can also use the JOnAS naming convention. With the same example as below, the EJB descriptor can also be named ejb/Account.xml (no base name terminator here) in the descriptor directory. Then the <jonas> element will search for a JOnAS-specific EJB descriptor file called ejb/jonas-Account.xml. This convention do not follow strictly the ejb-jar naming convention recommendation but is supported for backward compatibility with previous version of JOnAS.

Note that when the EJB descriptors are added to the JAR file, they are automatically renamed META-INF/ejb-jar.xml and META-INF/jonas-ejb-jar.xml.

Of course, this naming behavior can be modified by specifying attributes in the ejbjar task (for example, basejarname, basenameterminator, and flatdestdir) as well as the iplanet element (for example, suffix). Refer to the appropriate documentation for more details.

Parameters:

Attribute	Description	Required
destdir	The base directory into which the generated JAR files will be written. Each JAR file is written in directories which correspond to their location within the "descriptordir" namespace.	Yes
jonasroot	The root directory for JOnAS.	Yes
classpath	The classpath used when generating EJB stubs and skeletons. If omitted, the classpath specified in the "ejbjar" parent task will be used. If specified, the classpath elements will be prepended to the classpath specified in the parent "ejbjar" task (see also the ORB attribute documentation below). Note that nested "classpath" elements may also be used.	No
keepgenerated	true if the intermediate Java source files generated by GenIC must be deleted or not. If omitted, it defaults to false.	No
nocompil	true if the generated source files must not be compiled via the java and rmi compilers. If omitted, it defaults to false.	No
novalidation	true if the XML deployment descriptors must be parsed without validation. If omitted, it defaults to false.	No
javac	Java compiler to use. If omitted, it defaults to the value of build.compiler property.	No
javacopts	Options to pass to the java compiler.	No
rmicopts	Options to pass to the rmi compiler.	No
secpropag	true if the RMI Skel. and Stub. must be modified to implement the implicit propagation of the security context (the transactional context is always provided). If omitted, it defaults to false.	No
verbose	Indicates whether or not to use -verbose switch. If omitted, it defaults to false.	No

additionalargs	Add additional args to GenIC.	No
keepgeneric	`true` if the generic JAR file used as input to GenIC must be retained. If omitted, it defaults to `false`.	No
jarsuffix	String value appended to the JAR filename when creating each JAR. If omitted, it defaults to ".jar".	No
orb	Choose your ORB : RMI, JEREMIE, DAVID. If omitted, it defaults to the one present in classpath. If specified, the corresponding JOnAS JAR is automatically added to the classpath.	No
nogenic	If this attribute is set to `true`, JOnAS's GenIC will not be run on the EJB JAR. Use this if you prefer to run GenIC at deployment time. If omitted, it defaults to `false`.	No

As noted above, the jonas element supports additional `<classpath>` nested elements.

Examples

This example shows ejbjar being used to generate deployment jars using a JOnAS EJB container. This example requires the naming standard to be used for the deployment descriptors. Using this format will create a EJB JAR file for each variation of '*-jar.xml' that is found in the deployment descriptor directory.

```
<ejbjar srcdir="${build.classes}"
        descriptordir="${descriptor.dir}">
  <jonas destdir="${deployment.jars.dir}"
       jonasroot="${jonas.root}"
       orb="RMI"/>
  <include name="**/*.xml"/>
  <exclude name="**/jonas-*.xml"/>
  <support dir="${build.classes}">
       <include name="**/*.class"/>
  </support>
</ejbjar>
```

This example shows ejbjar being used to generate a single deployment jar using a JOnAS EJB container. This example does require the deployment descriptors to use the naming standard. This will create only one ejb jar file - 'TheEJBJar.jar'.

```
<ejbjar srcdir="${build.classes}"
        descriptordir="${descriptor.dir}"
        basejarname="TheEJBJar">
  <jonas destdir="${deployment.jars.dir}"
           jonasroot="${jonas.root}"
           suffix=".jar"
           classpath="${descriptor.build.classpath}"/>
  <include name="**/ejb-jar.xml"/>
  <exclude name="**/jonas-ejb-jar.xml"/>
</ejbjar>
```

Javah

Description

Generates JNI headers from a Java class.

When this task executes, it will generate the C header and source files that are needed to implement native methods. JNI operates differently depending on whether JDK1.2 (or later) or pre-JDK1.2 systems are used.

It is possible to use different compilers. This can be selected with the `implementation` attribute or a nested element. Here are the choices of the attribute:

- default - the default compiler (kaffeh or sun) for the platform.
- sun (the standard compiler of the JDK)
- kaffeh (the native standard compiler of Kaffe)

Note: if you are using this task to work on multiple files the command line may become too long on some operating systems. Unfortunately the javah command doesn't support command argument files the way javac (for example) does, so all that can be done is breaking the amount of classes to compile into smaller chunks.

Parameters

Attribute	Description	Required
class	the fully-qualified name of the class (or classes, separated by commas)	Yes
outputFile	concatenates the resulting header or source files for all the classes listed into this file	Yes
destdir	sets the directory where javah saves the header files or the stub files.	
force	specifies that output files should always be written (JDK1.2 only)	No
old	specifies that old JDK1.0-style header files should be generated (otherwise output file contain JNI-style native method function prototypes) (JDK1.2 only)	No
stubs	generate C declarations from the Java object file (used with old)	No
verbose	causes Javah to print a message concerning the status of the generated files	No
classpath	the classpath to use.	No
bootclasspath	location of bootstrap class files.	No
extdirs	location of installed extensions.	No
implementation	The compiler implementation to use. If this attribute is not set, the default compiler for the current VM will be used. (See the above list of valid compilers.)	No

Either outputFile or destdir must be supplied, but not both.

Parameters specified as nested elements

arg

You can specify additional command line arguments for the compiler with nested `<arg>` elements. These elements are specified like Command-line Arguments but have an additional attribute that can be used to enable arguments only if a given compiler implementation will be used.

Attribute	Description	Required
value	See Command-line Arguments.	Exactly one of these.
line		
file		
path		
prefix	See Command-line Arguments. *Since Ant 1.8.*	No
suffix		No
implementation	Only pass the specified argument if the chosen compiler implementation matches the value of this attribute. Legal values are the same as those in the above list of valid compilers.)	No

implementationclasspath *since Ant 1.8.0*

A PATH like structure holding the classpath to use when loading the compiler implementation if a custom class has been specified. Doesn't have any effect when using one of the built-in compilers.

Any nested element of a type that implements JavahAdapter *since Ant 1.8.0*

If a defined type implements the `JavahAdapter` interface a nested element of that type can be used as an alternative to the `implementation` attribute.

Examples

```
<javah destdir="c" class="org.foo.bar.Wibble"/>
```

makes a JNI header of the named class, using the JDK1.2 JNI model. Assuming the directory 'c' already exists, the file `org_foo_bar_Wibble.h` is created there. If this file already exists, it is left unchanged.

```
<javah outputFile="wibble.h">
  <class name="org.foo.bar.Wibble,org.foo.bar.Bobble"/>
</javah>
```

is similar to the previous example, except the output is written to a file called `wibble.h` in the current directory.

```
<javah destdir="c" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
```

writes three header files, one for each of the classes named. Because the force option is set, these header files are always written when the Javah task is invoked, even if they already exist.

```
<javah destdir="c" verbose="yes" old="yes" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
<javah destdir="c" verbose="yes" stubs="yes" old="yes" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
```

writes the headers for the three classes using the 'old' JNI format, then writes the corresponding .c stubs. The verbose option will cause Javah to describe its progress.

If you want to use a custom JavahAdapter `org.example.MyAdapter` you can either use the implementation attribute:

```
<javah destdir="c" class="org.foo.bar.Wibble"
       implementation="org.example.MyAdapter"/>
```

or a define a type and nest this into the task like in:

```
<componentdef classname="org.example.MyAdapter"
              name="myadapter"/>
<javah destdir="c" class="org.foo.bar.Wibble">
  <myadapter/>
</javah>
```

in which case your javah adapter can support attributes and nested elements of its own.

Translate

Description

Identifies keys in files delimited by special tokens and translates them with values read from resource bundles.

A resource bundle contains locale-specific key-value pairs. A resource bundle is a hierarchical set of property files. A bundle name makes up its base family name. Each file that makes up this bundle has this name plus its locale. For example, if the resource bundle name is MyResources, the file that contains German text will take the name MyResources_de. In addition to language, country and variant are also used to form the files in the bundle.

The resource bundle lookup searches for resource files with various suffixes on the basis of (1) the desired locale and (2) the default locale (basebundlename), in the following order from lower-level (more specific) to parent-level (less specific):

```
basebundlename + "_" + language1 + "_" + country1 + "_" + variant1
basebundlename + "_" + language1 + "_" + country1
basebundlename + "_" + language1
basebundlename
basebundlename + "_" + language2 + "_" + country2 + "_" + variant2
basebundlename + "_" + language2 + "_" + country2
basebundlename + "_" + language2
```

The file names generated thus are appended with the string ".properties" to make up the file names that are to be used.

File encoding is supported. The encoding scheme of the source files, destination files and the bundle files can be specified. Destination files can be explicitly overwritten using the *forceoverwrite* attribute. If *forceoverwrite* is false, the destination file is overwritten only if either the source file or any of the files that make up the bundle have been modified after the destination file was last modified.

New in Ant 1.6:
Line endings of source files are preserved in the translated files.

FileSets are used to select files to translate.

Parameters

Attribute	Description	Required
todir	Destination directory where destination files are to be created.	Yes
starttoken	The starting token to identify keys.	Yes
endtoken	The ending token to identify keys.	Yes
bundle	Family name of resource bundle.	Yes
bundlelanguage	Locale specific language of resource bundle. Defaults to default locale's language.	No
bundlecountry	Locale specific country of resource bundle. Defaults to default locale's country.	No
bundlevariant	Locale specific variant of resource bundle. Defaults to the default variant of the country and language being used.	No
srcencoding	Source file encoding scheme. Defaults to system default file encoding.	No
destencoding	Destination file encoding scheme. Defaults to source file encoding.	No
bundleencoding	Resource Bundle file encoding scheme. Defaults to source file encoding.	No
forceoverwrite	Overwrite existing files even if the destination files are newer. Defaults to "no".	No

Parameters specified as nested elements

fileset

FileSets are used to select files that contain keys for which value translated files are to be generated.

Examples

Translate source file encoded in english into its japanese equivalent using a resource bundle encoded in japanese.

```
  <translate toDir="$(dest.dir}/ja"
        starttoken="#"
        endtoken="#"
        bundle="resource/BaseResource"
        bundlelanguage="ja"
        forceoverwrite="yes"
        srcencoding="ISO8859_1"
        destencoding="SJIS"
        bundleencoding="SJIS">
        <fileset dir="${src.dir}">
            <include name="**/*.jsp"/>
        </fileset>
  </translate>
```

Perforce Tasks User Manual

by

- Les Hughes (leslie.hughes@rubus.com)
- Kirk Wylie (kirk@radik.com)
- Matt Bishop (matt@thebishops.org)
- Antoine Levy-Lambert

Contents

Introduction

These tasks provide an interface to the Perforce SCM. The `org.apache.tools.ant.taskdefs.optional.perforce` package consists of a simple framework to support p4 functionality as well as some Ant tasks encapsulating frequently used (by me :-) p4 commands. However, the addition of new p4 commands is a pretty simple task (see the source). Although it is possible to use these commands on the desktop, they were primarily intended to be used by automated build systems.

Note: These tasks require the oro regular expression package. See library dependencies for the precise version required. You will also need the Perforce client executable (p4 or p4.exe but not p4win.exe) in your path.

The Tasks

Task	Description
P4Sync	Synchronise a workspace to a depot
P4Change	Request a new changelist from the Perforce server
P4Edit	Open files for edit (checkout)
P4Submit	Submit a changelist to the Perforce server (checkin)
P4Have	List current files in client view, useful for reporting
P4Label	Create a label reflecting files in the current workspace
P4Labelsync	Syncs an existing label
P4Counter	Obtain or set the value of a counter
P4Reopen	Move files between changelists
P4Revert	Revert files

P4Add	Add files
P4Delete	Delete files
P4Integrate	Integrate files
P4Resolve	Resolve files
P4Fstat	Show differences between local repository and p4 repository

General P4 Properties

Each p4 task requires a number of settings, either through build-wide properties, individual attributes or environment variables. These are

Property	Attribute	Env Var	Description	Default
p4.port	port	P4PORT	The p4d server and port to connect to	perforce:1666
p4.client	client	P4CLIENT	The p4 client spec to use	The logged in username
p4.user	user	P4USER	The p4 username	The logged in username
--	view	--	The client, branch or label view to operate upon. See the p4 user guide for more info.	//...

Your local installation of Perforce may require other settings (e.g. P4PASSWD, P4CONFIG). Many of these settings can be set using the globalopts attribute (described below), but not all. If a setting cannot be set by the command-line options, then it can only be set outside of Ant as an environment variable.

Additionally, you may also specify the following attributes:

Attribute	Description	Required
failonerror	Specifies whether to stop the build (`true`|`yes`|`on`) or keep going (`false`|`no`|`off`) if an error is returned from the p4 command.	No; defaults to true.
globalopts	Specifies global options for perforce to use while executing the task. These properties should be concatenated into one string, such as "-P *password* -C EUCJIS". Use the command-line option syntax, NOT the environment variable names. See the Perforce Command Reference for details.	No

Examples

Setting in the environment:-

(Unix csh)

```
setenv P4PORT myperforcebox:1666
```

(Unix sh et al)

```
P4USER=myp4userid; export P4USER
```

Using build properties:-

```
<property name="p4.client" value="nightlybuild"/>
```

Using task attributes:-

```
<p4Whatever
        port="myserver:1666"
        client="smoketest"
        user="smoketestdude"
        .
        .
        .
/>
```

For more information regarding the underlying 'p4' commands you are referred to the Perforce Command Reference available from the Perforce website.

Task Descriptions

P4Sync

Description:

Synchronize the current workspace with the depot.

Parameters

Attribute	Description	Required
force	force a refresh of files, if this attribute has been set.	no - if omitted, it will be off, otherwise a refresh will be forced.
label	sync client to label	no

Examples

```
<p4sync label="nightlybuild-0.0123" force="foo"/>
<p4sync view="//depot/projects/projectfoo/main/src/..."/>
```

P4Change

Description:

Request a new changelist from the Perforce server. This task sets the ${p4.change} property which can then be passed to P4Submit, P4Edit, or P4Add, or P4Delete, then to P4Submit.

Parameters

Attribute	Description	Required
description	Description for ChangeList. If none specified, it will default to "AutoSubmit By Ant"	No.

Examples

```
<p4change description="Change Build Number in Script">
```

P4Edit

Description:

Open file(s) for edit. P4Change should be used to obtain a new changelist for P4Edit as, although P4Edit can open files to the default change, P4Submit cannot yet submit it.

Parameters

Attribute	Description	Required
view	The filespec to request to edit	Yes
change	An existing changelist number to assign files to.	No, but see above.

Examples

```
<p4edit
    view="//depot/projects/projectfoo/main/src/Blah.java..."
    change="${p4.change}"/>
```

P4Submit

Description:

Submit a changelist, usually obtained from P4Change.

P4Submit will also change the value of the property p4.change if the change list is renamed by the Perforce server.

P4Submit will set a property p4.needsresolve to 1 if the change could not be submitted due to files needing resolving.

Files will need resolve if at the time of checking in, the revision that was checked out to do the current edit is not the latest any more.

If no files need resolve, the p4.needsresolve will be set to 0.

Parameters

Attribute	Description	Required
change	The changelist number to submit	Yes
changeproperty	Name of a property to which the new change number will be assigned if the	No

	Perforce server renumbers the change Since ant 1.6.1	
needsresolveproperty	Name of property which will be set to `true` if the submit requires a resolve Since ant 1.6.1	No

Examples

```
<p4submit change="${p4.change}"/>
```

P4Have

Description:

List handy file info reflecting the current client contents.

Parameters

Attribute	Description	Required
None	--	--

Examples

```
<p4have/>
```

P4Label

Description:

Create a new label and set contents to reflect current client file revisions.

Parameters

Attribute	Description	Required
name	The name of the label	Yes
view	client view to use for label The view can contain multiple lines separated by ; or :	No
desc	Label Description	No
lock	Lock the label once created.	No

Examples

```
<p4label
```

```
        name="NightlyBuild:${DSTAMP}:${TSTAMP}"
        desc="Auto Nightly Build"
        lock="locked"
        view="//firstdepot/...;//secondepot/foo/bar/..."
/>
```

P4Labelsync

Description:

Syncs an existing label against the current workspace or against specified revisions.

Parameters

Attribute	Description	Required	Perforce command line flag
name	The name of the label	Yes	-l labelname
view	list of files or revision specs separated by : or ; in the absence of this attribute, the labelsync will be done against the current Perforce client or the value of the p4client attribute or the value of the p4.client property or the value of the environment variable P4CLIENT	No	file[revRange] ...
simulationmode	Displays which effect the operation would have on the label but do not actually do it	No	-n
add	If set to true, preserve files which exist in the label, but not in the current view	No	-a
delete	If set to true, remove from the label the files mentioned in the view attribute	No	-d

Examples

```
<
p4labelsync
name="current_release"
view="//depot/...#head;//depot2/file1#25"
add="true"
/>
```

This example will add into the label called *current_release* the current head revision of all the files located under *//depot* and the revision 25 of the file *//depot2/file1*.

```
<
p4labelsync
name="current_release"
p4client="myclient"
/>
```

This example will update the label called *current_release* so that it reflects the Perforce client *myclient*.
Files present in the label before the sync and not present currently in the client will be removed from the label, because the add attribute is not set.

```
<
p4labelsync
name="current_release"
/>
```

This example will update the label called *current_release* so that it reflects the current default client for the ant Perforce tasks.
The default client is by order of priority :

- the value of the p4.client property if set in the project
- the value of the P4CLIENT environment variable
- the default Perforce client from the Windows registry under Windows operating systems

Files present in the label before the sync and not present currently in the client will be removed from the label, because the add attribute is not set.

P4Counter

Description:

Obtain or set the value of a counter. When used in its base form (where only the counter name is provided), the counter value will be printed to the output stream. When the value is provided, the counter will be set to the value provided. When a property name is provided, the property will be filled with the value of the counter. You may not specify to both get and set the value of the counter in the same Task.

The user performing this task must have Perforce "review" permissions as defined by Perforce protections in order for this task to succeed.

Parameters

Attribute	Description	Required
name	The name of the counter	Yes
value	The new value for the counter	No
property	The property to be set with the value of the counter	No

Examples

Print the value of the counter named "last-clean-build" to the output stream:

```
<p4counter name="last-clean-build"/>
```

Set the value of the counter based on the value of the "TSTAMP" property:

```
<p4counter name="last-clean-build" value="${TSTAMP}"/>
```

Set the value of the "p4.last.clean.build" property to the current value of the "last-clean-build" counter:

```
<p4counter name="last-clean-build" property="p4.last.clean.build"/>
```

P4Reopen

Description:

Move (or reopen in Perforce speak) checkout files between changelists.

Parameters

Attribute	Description	Required
tochange	The changelist to move files to.	Yes

Examples

Move all open files to the default changelist

```
<p4reopen view="//..." tochange="default"/>
```

Create a new changelist then reopen into it, any files from the view //projects/foo/main/...

```
<p4change description="Move files out of the way"/>
<p4reopen view="//projects/foo/main/..." tochange="${p4.change}"/>
```

P4Revert

Description:

Reverts files.

Parameters

Attribute	Description	Required
change	The changelist to revert.	No
revertOnlyUnchanged	Revert only unchanged files (p4 revert -a)	No

Examples

Revert everything!

```
<p4revert view="//..."/>
```

Revert any unchanged files in the default change

```
<p4revert change="default" revertonlyunchanged="true"/>
```

P4Add

Description:

Adds files specified in nested fileset children.

Parameters

Attribute	Description	Required
commandlength	A positive integer specifying the maximum length of the commandline when calling Perforce to add the files. Defaults to 450, higher values mean faster execution, but also possible failures.	No
changelist	If specified the open files are associated with the specified pending changelist number; otherwise the open files are associated with the default changelist.	No

Examples

Require a changelist, add all java files starting from a directory, and submit

```
<p4change/>
<p4add commandlength="20000" changelist="${p4.change}">
    <fileset dir="../dir/src/" includes="**/*.java"/>
<p4add>
<p4submit change="${p4.change}"/>
```

P4Fstat

Description:

Lists files under Perforce control and files not under Perforce control in one or several filesets

Parameters

Attribute	Description	Required
showfilter	should be one of value: description "all": list all files, first the ones which are under Perforce control, then the others "existing": list only files under Perforce control "non-existing": list only files which are not under Perforce control	Yes
fileset	one or several fileset(s)	yes (at least one fileset is needed)

Several nested filesets can be used with P4Fstat, one should be there at least.

Examples

will list all the files under C:\dev\gnu\depot, sorted by under Perforce or not under Perforce

```
<project name="p4fstat" default="p4fstat" basedir="C:\dev\gnu">
    <target name="p4fstat" >
        <p4fstat showfilter="all">
            <fileset dir="depot" includes="**/*"/>
```

```
        </p4fstat>
    </target>
</project>
```

P4Delete

Description:

Open file(s) for delete. P4Change should be used to obtain a new changelist for P4Delete as, although P4Delete can open files to the default change, P4Submit cannot yet submit it.

Parameters

Attribute	Description	Required
view	The filespec to request to delete	Yes
change	An existing changelist number to assign files to.	No, but see above.

Examples

```
<p4delete
    view="//depot/projects/projectfoo/main/src/Blah.java..."
    change="${p4.change}"/>
```

P4Integrate

Description:

Open file(s) for integrate. P4Change should be used to obtain a new changelist for P4Integrate as, although P4Integrate can open files to the default change, P4Submit cannot yet submit it.

Parameters

If this task is used without using a branch definition, both fromfile and tofile must be supplied. If a branch definition is supplied, at least one of fromfile or tofile should be supplied. Both fromfile and tofile can be supplied together with a branch definition.

Attribute	Description	Required	Perforce command line flag
fromfile	Original file or view	required if a branch is not specified	
tofile	Target file or view.	required if a branch is not specified	

branch	Name of branch specification	No	-b
change	An existing changelist number to assign files to.	No, but see above.	-c
forceintegrate	Forces integration regardless of previous integration history (*)	No	-f
restoredeletedrevisions	Enables integration around deleted revisions (*)	No	-d
leavetargetrevision	Prevents target files from being synced to head revision before integration (*)	No	-h
enablebaselessmerges	Forces integration to existing target files which have no integration history relative to the source files (*)	No	-i
simulationmode	Displays which integrations are necessary but do not actually schedule them (*)	No	-n
reversebranchmappings	Reverses mappings in the branch view, with source and target files exchanging place (*)	No	-r
propagatesourcefiletype	Makes source file type propagate to existing target files (*)	No	-t
nocopynewtargetfiles	Prevents the physical delivery on disk of new target files (*)	No	-v

(*) The following applies for a number of flags. The default is false. To set the flag, use "true"

Examples

```
<p4integrate
    fromfile="//depot/projects/projectfoo/main/src/Blah.java..."
    tofile="//depot/projects/projectfoo/release/src/Blah.java..."
    change="${p4.change}"/>
```

P4Resolve

Description:

Resolves files. You want to do this if :

- there have been or there may be concurrent edits of the same file. For instance, you have begun to edit a file, and while you were working on it, somebody has submitted a new version of the same file. When you first attempt to submit your file(s), you will get a message (property p4.needsresolve set).
- you have just been doing an integration to existing target files

P4Resolve does not use a change list number (it takes it from the files it is working on).

Parameters

Attribute	Description	Required	Perforce command line flag
view	The filespec to request to delete	Yes	
resolvemode	Should have one of these values :	Yes	corresponds to one of -am -

	• "automatic" • "force" • "safe" • "theirs" • "yours"		af -as -at -ay
redoall	allows previously resolved files to be resolved again (*)	No	-f
simulationmode	Lists the integrations which would be performed, without actually doing them. (*)	No	-n
forcetextmode	Attempts a textual merge, even for binary files (*)	No	-t
markersforall	Puts in markers for all changes, conflicting or not (*)	No	-v

(*) The following applies for a number of flags. The default is false. To set the flag, use "true"

Examples

```
<p4resolve
    view="//depot/projects/projectfoo/main/src/Blah.java..."
    resolvemode="automatic"/>
```

Change History

Sept 2000	--	Internal Release within Rubus
Nov 2000	V1.0	Initial Release donated to ASF :-)
Jan 2001	V1.1	Fixed cross platform (NT/Unix) bug Refactored p4 output handling code Refactored exec'ing code
Jan 2003	V1.2	Added globalopts to P4Base to allow additional global options to be set. Added p4fstat task
May 2003	V1.3	Added p4labelsync, p4resolve, p4integrate. Changed p4submit (detection of changes of change numbers, and of failed submits due to resolution needed)
Jan 2004	ant 1.6.1	Changed p4submit, needsresolveproperty and changeproperty added

GUnzip/BUnzip2

Description

Expands a resource packed using GZip or BZip2.

If *dest* is a directory the name of the destination file is the same as *src* (with the ".gz" or ".bz2" extension removed if present). If *dest* is omitted, the parent dir of *src* is taken. The file is only expanded if the source resource is newer than the destination file, or when the destination file does not exist.

Parameters

Attribute	Description	Required
src	the file to expand.	Yes, or a nested resource collection.
dest	the destination file or directory.	No

Parameters specified as nested elements

any resource or single element resource collection

The specified resource will be used as src.

Examples

```
<gunzip src="test.tar.gz"/>
```

expands *test.tar.gz* to *test.tar*

```
<bunzip2 src="test.tar.bz2"/>
```

expands *test.tar.bz2* to *test.tar*

```
<gunzip src="test.tar.gz" dest="test2.tar"/>
```

expands *test.tar.gz* to *test2.tar*

```
<gunzip src="test.tar.gz" dest="subdir"/>
```

expands *test.tar.gz* to *subdir/test.tar* (assuming subdir is a directory).

```
<gunzip dest=".">
  <url url="http://example.org/archive.tar.gz"/>
</gunzip>
```

downloads *http://example.org/archive.tar.gz* and expands it to *archive.tar* in the project's basedir on the fly.

Related tasks

```
<gunzip src="some-archive.gz" dest="some-dest-dir"/>
```

is identical to

```
<copy todir="some-dest-dir">
  <gzipresource>
    <file file="some-archive.gz"/>
  </gzipresource>
  <mapper type="glob" from="*.gz" to="*"/>
</copy>
```

The same is also true for `<bunzip2>` and `<bzip2resource>`. `<copy>` offers additional features like filtering files on the fly, allowing a file to be mapped to multiple destinations, preserving the last modified time or a configurable file system timestamp granularity.

GZip/BZip2

Description

Packs a resource using the GZip or BZip2 algorithm. The output file is only generated if it doesn't exist or the source resource is newer.

Parameters

Attribute	Description	Required
src	the file to gzip/bzip.	Yes, or a nested resource collection.
destfile	the destination file to create.	Exactly one of the two.
zipfile	the *deprecated* old name of destfile.	

any resource or single element resource collection

The specified resource will be used as src. *Since Ant 1.7*

Examples

```
<gzip src="test.tar" destfile="test.tar.gz"/>
```

```
<bzip2 src="test.tar" destfile="test.tar.bz2"/>
```

```
<gzip destfile="archive.tar.gz">
  <url url="http://example.org/archive.tar"/>
</gzip>
```

downloads *http://example.org/archive.tar* and compresses it to *archive.tar.gz* in the project's basedir on the fly.

Cab

Description

The cab task creates Microsoft cab archive files. It is invoked similar to the jar or zip tasks. This task will work on Windows using the external cabarc tool (provided by Microsoft) which must be located in your executable path.

To use this task on other platforms you need to download and compile libcabinet from http://trill.cis.fordham.edu/~barbacha/cabinet_library/.

See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Parameters

Attribute	Description	Required
cabfile	the name of the cab file to create.	Yes
basedir	the directory to start archiving files from.	No
verbose	set to "yes" if you want to see the output from the cabarc tool. defaults to "no".	No
compress	set to "no" to store files without compressing. defaults to "yes".	No
options	use to set additional command-line options for the cabarc tool. should not normally be necessary.	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

Parameters specified as nested elements

fileset

The cab task supports one nested `<fileset>` element to specify the files to be included in the archive. If this is specified, the "basedir" attribute cannot be used.

Examples

```
<cab cabfile="${dist}/manual.cab"
     basedir="htdocs/manual"
  />
```

cabs all files in the htdocs/manual directory into a file called manual.cab in the ${dist} directory.

```
<cab cabfile="${dist}/manual.cab"
     basedir="htdocs/manual"
     excludes="mydocs/**, **/todo.html"
  />
```

cabs all files in the htdocs/manual directory into a file called manual.cab in the ${dist} directory. Files in the directory mydocs, or files with the name todo.html are excluded.

```
<cab cabfile="${dist}/manual.cab"
     basedir="htdocs/manual"
     includes="api/**/*.html"
     excludes="**/todo.html"
     verbose="yes"
  />
```

Cab all files in the htdocs/manual directory into a file called manual.cab in the ${dist} directory. Only html files under the directory api are archived, and files with the name todo.html are excluded. Output from the cabarc tool is displayed in the build output.

```
<cab cabfile="${dist}/manual.cab"
     verbose="yes">
  <fileset
       dir="htdocs/manual"
       includes="api/**/*.html"
       excludes="**/todo.html"
  />
</cab>
```

is equivalent to the example above.

Jar

Description

Jars a set of files.

The *basedir* attribute is the reference directory from where to jar.

Note that file permissions will not be stored in the resulting jarfile.

It is possible to refine the set of files that are being jarred. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

You can also use nested file sets for more flexibility, and specify multiple ones to merge together different trees of files into one JAR. The extended fileset and groupfileset child elements from the zip task are also available in the jar task. See the Zip task for more details and examples.

If the manifest is omitted, a simple one will be supplied by Ant.

The `update` parameter controls what happens if the JAR file already exists. When set to `yes`, the JAR file is updated with the files specified. When set to `no` (the default) the JAR file is overwritten. An example use of this is provided in the Zip task documentation. Please note that ZIP files store file modification times with a granularity of two seconds. If a file is less than two seconds newer than the entry in the archive, Ant will not consider it newer.

The `whenmanifestonly` parameter controls what happens when no files, apart from the manifest file, or nested services, match. If `skip`, the JAR is not created and a warning is issued. If `fail`, the JAR is not created and the build is halted with an error. If `create`, (default) an empty JAR file (only containing a manifest and services) is created.

(The Jar task is a shortcut for specifying the manifest file of a JAR file. The same thing can be accomplished by using the *fullpath* attribute of a zipfileset in a Zip task. The one difference is that if the *manifest* attribute is not specified, the Jar task will include an empty one for you.)

Manifests are processed by the Jar task according to the Jar file specification. Note in particular that this may result in manifest lines greater than 72 bytes being wrapped and continued on the next line.

The Jar task checks whether you specified package information according to the versioning specification.

Please note that the zip format allows multiple files of the same fully-qualified name to exist within a single archive. This has been documented as causing various problems for unsuspecting users. If you wish to avoid this behavior you must set the `duplicate` attribute to a value other than its default, `"add"`.

Parameters

Attribute	Description	Required

destfile	the JAR file to create.	Yes
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true. Unless you set the *keepcompression* attribute to false, this will apply to the entire archive, not only the files you've added while updating.	No
keepcompression	For entries coming from existing archives (like nested *zipfileset*s or while updating the archive), keep the compression as it has been originally instead of using the *compress* attribute. Defaults false. *Since Ant 1.6*	No
encoding	The character encoding to use for filenames inside the archive. Defaults to UTF8. **It is not recommended to change this value as the created archive will most likely be unreadable for Java otherwise.** See also the discussion in the zip task page	No
filesonly	Store only file entries, defaults to false	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use. This can be either the location of a manifest, or the name of a jar added through a fileset. If its the name of an added jar, the task expects the manifest to be in the jar at META-INF/MANIFEST.MF	No
filesetmanifest	behavior when a Manifest is found in a zipfileset or zipgroupfileset file is found. Valid values are "skip", "merge", and "mergewithoutmain". "merge" will merge all of the manifests together, and merge this into any other specified manifests. "mergewithoutmain" merges everything but the Main section of the manifests. Default value is "skip".	No
update	indicates whether to update or overwrite the destination file if it already exists. Default is "false".	No
whenmanifestonly	behavior when no files match. Valid values are "fail", "skip", and "create". Default is "create".	No
duplicate	behavior when a duplicate file is found. Valid values are "add", "preserve", and "fail". The default value is "add".	No
index	whether to create an index list to speed up classloading. This is a JDK 1.3+ specific feature. Unless you specify additional jars with nested `indexjars` elements, only the contents of this jar will be included in the index. Defaults to false.	No
indexMetaInf	whether to include META-INF and its children in the index. Doesn't have any effect if *index* is false. Sun's jar implementation used to skip the META-INF directory and Ant followed that example. The behavior has been changed with Java 5. In order to avoid problems with Ant generated jars on Java 1.4 or earlier Ant will not include META-INF unless explicitly asked to. *Ant 1.8.0* - Defaults to false.	No
manifestencoding	The encoding used to read the JAR manifest, when a manifest file is specified. The task will always use UTF-8 when writing the manifest.	No, defaults

		to the platform encoding.
roundup	Whether the file modification times will be rounded up to the next even number of seconds. Zip archives store file modification times with a granularity of two seconds, so the times will either be rounded up or down. If you round down, the archive will always seem out-of-date when you rerun the task, so the default is to round up. Rounding up may lead to a different type of problems like JSPs inside a web archive that seem to be slightly more recent than precompiled pages, rendering precompilation useless. Defaults to true. *Since Ant 1.6.2*	No
level	Non-default level at which file compression should be performed. Valid values range from 0 (no compression/fastest) to 9 (maximum compression/slowest). *Since Ant 1.7*	No
strict	Configures how to handle breaks of the packaging version specification: • **fail** = throws a BuildException • **warn** = logs a message on warn level • **ignore** = logs a message on verbose level (default) *Since Ant 1.7.1*	No, defaults to `ignore`.
preserve0permissions	when updating an archive or adding entries from a different archive Ant will assume that a Unix permissions value of 0 (nobody is allowed to do anything to the file/directory) means that the permissions haven't been stored at all rather than real permissions and will instead apply its own default values. Set this attribute to true if you really want to preserve the original permission field.*since Ant 1.8.0*	No, default is false
useLanguageEncodingFlag	Whether to set the language encoding flag if the encoding is UTF-8. This setting doesn't have any effect if the encoding is not UTF-8. *Since Ant 1.8.0.* See also the discussion in the zip task page	No, default is true
createUnicodeExtraFields	Whether to create unicode extra fields to store the file names a second time inside the entry's metadata. Possible values are "never", "always" and "not-encodable" which will only add Unicode extra fields if the file name cannot be encoded using the specified encoding. *Since Ant 1.8.0.* See also the discussion in the zip task page	No, default is "never"
fallbacktoUTF8	Whether to use UTF-8 and the language encoding flag instead of the specified encoding if a file name cannot be encoded using the specified encoding. *Since Ant 1.8.0.* See also the discussion in the zip task page	No, default is false
mergeClassPathAttributes	Whether to merge the Class-Path attributes found in different manifests (if merging manifests). If false, only the attribute of the last merged manifest will be preserved. *Since Ant 1.8.0.* unless you also set flattenAttributes to true this may result in manifests containing multiple Class-Path attributes which violates the manifest specification.	No, default is false
flattenAttributes	Whether to merge attributes occuring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute. *Since Ant 1.8.0.*	No, default is false

Nested elements

metainf

The nested `metainf` element specifies a FileSet. All files included in this fileset will end up in the `META-INF` directory of the jar file. If this fileset includes a file named `MANIFEST.MF`, the file is ignored and you will get a warning.

manifest

The manifest nested element allows the manifest for the Jar file to be provided inline in the build file rather than in an external file. This element is identical to the manifest task, but the file and mode attributes must be omitted.

If both an inline manifest and an external file are both specified, the manifests are merged.

When using inline manifests, the Jar task will check whether the manifest contents have changed (i.e. the manifest as specified is different in any way from the manifest that exists in the Jar, if it exists. If the manifest values have changed the jar will be updated or rebuilt, as appropriate.

indexjars

since ant 1.6.2

The nested `indexjars` element specifies a PATH like structure. Its content is completely ignored unless you set the index attribute of the task to true.

The index created by this task will contain indices for the archives contained in this path, the names used for the archives depend on your manifest:

- If the generated jar's manifest contains no Class-Path attribute, the file name without any leading directory path will be used and all parts of the path will get indexed.
- If the manifest contains a Class-Path attribute, this task will try to guess which part of the Class-Path belongs to a given archive. If it cannot guess a name, the archive will be skipped, otherwise the name listed inside the Class-Path attribute will be used.

This task will not create any index entries for archives that are empty or only contain files inside the META-INF directory unless the `indexmetainf` attribute has been set to `true`.

service

since ant 1.7.0

The nested `service` element specifies a service. Services are described by http://java.sun.com/j2se/1.5.0/docs/guide/jar/jar.html#Service%20Provider. The approach is to have providers JARs include files named by the service provided, for example, META-INF/services/javax.script.ScriptEngineFactory which can include implementation class names, one per line (usually just one per JAR). The name of the service is set by the "type" attribute. The classname implementing the service is the the "provider" attribute, or it one wants to specify a number of classes that implement the service, by "provider" nested elements.

Attribute	Description	Required
type	The name of the service.	Yes
provider	The classname of the class implemening the service.	Yes, unless there is a nested `<provider>` element.

The provider classname is specified either by the "provider" attribute, or by a nested <provider> element, which has a single "classname" attribute. If a JAR file has more that one implementation of the service, a number of nested <provider> elements may be used.

Examples

```
  <jar destfile="${dist}/lib/app.jar" basedir="${build}/classes"/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory.

```
  <jar destfile="${dist}/lib/app.jar"
       basedir="${build}/classes"
       excludes="**/Test.class"
  />
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory. Files with the name `Test.class` are excluded.

```
  <jar destfile="${dist}/lib/app.jar"
       basedir="${build}/classes"
       includes="mypackage/test/**"
       excludes="**/Test.class"
  />
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory. Only files under the directory `mypackage/test` are used, and files with the name `Test.class` are excluded.

```
  <jar destfile="${dist}/lib/app.jar">
    <fileset dir="${build}/classes"
             excludes="**/Test.class"
    />
    <fileset dir="${src}/resources"/>
  </jar>
```

jars all files in the `${build}/classes` directory and also in the `${src}/resources` directory together into a file called `app.jar` in the `${dist}/lib` directory. Files with the name `Test.class` are excluded. If there are files such as `${build}/classes/mypackage/MyClass.class` and `${src}/resources/mypackage/image.gif`, they will appear in the same directory in the JAR (and thus be considered in the same package by Java).

```
 <jar destfile="build/main/checksites.jar">
    <fileset dir="build/main/classes"/>
    <zipfileset src="lib/main/util.jar">
    <manifest>
      <attribute name="Main-Class"
            value="com.acme.checksites.Main"/>
    </manifest>
  </jar>
```

Creates an executable jar file with a main class "com.acme.checksites.Main", and embeds all the classes from util.jar.

```
  <jar destfile="test.jar" basedir=".">
    <include name="build"/>
    <manifest>
      <!-- Who is building this jar? -->
      <attribute name="Built-By" value="${user.name}"/>
      <!-- Information about the program itself -->
      <attribute name="Implementation-Vendor" value="ACME inc."/>
      <attribute name="Implementation-Title" value="GreatProduct"/>
      <attribute name="Implementation-Version" value="1.0.0beta2"/>
      <!-- details -->
      <section name="common/MyClass.class">
        <attribute name="Sealed" value="false"/>
      </section>
    </manifest>
  </jar>
```

This is an example of an inline manifest specification including the version of the build program (Implementation-

Version). Note that the Built-By attribute will take the value of the Ant property ${user.name}. The manifest produced by the above would look like this:

```
Manifest-Version: 1.0
Built-By: conor
Implementation-Vendor: ACME inc.
Implementation-Title: GreatProduct
Implementation-Version: 1.0.0beta2
Created-By: Apache Ant 1.7.0

Name: common/MyClass.class
Sealed: false
```

The following shows how to create a jar file specifing a service with an implementation of the JDK6 scripting interface:

```
<jar jarfile="pinky.jar">
  <fileset dir="build/classes"/>
  <service type="javax.script.ScriptEngineFactory"
           provider="org.acme.PinkyLanguage"/>
</jar>
```

The following shows how to create a jar file specifing a service with two implementations of the JDK6 scripting interface:

```
<jar jarfile="pinkyandbrain.jar">
  <fileset dir="classes"/>
  <service type="javax.script.ScriptEngineFactory">
    <provider classname="org.acme.PinkyLanguage"/>
    <provider classname="org.acme.BrainLanguage"/>
  </service>
</jar>
```

Zip

Description

Creates a zipfile.

The *basedir* attribute is the reference directory from where to zip.

Note that file permissions will not be stored in the resulting zipfile.

It is possible to refine the set of files that are being zipped. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Or, you may place within it nested file sets, or references to file sets. In this case `basedir` is optional; the implicit file set is *only used* if `basedir` is set. You may use any mixture of the implicit file set (with `basedir` set, and optional attributes like `includes` and optional subelements like `<include>`); explicit nested `<fileset>` elements so long as at least one fileset total is specified. The ZIP file will only reflect the relative paths of files *within* each fileset. The Zip task and its derivatives know a special form of a fileset named zipfileset that has additional attributes (described below).

The Zip task also supports the merging of multiple zip files into the zip file. This is possible through either the *src* attribute of any nested filesets or by using the special nested fileset *zipgroupfileset*.

The `update` parameter controls what happens if the ZIP file already exists. When set to `yes`, the ZIP file is updated with the files specified. (New files are added; old files are replaced with the new versions.) When set to `no` (the default) the ZIP file is overwritten if any of the files that would be added to the archive are newer than the entries inside the archive. Please note that ZIP files store file modification times with a granularity of two seconds. If a file is less than two seconds newer than the entry in the archive, Ant will not consider it newer.

The `whenempty` parameter controls what happens when no files match. If `skip` (the default), the ZIP is not created and a warning is issued. If `fail`, the ZIP is not created and the build is halted with an error. If `create`, an empty ZIP file (explicitly zero entries) is created, which should be recognized as such by compliant ZIP manipulation tools.

This task will now use the platform's default character encoding for filenames - this is consistent with the command line ZIP tools, but causes problems if you try to open them from within Java and your filenames contain non US-ASCII characters. Use the encoding attribute and set it to UTF8 to create zip files that can safely be read by Java. For a more complete discussion, see below

Starting with Ant 1.5.2, `<zip>` can store Unix permissions inside the archive (see description of the filemode and dirmode attributes for <zipfileset>). Unfortunately there is no portable way to store these permissions. Ant uses the algorithm used by Info-Zip's implementation of the zip and unzip commands - these are the default versions of zip and unzip for many Unix and Unix-like systems.

Please note that the zip format allows multiple files of the same fully-qualified name to exist within a single archive. This has been documented as causing various problems for unsuspecting users. If you wish to avoid this

behavior you must set the `duplicate` attribute to a value other than its default, `"add"`.

Please also note that different ZIP tools handle timestamps differently when it comes to applying timezone offset calculations of files. Some ZIP libraries will store the timestamps as they've been read from the filesystem while others will modify the timestamps both when reading and writing the files to make all timestamps use the same timezone. A ZIP archive created by one library may extract files with "wrong timestamps" when extracted by another library.

Ant's ZIP classes use the same algorithm as the InfoZIP tools and zlib (timestamps get adjusted), Windows' "compressed folders" function and WinZIP don't change the timestamps. This means that using the unzip task on files created by Windows' compressed folders function may create files with timestamps that are "wrong", the same is true if you use Windows' functions to extract an Ant generated ZIP archive.

Parameters

Attribute	Description	Required
destfile	the zip-file to create.	Exactly one of the two.
zipfile	the *deprecated* old name of destfile.	
basedir	the directory from which to zip the files.	No
compress	Not only store data but also compress them, defaults to true. Unless you set the *keepcompression* attribute to false, this will apply to the entire archive, not only the files you've added while updating.	No
keepcompression	For entries coming from existing archives (like nested *zipfileset*s or while updating the archive), keep the compression as it has been originally instead of using the *compress* attribute. Defaults false. *Since Ant 1.6*	No
encoding	The character encoding to use for filenames inside the zip file. For a list of possible values see http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html. Defaults to the platform's default character encoding. See also the discussion below	No
filesonly	Store only file entries, defaults to false	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
update	indicates whether to update or overwrite the destination file if it already exists. Default is "false".	No
whenempty	behavior when no files match. Valid values are "fail", "skip", and "create". Default is "skip".	No
duplicate	behavior when a duplicate file is found. Valid values are "add", "preserve", and "fail". The default value is "add".	No
roundup	Whether the file modification times will be rounded up to the next even number of seconds. Zip archives store file modification times with a granularity of two seconds, so	No

	the times will either be rounded up or down. If you round down, the archive will always seem out-of-date when you rerun the task, so the default is to round up. Rounding up may lead to a different type of problems like JSPs inside a web archive that seem to be slightly more recent than precompiled pages, rendering precompilation useless. Defaults to true. *Since Ant 1.6.2*	
comment	Comment to store in the archive. *Since Ant 1.6.3*	No
level	Non-default level at which file compression should be performed. Valid values range from 0 (no compression/fastest) to 9 (maximum compression/slowest). *Since Ant 1.7*	No
preserve0permissions	when updating an archive or adding entries from a different archive Ant will assume that a Unix permissions value of 0 (nobody is allowed to do anything to the file/directory) means that the permissions haven't been stored at all rather than real permissions and will instead apply its own default values. Set this attribute to true if you really want to preserve the original permission field.*since Ant 1.8.0*	No, default is false
useLanguageEncodingFlag	Whether to set the language encoding flag if the encoding is UTF-8. This setting doesn't have any effect if the encoding is not UTF-8. *Since Ant 1.8.0*. See also the discussion below	No, default is true
createUnicodeExtraFields	Whether to create unicode extra fields to store the file names a second time inside the entry's metadata. Possible values are "never", "always" and "not-encodable" which will only add Unicode extra fields if the file name cannot be encoded using the specified encoding. *Since Ant 1.8.0*. See also the discussion below	No, default is "never"
fallbacktoUTF8	Whether to use UTF-8 and the language encoding flag instead of the specified encoding if a file name cannot be encoded using the specified encoding. *Since Ant 1.8.0*. See also the discussion below	No, default is false

Encoding of File Names

Traditionally the ZIP archive format uses CodePage 437 as encoding for file name, which is not sufficient for many international character sets.

Over time different archivers have chosen different ways to work around the limitation - the `java.util.zip` packages simply uses UTF-8 as its encoding for example.

Ant has been offering the encoding attribute of the zip and unzip task as a way to explicitly specify the encoding to use (or expect) since Ant 1.4. It defaults to the platform's default encoding for zip and UTF-8 for jar and other jar-like tasks (war, ear, ...) as well as the unzip family of tasks.

More recent versions of the ZIP specification introduce something called the "language encoding flag" which can be used to signal that a file name has been encoded using UTF-8. Starting with Ant 1.8.0 all zip-/jar- and similar archives written by Ant will set this flag, if the encoding has been set to UTF-8. Our interoperabilty tests with existing archivers didn't show any ill effects (in fact, most archivers ignore the flag to date), but you can turn off the "language encoding flag" by setting the attribute `useLanguageEncodingFlag` to `false` on the zip-task if you should encounter problems.

The unzip (and similar tasks) -task will recognize the language encoding flag and ignore the encoding set on the task if it has been found.

The InfoZIP developers have introduced new ZIP extra fields that can be used to add an additional UTF-8 encoded file name to the entry's metadata. Most archivers ignore these extra fields. The zip family of tasks support an option `createUnicodeExtraFields` since Ant 1.8.0 which makes Ant write these extra fields either for all entries ("always") or only those whose name cannot be encoded using the spcified encoding (not-encodeable), it defaults to "never" since the extra fields create bigger archives.

The fallbackToUTF8 attribute of zip can be used to create archives that use the specified encoding in the majority of cases but UTF-8 and the language encoding flag for filenames that cannot be encoded using the specified encoding.

The unzip-task will recognize the unicode extra fields by default and read the file name information from them, unless you set the optional attribute `scanForUnicodeExtraFields` to false.

Recommendations for Interoperability

The optimal setting of flags depends on the archivers you expect as consumers/producers of the ZIP archives. Below are some test results which may be superseeded with later versions of each tool.

- The java.util.zip package used by the jar executable or to read jars from your CLASSPATH reads and writes UTF-8 names, it doesn't set or recognize any flags or unicode extra fields.
- 7Zip writes CodePage 437 by default but uses UTF-8 and the language encoding flag when writing entries that cannot be encoded as CodePage 437 (similar to the zip task with fallbacktoUTF8 set to true). It recognizes the language encoding flag when reading and ignores the unicode extra fields.
- WinZIP writes CodePage 437 and uses unicode extra fields by default. It recognizes the unicode extra field and the language encoding flag when reading.
- Windows' "compressed folder" feature doesn't recognize any flag or extra field and creates archives using the platforms default encoding - and expects archives to be in that encoding when reading them.
- InfoZIP based tools can recognize and write both, it is a compile time option and depends on the platform so your mileage may vary.
- PKWARE zip tools recognize both and prefer the language encoding flag. They create archives using CodePage 437 if possible and UTF-8 plus the language encoding flag for file names that cannot be encoded as CodePage 437.

So, what to do?

If you are creating jars, then java.util.zip is your main consumer. We recommend you set the encoding to UTF-8 and keep the language encoding flag enabled. The flag won't help or hurt java.util.zip but archivers that support it will show the correct file names.

For maximum interop it is probably best to set the encoding to UTF-8, enable the language encoding flag and create unicode extra fields when writing ZIPs. Such archives should be extracted correctly by java.util.zip, 7Zip, WinZIP, PKWARE tools and most likely InfoZIP tools. They will be unusable with Windows' "compressed folders" feature and bigger than archives without the unicode extra fields, though.

If Windows' "compressed folders" is your primary consumer, then your best option is to explicitly set the encoding to the target platform. You may want to enable creation of unicode extra fields so the tools that support them will extract the file names correctly.

Parameters specified as nested elements

any resource collection

Resource Collections are used to select groups of files to archive.

Prior to Ant 1.7 only `<fileset>` and `<zipfileset>` have been supported as nested elements.

zipgroupfileset

A `<zipgroupfileset>` allows for multiple zip files to be merged into the archive. Each file found in this fileset is added to the archive the same way that *zipfileset src* files are added.

`<zipgroupfileset>` is a fileset and supports all of its attributes and nested elements.

Examples

```
<zip destfile="${dist}/manual.zip"
     basedir="htdocs/manual"
/>
```

zips all files in the `htdocs/manual` directory into a file called `manual.zip` in the `${dist}` directory.

```
<zip destfile="${dist}/manual.zip"
     basedir="htdocs/manual"
     update="true"
/>
```

zips all files in the `htdocs/manual` directory into a file called `manual.zip` in the `${dist}` directory. If `manual.zip` doesn't exist, it is created; otherwise it is updated with the new/changed files.

```
<zip destfile="${dist}/manual.zip"
     basedir="htdocs/manual"
     excludes="mydocs/**, **/todo.html"
/>
```

zips all files in the `htdocs/manual` directory. Files in the directory `mydocs`, or files with the name `todo.html` are excluded.

```
<zip destfile="${dist}/manual.zip"
     basedir="htdocs/manual"
     includes="api/**/*.html"
     excludes="**/todo.html"
/>
```

zips all files in the `htdocs/manual` directory. Only html files under the directory `api` are zipped, and files with the name `todo.html` are excluded.

```
<zip destfile="${dist}/manual.zip">
  <fileset dir="htdocs/manual"/>
  <fileset dir="." includes="ChangeLog.txt"/>
</zip>
```

zips all files in the `htdocs/manual` directory, and also adds the file `ChangeLog.txt` in the current directory. `ChangeLog.txt` will be added to the top of the ZIP file, just as if it had been located at `htdocs/manual/ChangeLog.txt`.

```
<zip destfile="${dist}/manual.zip">
  <zipfileset dir="htdocs/manual" prefix="docs/user-guide"/>
  <zipfileset dir="." includes="ChangeLog27.txt" fullpath="docs/ChangeLog.txt"/>
  <zipfileset src="examples.zip" includes="**/*.html" prefix="docs/examples"/>
</zip>
```

zips all files in the `htdocs/manual` directory into the `docs/user-guide` directory in the archive, adds the file `ChangeLog27.txt` in the current directory as `docs/ChangeLog.txt`, and includes all the html files in `examples.zip` under `docs/examples`. The archive might end up containing the files:

```
  docs/user-guide/html/index.html
  docs/ChangeLog.txt
```

```
    docs/examples/index.html
```

The code

```
  <zip destfile="${dist}/manual.zip">
    <zipfileset dir="htdocs/manual" prefix="docs/user-guide"/>
    <zipgroupfileset dir="." includes="examples*.zip"/>
  </zip>
```

zips all files in the `htdocs/manual` directory into the `docs/user-guide` directory in the archive and includes all the files in any file that maches `examples*.zip`, such as all files within `examples1.zip` or `examples_for_brian.zip`. The same can be achieved with

```
  <zip destfile="${dist}/manual.zip">
    <mappedresources>
      <fileset dir="htdocs/manual"/>
      <globmapper from="*" to="docs/user-guide/*"/>
    </mappedresources>
    <archives>
      <zips>
        <fileset dir="." includes="examples*.zip"/>
      </zips>
    </archives>
  </zip>
```

The next example

```
<zip dest="release.zip">
  <tarfileset src="release.tar"/>
</zip>
```

re-packages a TAR archive as a ZIP archive. If Unix file permissions have been stored as part of the TAR file, they will be retained in the resulting ZIP archive.

Ear

Description

An extension of the Jar task with special treatment for files that should end up in an Enterprise Application archive.

(The Ear task is a shortcut for specifying the particular layout of a EAR file. The same thing can be accomplished by using the *prefix* and *fullpath* attributes of zipfilesets in a Zip or Jar task.)

The extended zipfileset element from the zip task (with attributes *prefix*, *fullpath*, and *src*) is available in the Ear task.

Please note that the zip format allows multiple files of the same fully-qualified name to exist within a single archive. This has been documented as causing various problems for unsuspecting users. If you wish to avoid this behavior you must set the `duplicate` attribute to a value other than its default, `"add"`.

Parameters

Attribute	Description	Required
destfile	the EAR file to create.	Yes
appxml	The deployment descriptor to use (META-INF/application.xml).	Yes, unless update is set to true
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true. Unless you set the *keepcompression* attribute to false, this will apply to the entire archive, not only the files you've added while updating.	No
keepcompression	For entries coming from existing archives (like nested *zipfileset*s or while updating the archive), keep the compression as it has been originally instead of using the *compress* attribute. Defaults false. *Since Ant 1.6*	No
encoding	The character encoding to use for filenames inside the archive. Defaults to UTF8. **It is not recommended to change this value as the created archive will most likely be unreadable for Java otherwise.** See also the discussion in the zip task page	No
filesonly	Store only file entries, defaults to false	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
filesetmanifest	behavior when a Manifest is found in a zipfileset or zipgroupfileset file is found. Valid values are "skip", "merge", and "mergewithoutmain". "merge"	No

	will merge all of the manifests together, and merge this into any other specified manifests. "mergewithoutmain" merges everything but the Main section of the manifests. Default value is "skip".	
whenmanifestonly	behavior when no files match. Valid values are "fail", "skip", and "create". Default is "create".	No
manifestencoding	The encoding used to read the JAR manifest, when a manifest file is specified.	No, defaults to the platform encoding.
index	whether to create an index list to speed up classloading. This is a JDK 1.3+ specific feature. Unless you specify additional jars with nested `indexjars` elements, only the contents of this jar will be included in the index. Defaults to false.	No
indexMetaInf	whether to include META-INF and its children in the index. Doesn't have any effect if *index* is false. Sun's jar implementation used to skip the META-INF directory and Ant followed that example. The behavior has been changed with Java 5. In order to avoid problems with Ant generated jars on Java 1.4 or earlier Ant will not include META-INF unless explicitly asked to. *Ant 1.8.0* - Defaults to false.	No
update	indicates whether to update or overwrite the destination file if it already exists. Default is "false".	No
duplicate	behavior when a duplicate file is found. Valid values are "add", "preserve", and "fail". The default value is "add".	No
roundup	Whether the file modification times will be rounded up to the next even number of seconds. Zip archives store file modification times with a granularity of two seconds, so the times will either be rounded up or down. If you round down, the archive will always seem out-of-date when you rerun the task, so the default is to round up. Rounding up may lead to a different type of problems like JSPs inside a web archive that seem to be slightly more recent than precompiled pages, rendering precompilation useless. Defaults to true. *Since Ant 1.6.2*	No
level	Non-default level at which file compression should be performed. Valid values range from 0 (no compression/fastest) to 9 (maximum compression/slowest). *Since Ant 1.7*	No
preserve0permissions	when updating an archive or adding entries from a different archive Ant will assume that a Unix permissions value of 0 (nobody is allowed to do anything to the file/directory) means that the permissions haven't been stored at all rather than real permissions and will instead apply its own default values. Set this attribute to true if you really want to preserve the original permission field.*since Ant 1.8.0*	No, default is false
useLanguageEncodingFlag	Whether to set the language encoding flag if the encoding is UTF-8. This setting doesn't have any effect if the encoding is not UTF-8. *Since Ant 1.8.0*. See also the discussion in the zip task page	No, default is true
createUnicodeExtraFields	Whether to create unicode extra fields to store the file names a second time inside the entry's metadata. Possible values are "never", "always" and "not-encodable" which will only add Unicode extra fields if the file name cannot be encoded using the specified	No, default is "never"

	encoding. *Since Ant 1.8.0.* See also the discussion in the zip task page	
fallbacktoUTF8	Whether to use UTF-8 and the language encoding flag instead of the specified encoding if a file name cannot be encoded using the specified encoding. *Since Ant 1.8.0.* See also the discussion in the zip task page	No, default is false
mergeClassPathAttributes	Whether to merge the Class-Path attributes found in different manifests (if merging manifests). If false, only the attribute of the last merged manifest will be preserved. *Since Ant 1.8.0.* unless you also set flattenAttributes to true this may result in manifests containing multiple Class-Path attributes which violates the manifest specification.	No, default is false
flattenAttributes	Whether to merge attributes occuring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute. *Since Ant 1.8.0.*	No, default is false

Nested elements

metainf

The nested `metainf` element specifies a FileSet. All files included in this fileset will end up in the `META-INF` directory of the ear file. If this fileset includes a file named `MANIFEST.MF`, the file is ignored and you will get a warning.

manifest, indexjars, service

These are inherited from <jar>

Example

```
<ear destfile="${build.dir}/myapp.ear" appxml="${src.dir}/metadata/application.xml">
  <fileset dir="${build.dir}" includes="*.jar,*.war"/>
</ear>
```

This document's new home is here

This document's new home is here

Jlink

Deprecated

This task has been deprecated. Use a zipfileset or zipgroupfileset with the Jar task or Zip task instead.

Description:

Links entries from sub-builds and libraries.

The jlink task can be used to build jar and zip files, similar to the *jar* task. However, jlink provides options for controlling the way entries from input files are added to the output file. Specifically, capabilities for merging entries from multiple zip or jar files is available.

If a mergefile is specified directly (eg. at the top level of a *mergefiles* pathelement) *and* the mergefile ends in ".zip" or ".jar", entries in the mergefile will be merged into the outfile. A file with any other extension will be added to the output file, even if it is specified in the mergefiles element. Directories specified in either the mergefiles or addfiles element are added to the output file as you would expect: all files in subdirectories are recursively added to the output file with appropriate prefixes in the output file (without merging).

In the case where duplicate entries and/or files are found among the files to be merged or added, jlink merges or adds the first entry and ignores all subsequent entries.

jlink ignores META-INF directories in mergefiles. Users should supply their own manifest information for the output file.

It is possible to refine the set of files that are being jlinked. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile*, and *defaultexcludes* attributes on the *addfiles* and *mergefiles* nested elements. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns. The patterns are relative to the *base* directory.

Parameters:

Attribute	Description	Required
outfile	the path of the output file.	Yes
compress	whether or not the output should be compressed. *true*, *yes*, or *on* result in compressed output. If omitted, output will be uncompressed (inflated).	No
mergefiles	files to be merged into the output, if possible.	At least one of mergefiles or addfiles
addfiles	files to be added to the output.	

Examples

The following will merge the entries in mergefoo.jar and mergebar.jar into out.jar. mac.jar and pc.jar will be added as single entries to out.jar.

```
<jlink compress="false" outfile="out.jar">
   <mergefiles>
     <pathelement path="${build.dir}/mergefoo.jar"/>
     <pathelement path="${build.dir}/mergebar.jar"/>
   </mergefiles>
   <addfiles>
     <pathelement path="${build.dir}/mac.jar"/>
     <pathelement path="${build.dir}/pc.zip"/>
   </addfiles>
</jlink>
```

Non-deprecated alternative to the above:

```
<jar compress="false" destfile="out.jar">
  <zipgroupfileset dir="${build.dir}">
    <include name="mergefoo.jar"/>
    <include name="mergebar.jar"/>
  </zipgroupfileset>
  <fileset dir="${build.dir}">
    <include name="mac.jar"/>
    <include name="pc.jar"/>
  </fileset>
</jar>
```

Suppose the file foo.jar contains two entries: bar.class and barnone/myClass.zip. Suppose the path for file foo.jar is build/tempbuild/foo.jar. The following example will provide the entry tempbuild/foo.jar in the out.jar.

```
<jlink compress="false" outfile="out.jar">
   <mergefiles>
     <pathelement path="build/tempbuild"/>
   </mergefiles>
</jlink>
```

However, the next example would result in two top-level entries in out.jar, namely bar.class and barnone/myClass.zip

```
<jlink compress="false" outfile="out.jar">
   <mergefiles>
     <pathelement path="build/tempbuild/foo.jar"/>
   </mergefiles>
</jlink>
```

Manifest

Description

Creates a manifest file.

This task can be used to write a Manifest file, optionally replacing or updating an existing file.

Manifests are processed according to the Jar file specification.. Specifically, a manifest element consists of a set of attributes and sections. These sections in turn may contain attributes. Note in particular that this may result in manifest lines greater than 72 bytes being wrapped and continued on the next line.

The Ant team regularly gets complaints that this task in generating invalid manifests. By and large, this is not the case: we believe that we are following the specification to the letter. The usual problem is that some third party manifest reader is not following the same specification as well as they think they should; we cannot generate invalid manifest files just because one single application is broken. J2ME runtimes appear to be particularly troublesome.

If you find that Ant generates manifests incompatible with your runtime, take a manifest it has built, fix it up however you need and switch to using the <zip> task to create the JAR, feeding in the hand-crafted manifest.

Parameters

Attribute	Description	Required
file	the manifest-file to create/update.	Yes
mode	One of "update" or "replace", default is "replace".	No
encoding	The encoding used to read the existing manifest when updating. The task will always use UTF-8 when writing the manifest.	No, defaults to UTF-8 encoding.
mergeClassPathAttributes	Whether to merge the Class-Path attributes found in different manifests (if updating). If false, only the attribute of the most recent manifest will be preserved. *Since Ant 1.8.0*. unless you also set flattenAttributes to true this may result in manifests containing multiple Class-Path attributes which violates the manifest specification.	No, default is false
flattenAttributes	Whether to merge attributes occuring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute. *Since Ant 1.8.0*.	No, default is false

Nested elements

attribute

One attribute for the manifest file. Those attributes that are not nested into a section will be added to the "Main" section.

Attribute	Description	Required
name	the name of the attribute, must match the regexp `[A-Za-z0-9][A-Za-z0-9-_]*`.	Yes

value	the value of the attribute.	Yes

section

A manifest section - you can nest attribute elements into sections.

Attribute	Description	Required
name	the name of the section.	No, if omitted it will be assumed to be the main section.

Examples

```
  <manifest file="MANIFEST.MF">
    <attribute name="Built-By" value="${user.name}"/>
    <section name="common">
      <attribute name="Specification-Title" value="Example"/>
      <attribute name="Specification-Version" value="${version}"/>
      <attribute name="Specification-Vendor" value="Example Organization"/>
      <attribute name="Implementation-Title" value="common"/>
      <attribute name="Implementation-Version" value="${version} ${TODAY}"/>
      <attribute name="Implementation-Vendor" value="Example Corp."/>
    </section>
    <section name="common/class1.class">
      <attribute name="Sealed" value="false"/>
    </section>
  </manifest>
```

Creates or replaces the file MANIFEST.MF. Note that the Built-By attribute will take the value of the Ant property ${user.name}. The same is true for the ${version} and ${TODAY} properties. This example produces a MANIFEST.MF that contains package version identification for the package `common`.

The manifest produced by the above would look like this:

```
Manifest-Version: 1.0
Built-By: bodewig
Created-By: Apache Ant 1.7

Name: common
Specification-Title: Example
Specification-Vendor: Example Organization
Implementation-Vendor: Example Corp.
Specification-Version: 1.2
Implementation-Version: 1.2 February 19 2006
Implementation-Title: common

Name: common/class1.class
Sealed: false
```

Rpm

Description

A basic task for invoking the rpm executable to build a RedHat Package Manager Linux installation file. The task currently only works on Linux or other Unix platforms with rpm support.

Parameters

Attribute	Description	Required
specFile	The name of the spec file to be used. This must be relative to the SPECS directory under the root of the RPM set in the topDir attribute.	Yes
topDir	This is the directory which will have the expected subdirectories, SPECS, SOURCES, BUILD, SRPMS. If this isn't specified, the default RPM directory of the system (or user, if ~/.rpmmacros defines it) is used (often /usr/src/rpm. Defining a topdir will set `%_topdir` to the specified directory -there is no need to edit your .rpmmacros file.	No, but your build file is very brittle if it is not set.
cleanBuildDir	This will remove the generated files in the BUILD directory. See the the `--clean` option of rpmbuild.	No
removeSpec	This will remove the spec file from SPECS. See the the `--rmspec` option of rpmbuild.	No
removeSource	Flag (optional, default=false) to remove the sources after the build. See the the `--rmsource` option of rpmbuild.	No
rpmBuildCommand	The executable to use for building the RPM. Defaults to `rpmbuild` if it can be found or `rpm` otherwise. Set this if you don't have either on your PATH or want to use a different executable. *Since Ant 1.6*.	No
command	The command to pass to the rpmbuild program. The default is "-bb"	No
quiet	Suppress output. Defaults to false.	No
output/error	Where standard output and error go	No
failOnError	Stop the buildprocess if the RPM build command exits with a non-zero retuncode. Defaults to false	No

Examples

```
    <rpm
        specFile="example.spec"
        topDir="build/rpm"
        cleanBuildDir="true"
        failOnError="true"/>
```

SignJar

Description

Signs JAR files with the `jarsigner` command line tool. It will take a named file in the `jar` attribute, and an optional `destDir` or `signedJar` attribute. Nested paths are also supported; here only an (optional) `destDir` is allowed. If a destination directory or explicit JAR file name is not provided, JARs are signed in place.

Dependency rules

- Nonexistent destination JARs are created/signed
- Out of date destination JARs are created/signed
- If a destination file and a source file are the same, and `lazy` is true, the JAR is only signed if it does not contain a signature by this alias.
- If a destination file and a source file are the same, and `lazy` is false, the JAR is signed.

Parameters

Attribute	Description	Required
jar	the jar file to sign	Yes, unless nested paths have been used.
alias	the alias to sign under	Yes.
storepass	password for keystore integrity.	Yes.
keystore	keystore location	No
storetype	keystore type	No
keypass	password for private key (if different)	No
sigfile	name of .SF/.DSA file	No
signedjar	name of signed JAR file. This can only be set when the `jar` attribute is set.	No.
verbose	(true | false) verbose output when signing	No; default false
internalsf	(true | false) include the .SF file inside the signature block	No; default false
sectionsonly	(true | false) don't compute hash of entire manifest	No; default false
lazy	flag to control whether the presence of a signature file means a JAR is signed. This is only used when the target JAR matches the source JAR	No; default false
maxmemory	Specifies the maximum memory the jarsigner VM will use. Specified in the style of standard java memory specs (e.g. 128m = 128 MBytes)	No
preservelastmodified	Give the signed files the same last modified time as the original jar files.	No; default false.
tsaurl	URL for a timestamp authority for timestamped JAR files in Java1.5+	No
tsacert	alias in the keystore for a timestamp authority for timestamped JAR files in Java1.5+	No
executable	Specify a particular `jarsigner` executable to use in place of the default binary (found in the same JDK as Ant is running in). Must support the same command line options as the Sun JDK jarsigner command. *since Ant 1.8.0*.	No

force	Whether to force signing of the jar file even if it doesn't seem to be out of date or already signed. *since Ant 1.8.0*.	No; default false

Parameters as nested elements

Attribute	Description	Required
path	path of JAR files to sign. *since Ant 1.7*	No
fileset	fileset of JAR files to sign.	No
mapper	A mapper to rename jar files during signing	No, and only one can be supplied
sysproperty	JVM system properties, with the syntax of Ant environment variables	No, and only one can be supplied

Examples

```
<signjar jar="${dist}/lib/ant.jar"
alias="apache-group" storepass="secret"/>
```

signs the ant.jar with alias "apache-group" accessing the keystore and private key via "secret" password.

```
<signjar destDir="signed"
    alias="testonly" keystore="testkeystore"
    storepass="apacheant"
    preservelastmodified="true">
  <path>
    <fileset dir="dist" includes="**/*.jar" />
  </path>
  <flattenmapper />
</signjar>
```

Sign all JAR files matching the dist/**/*.jar pattern, copying them to the directory "signed" afterwards. The flatten mapper means that they will all be copied to this directory, not to subdirectories.

```
<signjar
    alias="testonly" keystore="testkeystore"
    storepass="apacheant"
    lazy="true"
    >
  <path>
    <fileset dir="dist" includes="**/*.jar" />
  </path>
</signjar>
```

Sign all the JAR files in dist/**/*.jar *in-situ*. Lazy signing is used, so the files will only be signed if they are not already signed.

About timestamp signing

Timestamped JAR files are a new feature in Java1.5; a feature supported in Ant since Ant 1.7. Ant does not yet support proxy setup for this singing process, and the whole TSA feature is not tested yet. Furthermore, the official TSA documentation warns that the API is subject to change. If a future version of Java changes the API, Ant will break. It may be possible to hide changes if and when they occur, but this can not be guaranteed.

Tar

Description

Creates a tar archive.

The *basedir* attribute is the reference directory from where to tar.

This task is a directory based task and, as such, forms an implicit Fileset. This defines which files, relative to the *basedir*, will be included in the archive. The tar task supports all the attributes of Fileset to refine the set of files to be included in the implicit fileset.

In addition to the implicit fileset, the tar task supports nested resource collections and a special form of filesets. These filesets are extended to allow control over the access mode, username and groupname to be applied to the tar entries. This is useful, for example, when preparing archives for Unix systems where some files need to have execute permission. By default this task will use Unix permissions of 644 for files and 755 for directories.

Early versions of tar did not support path lengths greater than 100 characters. Modern versions of tar do so, but in incompatible ways. The behaviour of the tar task when it encounters such paths is controlled by the *longfile* attribute. If the longfile attribute is set to `fail`, any long paths will cause the tar task to fail. If the longfile attribute is set to `truncate`, any long paths will be truncated to the 100 character maximum length prior to adding to the archive. If the value of the longfile attribute is set to `omit` then files containing long paths will be omitted from the archive. Either option ensures that the archive can be untarred by any compliant version of tar. If the loss of path or file information is not acceptable, and it rarely is, longfile may be set to the value `gnu`. The tar task will then produce a GNU tar file which can have arbitrary length paths. Note however, that the resulting archive will only be able to be untarred with GNU tar. The default for the longfile attribute is `warn` which behaves just like the gnu option except that it produces a warning for each file path encountered that does not match the limit.

This task can perform compression by setting the compression attribute to "gzip" or "bzip2".

Parameters

Attribute	Description	Required
destfile	the tar-file to create.	Yes
basedir	the directory from which to tar the files.	No
longfile	Determines how long files (>100 chars) are to be handled. Allowable values are "truncate", "fail", "warn", "omit" and "gnu". Default is "warn".	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
compression	compression method. Allowable values are "none", "gzip" and "bzip2". Default is "none".	No

Nested Elements

The tar task supports nested tarfileset elements. These are extended FileSets which, in addition to the standard elements, support one additional attributes

Attribute	Description	Required
preserveLeadingSlashes	Indicates whether leading `/'s should be preserved in the file names. Default is `false`.	No

any other resource collection

Resource Collections are used to select groups of files to archive.

Prior to Ant 1.7 only `<fileset>` has been supported as a nested element.

Examples

```
<tar destfile="${dist}/manual.tar" basedir="htdocs/manual"/>
<gzip destfile="${dist}/manual.tar.gz" src="${dist}/manual.tar"/>
```

tars all files in the `htdocs/manual` directory into a file called `manual.tar` in the `${dist}` directory, then applies the gzip task to compress it.

```
<tar destfile="${dist}/manual.tar"
     basedir="htdocs/manual"
     excludes="mydocs/**, **/todo.html"
/>
```

tars all files in the `htdocs/manual` directory into a file called `manual.tar` in the `${dist}` directory. Files in the directory `mydocs`, or files with the name `todo.html` are excluded.

```
<tar destfile="${basedir}/docs.tar">
  <tarfileset dir="${dir.src}/docs"
              fullpath="/usr/doc/ant/README"
              preserveLeadingSlashes="true">
    <include name="readme.txt"/>
  </tarfileset>
  <tarfileset dir="${dir.src}/docs"
              prefix="/usr/doc/ant"
              preserveLeadingSlashes="true">
    <include name="*.html"/>
  </tarfileset>
</tar>
```

Writes the file `docs/readme.txt` as `/usr/doc/ant/README` into the archive. All `*.html` files in the `docs` directory are prefixed by `/usr/doc/ant`, so for example `docs/index.html` is written as `/usr/doc/ant/index.html` to the archive.

```
<tar longfile="gnu"
     destfile="${dist.base}/${dist.name}-src.tar">
  <tarfileset dir="${dist.name}/.." filemode="755" username="ant" group="ant">
    <include name="${dist.name}/bootstrap.sh"/>
    <include name="${dist.name}/build.sh"/>
  </tarfileset>
  <tarfileset dir="${dist.name}/.." username="ant" group="ant">
    <include name="${dist.name}/**"/>
    <exclude name="${dist.name}/bootstrap.sh"/>
    <exclude name="${dist.name}/build.sh"/>
  </tarfileset>
</tar>
```

This example shows building a tar which uses the GNU extensions for long paths and where some files need to be marked as executable (mode 755) and the rest are use the default mode (read-write by owner). The first fileset selects just the executable files. The second fileset must exclude the executable files and include all others.

Note: The tar task does not ensure that a file is only selected by one resource collection. If the same file is selected by more than one collection, it will be included in the tar file twice, with the same path.

Note: The patterns in the include and exclude elements are considered to be relative to the corresponding dir attribute as with all other filesets. In the example above, `${dist.name}` is not an absolute path, but a simple name of a directory, so `${dist.name}` is a valid path relative to `${dist.name}/...`

```
<tar destfile="release.tar.gz" compression="gzip">
  <zipfileset src="release.zip"/>
</tar>
```

Re-packages a ZIP archive as a GZip compressed tar archive. If Unix file permissions have been stored as part of the ZIP file, they will be retained in the resulting tar archive.

Note: Please note the tar task creates a tar file, it does not append to an existing tar file. The existing tar file is replaced instead. As with most tasks in Ant, the task only takes action if the output file (the tar file in this case) is older than the input files, or if the output file does not exist.

Unjar/Untar/Unwar/Unzip

Description

Unzips a zip-, war-, or jar file.

PatternSets are used to select files to extract *from* the archive. If no patternset is used, all files are extracted.

Resource Collections may be used to select archived files to perform unarchival upon. Only file system based resource collections are supported by Unjar/Unwar/Unzip, this includes fileset, filelist, path, and files. Untar supports arbitrary resource collections. Prior to Ant 1.7 only fileset has been supported as a nested element.

You can define filename transformations by using a nested mapper element. The default mapper is the identity mapper.

File permissions will not be restored on extracted files.

The untar task recognizes the long pathname entries used by GNU tar.

Please note that different ZIP tools handle timestamps differently when it comes to applying timezone offset calculations of files. Some ZIP libraries will store the timestamps as they've been read from the filesystem while others will modify the timestamps both when reading and writing the files to make all timestamps use the same timezone. A ZIP archive created by one library may extract files with "wrong timestamps" when extracted by another library.

Ant's ZIP classes use the same algorithm as the InfoZIP tools and zlib (timestamps get adjusted), Windows' "compressed folders" function and WinZIP don't change the timestamps. This means that using the unzip task on files created by Windows' compressed folders function may create files with timestamps that are "wrong", the same is true if you use Windows' functions to extract an Ant generated ZIP archive.

Parameters

Attribute	Description	Required
src	archive file to expand.	Yes, if filesets are not used.
dest	directory where to store the expanded files.	Yes
overwrite	Overwrite files, even if they are newer than the corresponding entries in the archive (true or false, default is true).	No
compression	**Note:** This attribute is only available for the `untar` task. compression method. Allowable values are "none", "gzip" and "bzip2". Default is "none".	No
encoding	**Note:** This attribute is not available for the `untar` task. The character encoding that has been used for filenames inside the zip file. For a list of possible values see http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html. Defaults to "UTF8", use the magic value `native-encoding` for the platform's default character encoding. See also the discussion in the zip task page	No
failOnEmptyArchive	whether trying to extract an empty archive is an error. *since Ant 1.8.0*	No,

		defaults to false
stripAbsolutePathSpec	whether Ant should remove leading '/' or '\' characters from the extracted file name before extracing it. Note that this changes the entry's name before applying include/exclude patterns and before using the nested mappers (if any). *since Ant 1.8.0*	No, defaults to false
scanForUnicodeExtraFields	**Note:** This attribute is not available for the `untar` task. If the archive contains uncode extra fields then use them to set the file names, ignoring the specified encoding. See also the discussion in the zip task page	No, defaults to true

Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}"/>

<gunzip src="tools.tar.gz"/>
<untar src="tools.tar" dest="${tools.home}"/>

<unzip src="${tomcat_src}/tools-src.zip"
       dest="${tools.home}">
    <patternset>
        <include name="**/*.java"/>
        <exclude name="**/Test*.java"/>
    </patternset>
</unzip>

<unzip dest="${tools.home}">
    <patternset>
        <include name="**/*.java"/>
        <exclude name="**/Test*.java"/>
    </patternset>
    <fileset dir=".">
        <include name="**/*.zip"/>
        <exclude name="**/tmp*.zip"/>
    </fileset>
</unzip>

<unzip src="apache-ant-bin.zip" dest="${tools.home}">
    <patternset>
        <include name="apache-ant/lib/ant.jar"/>
    </patternset>
    <mapper type="flatten"/>
</unzip>
```

Related tasks

```
<unzip src="some-archive" dest="some-dir">
  <patternset>
    <include name="some-pattern"/>
  </patternset>
  <mapper type="some-mapper"/>
</unzip>
```

is identical to

```
<copy todir="some-dir" preservelastmodified="true">
  <zipfileset src="some-archive">
    <patternset>
      <include name="some-pattern"/>
    </patternset>
  </zipfileset>
  <mapper type="some-mapper"/>
</copy>
```

The same is also true for `<untar>` and `<tarfileset>`. `<copy>` offers additional features like filtering files on the fly, allowing a file to be mapped to multiple destinations or a configurable file system timestamp granularity.

```
<zip destfile="new.jar">
  <zipfileset src="old.jar">
    <exclude name="do/not/include/this/class"/>
  </zipfileset>
</zip>
```

"Deletes" files from a zipfile.

```
<unzip src="${ant.home}/lib/ant.jar" dest="...">
  <patternset>
    <include name="images/"/>
  </patternset>
</unzip>
```

This extracts all images from `ant.jar` which are stored in the `images` directory of the Jar (or somewhere under it). While extracting the directory structure (`images`) will be taken.

```
<unzip src="${ant.home}/lib/ant.jar" dest="...">
  <patternset>
    <include name="**/ant_logo_large.gif"/>
    <include name="**/LICENSE.txt"/>
  </patternset>
</unzip>
```

This extracts the two files `ant_logo_large.gif` and `LICENSE.txt` from the `ant.jar`. More exactly: it extracts all files with these names from anywhere in the source file. While extracting the directory structure will be taken.

Untar

Description

Untars a tarfile.

This document has moved here

War

Description

An extension of the Jar task with special treatment for files that should end up in the `WEB-INF/lib`, `WEB-INF/classes` or `WEB-INF` directories of the Web Application Archive.

(The War task is a shortcut for specifying the particular layout of a WAR file. The same thing can be accomplished by using the *prefix* and *fullpath* attributes of zipfilesets in a Zip or Jar task.)

The extended zipfileset element from the zip task (with attributes *prefix*, *fullpath*, and *src*) is available in the War task. The task is also resource-enabled and will add nested resources and resource collections to the archive.

Before Servlet API 2.5/Java EE 5, a WEB-INF/web.xml file was mandatory in a WAR file, so this task failed if the `webxml` attribute was missing. As the web.xml file is now optional, the `webxml` attribute may now be made optional. However, as most real web applications do need a web.xml file, it is not optional by default. The task will fail if the file is not included, unless the `needxmlfile` attribute is set to `false`. The task will warn if more than one web.xml file is added to the JAR through the filesets.

Please note that the Zip format allows multiple files of the same fully-qualified name to exist within a single archive. This has been documented as causing various problems for unsuspecting users. If you wish to avoid this behavior you must set the `duplicate` attribute to a value other than its default, `"add"`.

Parameters

Attribute	Description	Required
destfile	the WAR file to create.	Exactly one of the two.
warfile	*Deprecated* name of the file to create -use `destfile` instead.	
webxml	The servlet configuration descriptor to use (WEB-INF/web.xml).	Yes, unless `needxmlfile` is true, the file is pulled in via a nested fileset, or an existing WAR file is being updated.
needxmlfile	Flag to indicate whether or not the web.xml file is needed. It should be set to false when generating servlet 2.5+ WAR files without a web.xml file. *Since Ant 1.7*	No -default "true"
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true. Unless you set the *keepcompression* attribute to false, this will apply to the entire archive, not only the files you've added while updating.	No
keepcompression	For entries coming from existing archives (like nested *zipfileset*s or while updating the archive), keep the compression as it has been originally instead of using the *compress* attribute. Defaults false. *Since Ant 1.6*	No
encoding	The character encoding to use for filenames inside the archive.	No

	Defaults to UTF8. **It is not recommended to change this value as the created archive will most likely be unreadable for Java otherwise.** See also the discussion in the zip task page	
filesonly	Store only file entries, defaults to false	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
filesetmanifest	behavior when a Manifest is found in a zipfileset or zipgroupfileset file is found. Valid values are "skip", "merge", and "mergewithoutmain". "merge" will merge all of the manifests together, and merge this into any other specified manifests. "mergewithoutmain" merges everything but the Main section of the manifests. Default value is "skip".	No
whenmanifestonly	behavior when no files match. Valid values are "fail", "skip", and "create". Default is "create".	No
update	indicates whether to update or overwrite the destination file if it already exists. Default is "false".	No
duplicate	behavior when a duplicate file is found. Valid values are "add", "preserve", and "fail". The default value is "add".	No
roundup	Whether the file modification times will be rounded up to the next even number of seconds. Zip archives store file modification times with a granularity of two seconds, so the times will either be rounded up or down. If you round down, the archive will always seem out-of-date when you rerun the task, so the default is to round up. Rounding up may lead to a different type of problems like JSPs inside a web archive that seem to be slightly more recent than precompiled pages, rendering precompilation useless. Defaults to true. *Since Ant 1.6.2*	No
level	Non-default level at which file compression should be performed. Valid values range from 0 (no compression/fastest) to 9 (maximum compression/slowest). *Since Ant 1.7*	No
preserve0permissions	when updating an archive or adding entries from a different archive Ant will assume that a Unix permissions value of 0 (nobody is allowed to do anything to the file/directory) means that the permissions haven't been stored at all rather than real permissions and will instead apply its own default values. Set this attribute to true if you really want to preserve the original permission field.*since Ant 1.8.0*	No, default is false

useLanguageEncodingFlag	Whether to set the language encoding flag if the encoding is UTF-8. This setting doesn't have any effect if the encoding is not UTF-8. *Since Ant 1.8.0*. See also the discussion in the zip task page	No, default is true
createUnicodeExtraFields	Whether to create unicode extra fields to store the file names a second time inside the entry's metadata. Possible values are "never", "always" and "not-encodable" which will only add Unicode extra fields if the file name cannot be encoded using the specified encoding. *Since Ant 1.8.0*. See also the discussion in the zip task page	No, default is "never"
fallbacktoUTF8	Whether to use UTF-8 and the language encoding flag instead of the specified encoding if a file name cannot be encoded using the specified encoding. *Since Ant 1.8.0*. See also the discussion in the zip task page	No, default is false
mergeClassPathAttributes	Whether to merge the Class-Path attributes found in different manifests (if merging manifests). If false, only the attribute of the last merged manifest will be preserved. *Since Ant 1.8.0*. unless you also set flattenAttributes to true this may result in manifests containing multiple Class-Path attributes which violates the manifest specification.	No, default is false
flattenAttributes	Whether to merge attributes occuring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute. *Since Ant 1.8.0*.	No, default is false

Nested elements

lib

The nested `lib` element specifies a FileSet. All files included in this fileset will end up in the `WEB-INF/lib` directory of the war file.

classes

The nested `classes` element specifies a FileSet. All files included in this fileset will end up in the `WEB-INF/classes` directory of the war file.

webinf

The nested `webinf` element specifies a FileSet. All files included in this fileset will end up in the `WEB-INF` directory of the war file. If this fileset includes a file named `web.xml`, the file is ignored and you will get a warning.

metainf

The nested `metainf` element specifies a FileSet. All files included in this fileset will end up in the `META-INF` directory of the war file. If this fileset includes a file named `MANIFEST.MF`, the file is ignored and you will get a warning.

manifest, indexjars, service

These are inherited from <jar>

Examples

Assume the following structure in the project's base directory:

```
thirdparty/libs/jdbc1.jar
thirdparty/libs/jdbc2.jar
build/main/com/myco/myapp/Servlet.class
src/metadata/myapp.xml
src/html/myapp/index.html
src/jsp/myapp/front.jsp
src/graphics/images/gifs/small/logo.gif
src/graphics/images/gifs/large/logo.gif
```

then the war file `myapp.war` created with

```
<war destfile="myapp.war" webxml="src/metadata/myapp.xml">
  <fileset dir="src/html/myapp"/>
  <fileset dir="src/jsp/myapp"/>
  <lib dir="thirdparty/libs">
    <exclude name="jdbc1.jar"/>
  </lib>
  <classes dir="build/main"/>
  <zipfileset dir="src/graphics/images/gifs"
              prefix="images"/>
</war>
```

will consist of

```
WEB-INF/web.xml
WEB-INF/lib/jdbc2.jar
WEB-INF/classes/com/myco/myapp/Servlet.class
META-INF/MANIFEST.MF
index.html
front.jsp
images/small/logo.gif
images/large/logo.gif
```

using Ant's default manifest file. The content of `WEB-INF/web.xml` is identical to `src/metadata/myapp.xml`.

We regulary receive bug reports that this task is creating the WEB-INF directory as web-inf (all lower case), and thus it is our fault your webapp doesn't work. The cause of these complaints lies in WinZip, which turns an all upper-case directory into an all lower case one in a fit of helpfulness. Please check that `jar xvf yourwebapp.war` shows the same behaviour before filing another report.
Winzip has an option allowing all uppercase names (which is off by default!). It can be enabled by: Menu "Options" -> "Configuration", "View" property/tab page, then "General" group box has an option called "Allow all uppercase file names".

JDepend

Description

Invokes the JDepend parser.

This parser "traverses a set of Java source file directories and generates design quality metrics for each Java package". It allows to "automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to effectively manage and control package dependencies."

Source file directories are defined by nested `<sourcespath>`; Class file directories are defined by nested `<classesespath>`, see nested elements.

Optionally, you can also set the `outputfile` name where the output is stored. By default the task writes its report to the standard output.

The task requires at least the JDepend 1.2 version.

Parameters

Attribute	Description	Required
outputfile	The output file name. If not set, the output is printed on the standard output.	No
format	The format to write the output in. The default is "text", the alternative is "xml"	No
fork	Run the tests in a separate VM.	No, default is "off"
haltonerror	Stop the build process if an error occurs during the jdepend analysis.	No, default is "off"
timeout	Cancel the operation if it doesn't finish in the given time (measured in milliseconds). (Ignored if fork is disabled.)	No
jvm	The command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by java.lang.Runtime.exec(). (Ignored if fork is disabled.)	No, default "java"
dir	The directory to invoke the VM in. (Ignored if fork is disabled)	No
includeruntime	Implicitly add the classes required to run jdepend in forked mode. (Ignored if fork is disabled). Since ant 1.6.	No, default is "no".
classpathref	the classpath to use, given as reference to a PATH defined elsewhere.	No

Nested Elements

`jdepend` supports four nested elements: `<classpath>`, `<classesepath>` and `<sourcespath>`, that represent PATH like structures, and `<exclude>`.

`<sourcespath>` is used to define the paths of the source code to analyze, but it is deprecated. With version 2.5 of JDepend, only class files are analyzed. The nested element `<classespath>` replaces `<sourcespath>` and is used to

define the paths of compiled class code to analyze; the <sourcespath> variable is still available in case you are using an earlier version of JDepend. The <exclude> element can be used to set packages to ignore (requires JDepend 2.5 or above).

Examples

```
<jdepend classpathref="base.path">
    <classespath>
        <pathelement location="build"/>
    </classespath>
</jdepend>
```

This invokes JDepend on the build directory, writing the output on the standard output. The classpath is defined using a classpath reference.

```
<jdepend outputfile="docs/jdepend.xml" fork="yes" format="xml">
    <sourcespath>
        <pathelement location="src"/>
    </sourcespath>
    <classpath>
        <pathelement location="classes"/>
        <pathelement location="lib/jdepend.jar"/>
    </classpath>
</jdepend>
```

This invokes JDepend in a separate VM on the src and testsrc directories, writing the output to the <docs/jdepend.xml> file in xml format. The classpath is defined using nested elements.

```
<jdepend classpathref="base.path">
    <exclude name="java.*>
    <exclude name="javax.*>
    <classespath>
        <pathelement location="build"/>
    </classespath>
</jdepend>
```

This invokes JDepend with the build directory as the base for class files to analyze, and will ignore all classes in the java.* and javax.* packages.

Depend

A task to manage Java class file dependencies.

Description

The depend task works by determining which classes are out of date with respect to their source and then removing the class files of any other classes which depend on the out-of-date classes.

To determine the class dependencies, the depend task analyses the class files of all class files passed to it. Depend does not parse your source code in any way but relies upon the class references encoded into the class files by the compiler. This is generally faster than parsing the Java source.

To learn more about how this information is obtained from the class files, please refer to the Java Virtual Machine Specification

Since a class' dependencies only change when the class itself changes, the depend task is able to cache dependency information. Only those class files which have changed will have their dependency information re-analysed. Note that if you change a class' dependencies by changing the source, it will be recompiled anyway. You can examine the dependency files created to understand the dependencies of your classes. Please do not rely, however, on the format of the information, as it may change in a later release.

Once depend discovers all of the class dependencies, it "inverts" this relation to determine, for each class, which other classes are dependent upon it. This "affects" list is used to discover which classes are invalidated by the out of date class. The class files of the invalidated classes are removed, triggering the compilation of the affected classes.

The depend task supports an attribute, "closure" which controls whether depend will only consider direct class-class relationships or whether it will also consider transitive, indirect relationships. For example, say there are three classes, A, which depends on B, which in-turn depend on C. Now say that class C is out of date. Without closure, only class B would be removed by depend. With closure set, class A would also be removed. Normally direct relationships are sufficient - it is unusual for a class to depend on another without having a direct relationship. With closure set, you will notice that depend typically removes far more class files.

The classpath attribute for `<depend>` is optional. If it is present, depend will check class dependencies against classes and jars on this classpath. Any classes which depend on an element from this classpath and which are older than that element will be deleted. A typical example where you would use this facility would be where you are building a utility jar and want to make sure classes which are out of date with respect to this jar are rebuilt. You should **not** include jars in this classpath which you do not expect to change, such as the JDK runtime jar or third party jars, since doing so will just slow down the dependency check. This means that if you do use a classpath for the depend task it may be different from the classpath necessary to actually compile your code.

Performance

The performance of the depend task is dependent on a number of factors such as class relationship complexity and how many class files are out of date. The decision about whether it is cheaper to just recompile all classes or to use the depend task will depend on the size of your project and how interrelated your classes are.

Limitations

There are some source dependencies which depend will not detect.

- If the Java compiler optimizes away a class relationship, there can be a source dependency without a class dependency.
- Non public classes cause two problems. Firstly depend cannot relate the class file to a source file. In the future this may be addressed using the source file attribute in the classfile. Secondly, neither depend nor the compiler tasks can detect when a non public class is missing. Inner classes are handled by the depend task.

The most obvious example of these limitations is that the task can't tell which classes to recompile when a constant primitive data type exported by other classes is changed. For example, a change in the definition of something like

```
public final class Constants {
  public final static boolean DEBUG=false;
}
```

will not be picked up by other classes.

Parameters

Attribute	Description	Required
srcDir	This is the directory where the source exists. depend will examine this to determine which classes are out of date. If you use multiple source directories you can pass this attribute a path of source directories.	Yes
destDir	This is the root directory of the class files which will be analysed. If this is not present, the srcdir is used.	No
cache	This is a directory in which depend can store and retrieve dependency information. If this is not present, depend will not use a cache	No
closure	This attribute controls whether depend only removes classes which directly depend on out of date classes. If this is set to true, depend will traverse the class dependency graph deleting all affected classes. Defaults to false	No
dump	If true the dependency information will be written to the debug level log	No
classpath	The classpath containing jars and classes for which `<depend>` should also check dependencies	No
warnOnRmiStubs	Flag to disable warnings about files that look like rmic generated stub/skeleton classes, and which have no .java source. Useful when doing rmi development.	No, default=true

Parameters specified as nested elements

The `depend` task's `classpath` attribute is a PATH-like structure and can also be set via a nested `<classpath>` element.

Additionally, this task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `srcdir`), as well as the nested `<include>`, `<exclude>`, and `<patternset>` elements.

Examples

```
<depend srcdir="${java.dir}"
        destdir="${build.classes}"
        cache="depcache"
        closure="yes"/>
```

removes any classes in the `${build.classes}` directory that depend on out-of-date classes. Classes are considered out-of-date with respect to the source in the `${java.dir}` directory, using the same mechanism as the `<javac>` task. In this example, the `<depend>` task caches its dependency information in the `depcache` directory.

```
<depend srcdir="${java.dir}" destdir="${build.classes}"
```

```
        cache="depcache" closure="yes">
  <include name="**/*.java"/>
  <excludesfile name="${java.dir}/build_excludes"/>
</depend>
```

does the same as the previous example, but explicitly includes all `.java` files, except those that match the list given in `${java.dir}/build_excludes`.

Apt

Description

Runs the annotation processor tool (apt), and then optionally compiles the original code, and any generated source code. This task requires Java 1.5. It may work on later versions, but this cannot be confirmed until those versions ship. Be advised that the Apt tool does appear to be an unstable part of the JDK framework, so may change radically in future versions. In particular it is likely to be obsolete in JDK 6, which can run annotation processors as part of javac. If the <apt> task does break when upgrading JVM, please check to see if there is a more recent version of Ant that tracks any changes.

This task inherits from the Javac Task, and thus supports nearly all of the same attributes, and subelements. There is one special case, the `fork` attribute, which is present but which can only be set to `true`. That is, apt only works as a forked process.

In addition, it supports the following addition items:

Parameters

Attribute	Description	Required
compile	After running the Apt, should the code be compiled. (see the `-nocompile` flag on the Apt executable)	No, defaults to false.
factory	The fully qualified classname of the AnnotationProcessFactory to be used to construct annotation processors. This represents the `-factory` command line flag of the Apt executable.	No
factorypathref	The reference id of the path used to find the classes needed by the AnnotationProcessorFactory (and the location of the factory itself). This represents the `-factorypath` flag on the Apt executable.	No
preprocessdir	The directory used for preprocessing. This is the directory where the generated source code will be place. This represents the `-s` flag on the Apt executable.	No

Parameters specified as nested elements

factorypath

You can specify the path used to find the classes needed by the AnnotationProcessorFactory at runtime, using this element. It is represents as a generic path like structure. This represents the `-factorypath` flag on the Apt executable.

option

Used to represent a generic option to pass to Apt. This represents the `-A` flag on the Apt executable. You can specify zero or more `<option>` elements.

Attribute	Description	Required
name	The name of the option	Yes.
value	The value to set the option to	Yes.

Examples

```
<apt srcdir="${src}"
     destdir="${build}"
     classpath="xyz.jar"
     debug="on"
     compile="true"
     factory="com.mycom.MyAnnotationProcessorFactory"
     factorypathref="my.factorypath.id"
     preprocessdir="${preprocess.dir}">
</apt>
```

compiles all `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. The classpath used includes `xyz.jar`, and compiling with debug information is on. It also forces the generated source code to be compiled. The generated source code will be placed in `${preprocess.dir}` directory, using the class `com.mycom.MyAnnotationProcessorFactory` to supply AnnotationProcessor instances.

Notes

The inherited "fork" attribute is set to true by default; please do not change it.

The inherited "compiler" attribute is ignored, as it is forced to use the Apt compiler

Using the Apt compiler with the "compile" option set to "true" forces you to use Sun's Apt compiler, which will use the JDK's Javac compiler. If you wish to use another compiler, you will first need run the Apt processor with the "compile" flag set to "false", and then use a `<javac>` task to compile first your original source code, and then the generated source code:

```
<apt srcdir="${src}"
     destdir="${build}"
     classpath="xyz.jar"
     debug="true"
     compile="false"
     factory="com.mycom.MyAnnotationProcessorFactory"
     factorypathref="my.factorypath.id"
     preprocessdir="${preprocess.dir}">
</apt>

<javac srcdir="${src}"
       destdir="${build}"
       classpath="xyz.jar"
       debug="on"/>

<javac srcdir="${preprocess.dir}"
       destdir="${build}"
       classpath="xyz.jar"
       debug="true"/>
```

This may involve more build file coding, but the speedup gained from switching to jikes may justify the effort.

jspc (deprecated)

Description

Ant task to run the JSP compiler and turn JSP pages into Java source.

Deprecated if you use this task with Tomcat's Jasper JSP compiler, you should seriously consider using the task shipping with Tomcat instead. This task is only tested against Tomcat 4.x. There are known problems with Tomcat 5.x that won't get fixed in Ant, please use Tomcat's jspc task instead.
Instead of relying on container specific JSP-compilers we suggest deploying the raw files (*.jsp) and use the container build-in functions: after deploying run a test suite (e.g. with Cactus or HttpUnit) against the deployed web application. So you'll get the test result *and* the compiled JSPs.

This task can be used to precompile JSP pages for fast initial invocation of JSP pages, deployment on a server without the full JDK installed, or simply to syntax check the pages without deploying them. In most cases, a javac task is usually the next stage in the build process. The task does basic dependency checking to prevent unnecessary recompilation -this checking compares source and destination timestamps, and does not factor in class or taglib dependencies, or `<jsp:include>` references.

By default the task uses the Jasper JSP compiler. This means the task needs jasper.jar and jasper-runtime.jar, which come with builds of Tomcat 4/Catalina from the Jakarta Tomcat project, and any other Jar files which may be needed in future versions (it changes) We recommend (in March 2003) Tomcat version 4.1.x for the most robust version of Jasper.

There are many limitations with this task which partially stem from the many versions of Jasper, others from implementation 'issues' in the task (i.e. nobody's willingness to radically change large bits of it to work around jasper). Because of this and the fact that JSP pages do not have to be portable across implementations -or versions of implementations- this task is better used for validating JSP pages before deployment, rather than precompiling them. For that, just deploy and run your httpunit junit tests after deployment to compile and test your pages, all in one go.

Parameters

The Task has the following attributes:

Attribute	Description	Required
destdir	Where to place the generated files. They are located under here according to the given package name.	Yes
srcdir	Where to look for source jsp files.	Yes
verbose	The verbosity integer to pass to the compiler. Default="0"	No
package	Name of the destination package for generated java classes.	No
compiler	class name of a JSP compiler adapter, such as "jasper" or "jasper41"	No -defaults to "jasper"
ieplugin	Java Plugin classid for Internet Explorer.	No
mapped	(boolean) Generate separate write() calls for each HTML line in the JSP.	No
classpath	The classpath to use to run the jsp compiler. This can also be specified by the nested element `classpath` Path).	No, but it seems to work better when used
classpathref	A Reference. As per `classpath`	No

failonerror	flag to control action on compile failures: default=yes	No
uribase	The uri context of relative URI references in the JSP pages. If it does not exist then it is derived from the location of the file relative to the declared or derived value of `uriroot`.	No
uriroot	The root directory that uri files should be resolved against.	No
compiler	Class name of jsp compiler adapter to use. Defaults to the standard adapter for Jasper.	No
compilerclasspath	The classpath used to find the compiler adapter specified by the `compiler` attribute.	No
webinc	Output file name for the fraction of web.xml that lists servlets.	No
webxml	File name for web.xml to be generated	No

The `mapped` option will, if set to true, split the JSP text content into a one line per call format. There are comments above and below the mapped write calls to localize where in the JSP file each line of text comes from. This can lead to a minor performance degradation (but it is bound by a linear complexity). Without this options all adjacent writes are concatenated into a single write.

The `ieplugin` option is used by the `<jsp:plugin>` tags. If the Java Plug-in COM Class-ID you want to use changes then it can be specified here. This should not need to be altered.

`uriroot` specifies the root of the web application. This is where all absolute uris will be resolved from. If it is not specified then the first JSP page will be used to derive it. To derive it each parent directory of the first JSP page is searched for a `WEB-INF` directory, and the directory closest to the JSP page that has one will be used. If none can be found then the directory Jasperc was called from will be used. This only affects pages translated from an explicitly declared JSP file -including references to taglibs

`uribase` is used to establish the uri context of relative URI references in the JSP pages. If it does not exist then it is derived from the location of the file relative to the declared or derived value of `uriroot`. This only affects pages translated from an explicitly declared JSP file.

Parameters specified as nested elements

This task is a directory based task, like **javac**, so the jsp files to be compiled are located as java files are by **javac**. That is, elements such as `includes` and `excludes` can be used directly inside the task declaration.

Elements specific to the jspc task are:-

classpath

The classpath used to compile the JSP pages, specified as for any other classpath.

classpathref

a reference to an existing classpath

webapp

Instructions to jasper to build an entire web application. The base directory must have a WEB-INF subdirectory beneath it. When used, the task hands off all dependency checking to the compiler.

Attribute	Description	Required

basedir	the base directory of the web application	Yes

Example

```
<jspc srcdir="${basedir}/src/war"
      destdir="${basedir}/gensrc"
      package="com.i3sp.jsp"
      compiler="jasper41"
      verbose="9">
  <include name="**/*.jsp"/>
</jspc>
```

Build all jsp pages under src/war into the destination /gensrc, in a package hierarchy beginning with com.i3sp.jsp.

```
<jspc
      destdir="interim"
      verbose="1"
      srcdir="src"
      compiler="jasper41"
      package="com.i3sp.jsp">
  <include name="**/*.jsp"/>
</jspc>
<depend
         srcdir="interim"
         destdir="build"
         cache="build/dependencies"
         classpath="lib/taglibs.jar"/>
<javac
         srcdir="interim"
         destdir="build"
         classpath="lib/taglibs.jar"
         debug="on"/>
```

Generate jsp pages then javac them down to bytecodes. Include lib/taglib jar in the java compilation. Dependency checking is used to scrub the java files if class dependencies indicate it is needed.

Notes

Using the `package` attribute it is possible to identify the resulting java files and thus do full dependency checking - this task should only rebuild java files if their jsp file has been modified. However, this only works with some versions of jasper. By default the checking supports tomcat 4.0.x with the "jasper" compiler, set the compiler to "jasper41" for the tomcat4.1.x dependency checking. Even when it does work, changes in .TLD imports or in compile time includes do not get picked up.

Jasper generates JSP pages against the JSP1.2 specification -a copy of version 2.3 of the servlet specification is needed on the classpath to compile the Java code.

wljspc

Description

Class to precompile JSP's using weblogic's jsp compiler (weblogic.jspc)

Tested only on Weblogic 4.5.1 - NT4.0 and Solaris 5.7,5.8

Parameters

Attribute	Values	Required
src	root of source tree for JSP, ie, the document root for your weblogic server	Yes
dest	root of destination directory, what you have set as WorkingDir in the weblogic properties	Yes
package	start package name under which your JSP's would be compiled	Yes
classpath	Class path to use when compiling jsp's	Yes

A classpath should be set which contains the weblogic classes as well as all application classes referenced by the JSP. The system classpath is also appended when the jspc is called, so you may choose to put everything in the classpath while calling Ant. However, since presumably the JSP's will reference classes being build by Ant, it would be better to explicitly add the classpath in the task

The task checks timestamps on the JSP's and the generated classes, and compiles only those files that have changed.

It follows the weblogic naming convention of putting classes in **_dirName/_fileName.class for dirname/fileName.jsp**

Example

```
<target name="jspcompile" depends="compile">
  <wljspc src="c:\\weblogic\\myserver\\public_html" dest="c:\\weblogic\\myserver\\serverclasses"
package="myapp.jsp">
    <classpath>
      <pathelement location="${weblogic.classpath}"/>
      <pathelement path="${compile.dest}"/>
    </classpath>
  </wljspc>
</target>
```

Limitations

- This works only on weblogic 4.5.1
- It compiles the files thru the Classic compiler only.
- Since it is my experience that weblogic jspc throws out of memory error on being given too many files at one go, it is called multiple times with one jsp file each.

ANT ServerDeploy User Manual

by

- Christopher A. Longo (cal@cloud9.net)
- Cyrille Morvan (cmorvan@ingenosya.com)

At present the tasks support:

- Weblogic servers
- JOnAS 2.4 Open Source EJB server

Over time we expect further optional tasks to support additional J2EE Servers.

Task	Application Servers	
serverdeploy	**Nested Elements**	
	generic	Generic task
	jonas	JOnAS 2.4
	weblogic	Weblogic

ServerDeploy element

Description:

The `serverdeploy` task is used to run a "hot" deployment tool for vendor-specific J2EE server. The task requires nested elements which define the attributes of the vendor-specific deployment tool being executed. Vendor-specific deployment tools elements may enforce rules for which attributes are required, depending on the tool.

Parameters:

Attribute	Description	Required
action	This is the action to be performed. For most cases this will be "deploy". Some tools support additional actions, such as "delete", "list", "undeploy", "update"...	Yes
source	A fully qualified path/filename of the component to be deployed. This may be an .ear, .jar, .war, or any other type that is supported by the server.	Tool dependent

Nested Elements

The serverdeploy task supports a nested `classpath` element to set the classpath.

Vendor-specific nested elements

Parameters used for all tools:

Attribute	Description	Required
classpath	The classpath to be passed to the JVM running the tool. The classpath may also be supplied as a nested element.	Tool dependent
server	The address or URL for the server where the component will be deployed.	Tool dependent
username	The user with privileges to deploy applications to the server.	Tool dependent
password	The password of the user with privileges to deploy applications to the server.	Tool dependent

Also supported are nested vendor-specific elements.

Generic element

This element is provided for generic Java-based deployment tools. The generic task accepts (but does not require) nested `arg` and `jvmarg` elements. A JVM will be spawned with the provided attributes. It is recommended that a vendor-specific element be used over the generic one if at all possible.

The following attributes are supported by the generic element.

Attribute	Description	Required
classname	This is the fully qualified classname of the Java based deployment tool to execute.	Yes

Nested Elements

The generic element supports nested `<arg>` and `<jvmarg>` elements.

Example

This example shows the use of generic deploy element to deploy a component using a Java based deploy tool:

```
    <serverdeploy action="deploy" source="${lib.dir}/ejb_myApp.ear">
        <generic classname="com.yamato.j2ee.tools.deploy.DeployTool"
            classpath="${classpath}"
            username="${user.name}"
            password="${user.password}">
            <arg value="-component=WildStar"/>
            <arg value="-force"/>
            <jvmarg value="-ms64m"/>
            <jvmarg value="-mx128m"/>
        </generic>
    </serverdeploy>
```

WebLogic element

The WebLogic element contains additional attributes to run the `weblogic.deploy` deployment tool.

Valid actions for the tool are `deploy`, `undeploy`, `list`, `update`, and `delete`.

If the action is `deploy` or `update`, the `application` and `source` attributes must be set. If the action is `undeploy` or

delete, the application attribute must be set. If the username attribute is omitted, it defaults to "system". The password attribute is required for all actions.

Attribute	Description	Required
application	This is the name of the application being deployed	Yes
component	This is the component string for deployment targets. It is in the form <component>:<target1>,<target2>... Where component is the archive name (minus the .jar, .ear, .war extension). Targets are the servers where the components will be deployed	no
debug	If set to true, additional information will be printed during the deployment process.	No

Examples

This example shows the use of serverdeploy to deploy a component to a WebLogic server:

```
    <serverdeploy action="deploy" source="${lib.dir}/ejb_myApp.ear">
        <weblogic application="myapp"
            server="t3://myserver:7001"
            classpath="${weblogic.home}/lib/weblogic.jar"
            username="${user.name}"
            password="${user.password}"
            component="ejb_foobar:myserver,productionserver"
            debug="true"/>
    </serverdeploy>
```

This example shows serverdeploy being used to delete a component from a WebLogic server:

```
    <serverdeploy action="delete" source="${lib.dir}/ejb_myApp.jar"/>
        <weblogic application="myapp"
            server="t3://myserver:7001"
            classpath="${weblogic.home}/lib/weblogic.jar"
            username="${user.name}"
            password="${user.password}"/>
    </serverdeploy>
```

JOnAS (Java Open Application Server) element

The JOnAS element contains additional attributes to run the JonasAdmin deployment tool.

Valid actions for the tool are deploy, undeploy, list and update.

You can't use user and password property with this task.

Attribute	Description	Required
jonasroot	The root directory for JOnAS.	Yes
orb	Choose your ORB : RMI, JEREMIE, DAVID, ... If omitted, it defaults to the one present in classpath. The corresponding JOnAS JAR is automatically added to the classpath. If your orb is DAVID (RMI/IIOP) you must specify davidhost and davidport properties.	No
davidhost	The value for the system property : david.CosNaming.default_host .	No
davidport	The value for the system property : david.CosNaming.default_port .	No
classname	This is the fully qualified classname of the Java based deployment tool to execute. Default to org.objectweb.jonas.adm.JonasAdmin	No

Nested Elements

The jonas element supports nested <arg> and <jvmarg> elements.

Examples

This example shows the use of serverdeploy to deploy a component to a JOnAS server:

```
    <serverdeploy action="deploy" source="${lib.dir}/ejb_myApp.jar">
        <jonas server="MyJOnAS" jonasroot="${jonas.root}">

           <classpath>
               <pathelement path="${jonas.root}/lib/RMI_jonas.jar"/>
               <pathelement path="${jonas.root}/config/"/>
           </classpath>
        </jonas>
    </serverdeploy>
```

This example shows serverdeploy being used to list the components from a JOnAS server and a WebLogic server:

```
    <serverdeploy action="list"/>
        <jonas jonasroot="${jonas.root}" orb="JEREMIE"/>
        <weblogic application="myapp"
            server="t3://myserver:7001"
            classpath="${weblogic.home}/lib/weblogic.jar"
            username="${user.name}"
            password="${user.password}"/>
    </serverdeploy>
```

Javadoc/*Javadoc2*

Description

Generates code documentation using the javadoc tool.

The source directory will be recursively scanned for Java source files to process but only those matching the inclusion rules, and not matching the exclusions rules will be passed to the javadoc tool. This allows wildcards to be used to choose between package names, reducing verbosity and management costs over time. This task, however, has no notion of "changed" files, unlike the javac task. This means all packages will be processed each time this task is run. In general, however, this task is used much less frequently.

NOTE: since javadoc calls System.exit(), javadoc cannot be run inside the same VM as Ant without breaking functionality. For this reason, this task always forks the VM. This overhead is not significant since javadoc is normally a heavy application and will be called infrequently.

NOTE: the packagelist attribute allows you to specify the list of packages to document outside of the Ant file. It's a much better practice to include everything inside the `build.xml` file. This option was added in order to make it easier to migrate from regular makefiles, where you would use this option of javadoc. The packages listed in packagelist are not checked, so the task performs even if some packages are missing or broken. Use this option if you wish to convert from an existing makefile. Once things are running you should then switch to the regular notation.

DEPRECATION: *the javadoc2 task simply points to the javadoc task and it's there for back compatibility reasons. Since this task will be removed in future versions, you are strongly encouraged to use javadoc instead.*

In the table below, 1.2 means available if your current Java VM is a 1.2 VM (but not 1.3 or later), 1.4+ for any VM of at least version 1.4, otherwise any VM of at least version 1.2 is acceptable. JDKs <1.4 are no longer supported. If you specify the `executable` attribute it is up to you to ensure that this command supports the attributes you wish to use.

Note:
When generating the JavaDocs for classes which contains annotations you maybe get a `java.lang.ClassCastException: com.sun.tools.javadoc.ClassDocImpl`. This is due bug-6442982. The cause is that JavaDoc cannot find the implementations of used annotations. The workaround is providing the jars with these implementations (like JAXBs `@XmlType`, ...) to <javadoc> using `classpath`, `classpathref` attributes or nested <classpath> element.

Note: many problems with running javadoc stem from command lines that have become too long - even though the error message doesn't give the slightest hint this may be the problem. If you encounter problems with the task, try to set the `useexternalfile` attribute to `true` first.

If you use multiple ways to specify where javadoc should be looking for sources your result will be the union of all specified documentations. If you, e.g., specify a sourcepath attribute and also a nested packageset both pointing at the same directory your excludepackagenames attribute won't have any effect unless it agrees with the exclude patterns of the packageset (and vice versa).

Parameters

Attribute	Description	Availability	Required
sourcepath	Specify where to find source files	all	At least one of the three or nested
sourcepathref	Specify where to find source files by reference to a PATH defined elsewhere.	all	

			`<sourcepath>`, `<fileset>` or `<packageset>`
sourcefiles	Comma separated list of source files -- see also the nested `source` element.	all	
destdir	Destination directory for output files	all	Yes, unless a doclet has been specified.
maxmemory	Max amount of memory to allocate to the javadoc VM	all	No
packagenames	Comma separated list of package files (with terminating wildcard) -- see also the nested `package` element.	all	No
packageList	The name of a file containing the packages to process	all	No
classpath	Specify where to find user class files	all	No
Bootclasspath	Override location of class files loaded by the bootstrap class loader	all	No
classpathref	Specify where to find user class files by reference to a PATH defined elsewhere.	all	No
bootclasspathref	Override location of class files loaded by the bootstrap class loader by reference to a PATH defined elsewhere.	all	No
Extdirs	Override location of installed extensions	all	No
Overview	Read overview documentation from HTML file	all	No
access	Access mode: one of `public`, `protected`, `package`, or `private`	all	No (default `protected`)
Public	Show only public classes and members	all	No
Protected	Show protected/public classes and members (default)	all	No
Package	Show package/protected/public classes and members	all	No
Private	Show all classes and members	all	No
Old	Generate output using JDK 1.1 emulating doclet. **Note:** as of Ant 1.8.0 this attribute doesn't have any effect since the javadoc of Java 1.4 (required by Ant 1.8.0) doesn't support the -1.1 switch anymore.	1.2	No
Verbose	Output messages about what Javadoc is doing	all	No
Locale	Locale to be used, e.g. en_US or en_US_WIN	all	No
Encoding	Source file encoding name	all	No
Version	Include @version paragraphs	all	No
Use	Create class and package usage pages	all	No
Author	Include @author paragraphs	all	No
Splitindex	Split index into one file per letter	all	No
Windowtitle	Browser window title for the documentation (text)	all	No
Doctitle	Include title for the package index(first) page (html-code)	all	No
Header	Include header text for each page (html-code)	all	No
Footer	Include footer text for each page (html-code)	all	No
bottom	Include bottom text for each page (html-code)	all	No
link	Create links to javadoc output at the given URL -- see also the nested `link` element.	all	No
linkoffline	Link to docs at `<url>` using package list at `<url2>` - separate	all	No

	the URLs by using a space character -- see also the nested `link` element.		
group	Group specified packages together in overview page. The format is as described below -- see also the nested `group` element.	all	No
nodeprecated	Do not include @deprecated information	all	No
nodeprecatedlist	Do not generate deprecated list	all	No
notree	Do not generate class hierarchy	all	No
noindex	Do not generate index	all	No
nohelp	Do not generate help link	all	No
nonavbar	Do not generate navigation bar	all	No
serialwarn	Generate warning about @serial tag	all	No
helpfile	Specifies the HTML help file to use	all	No
stylesheetfile	Specifies the CSS stylesheet to use	all	No
charset	Charset for cross-platform viewing of generated documentation	all	No
docencoding	Output file encoding name	all	No
doclet	Specifies the class file that starts the doclet used in generating the documentation -- see also the nested `doclet` element.	all	No
docletpath	Specifies the path to the doclet class file that is specified with the -doclet option.	all	No
docletpathref	Specifies the path to the doclet class file that is specified with the -doclet option by reference to a PATH defined elsewhere.	all	No
additionalparam	Lets you add additional parameters to the javadoc command line. Useful for doclets. Parameters containing spaces need to be quoted using " -- see also the nested `arg` element.	all	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	all	No
excludepackagenames	comma separated list of packages you don't want docs for -- see also the nested `excludepackage` element.	all	No
defaultexcludes	indicates whether default excludes should be used (`yes` | `no`); default excludes are used when omitted.	all	No
useexternalfile	indicates whether the sourcefile name specified in srcfiles or as nested source elements should be written to a temporary file to make the command line shorter. Also applies to the package names specified via the packagenames attribute or nested package elements. *Since Ant 1.7.0*, also applies to all the other command line options. (`yes` | `no`). Default is no.	all	No
source	Necessary to enable javadoc to handle assertions present in J2SE v 1.4 source code. Set this to "1.4" to documents code that compiles using `"javac -source 1.4"`. A default value for this attribute can be provided using the magic `ant.build.javac.source` property.	1.4+	No
linksource	Generate hyperlinks to source files. *since Ant 1.6.* (`yes` | `no`). Default is no.	1.4+	No
breakiterator	Use the new breakiterator algorithm. *since Ant 1.6.* (`yes` | `no`).	1.4+	No

	Default is no.		
noqualifier	Enables the `-noqualifier` argument - must be `all` or a colon separated list of packages. *since Ant 1.6.*	1.4+	No
includenosourcepackages	If set to true, packages that don't contain Java source but a package.html will get documented as well. *since Ant 1.6.3.*	all	No (default is `false`)
executable	Specify a particular `javadoc` executable to use in place of the default binary (found in the same JDK as Ant is running in). *since Ant 1.6.3.*	all	No
docfilessubdirs	Enables deep-copying of `doc-files` subdirectories. Defaults to false. *since Ant 1.8.0.*	1.4	No
excludedocfilessubdir	Colon-separated list of `doc-files`' subdirectories to exclude if `docfilessubdirs` is true. *since Ant 1.8.0.*	1.4	No

Format of the group attribute

The arguments are comma-delimited. Each single argument is 2 space-delimited strings, where the first one is the group's title and the second one a colon delimited list of packages.

If you need to specify more than one group, or a group whose title contains a comma or a space character, using nested group elements is highly recommended.

E.g.:

```
    group="XSLT_Packages org.apache.xalan.xslt*,XPath_Packages org.apache.xalan.xpath*"
```

Parameters specified as nested elements

packageset

A DirSet. All matched directories that contain Java source files will be passed to javadoc as package names. Package names are created from the directory names by translating the directory separator into dots. Ant assumes the base directory of the packageset points to the root of a package hierarchy.

The `packagenames`, `excludepackagenames` and `defaultexcludes` attributes of the task have no effect on the nested `<packageset>` elements.

fileset

A FileSet. All matched files will be passed to javadoc as source files. Ant will automatically add the include pattern `**/*.java` (and `**/package.html` if includenosourcepackages is true) to these filesets.

Nested filesets can be used to document sources that are in the default package or if you want to exclude certain files from documentation. If you want to document all source files and don't use the default package, packagesets should be used instead as this increases javadocs performance.

The `packagenames`, `excludepackagenames` and `defaultexcludes` attributes of the task have no effect on the nested `<fileset>` elements.

sourcefiles

A container for arbitrary file system based resource collections. All files contained in any of the nested collections

(this includes nested filesets, filelists or paths) will be passed to javadoc as source files.

package

Same as one entry in the list given by `packagenames`.

Parameters

Attribute	Description	Required
name	The package name (may be a wildcard)	Yes

excludepackage

Same as one entry in the list given by `excludepackagenames`.

Parameters

Same as for `package`.

source

Same as one entry in the list given by `sourcefiles`.

Parameters

Attribute	Description	Required
file	The source file to document	Yes

doctitle

Same as the `doctitle` attribute, but you can nest text inside the element this way.

If the nested text contains line breaks, you must use the useexternalfile attribute and set it to true.

header

Similar to `<doctitle>`.

footer

Similar to `<doctitle>`.

bottom

Similar to `<doctitle>`.

link

Create link to javadoc output at the given URL. This performs the same role as the link and linkoffline attributes. You

can use either syntax (or both at once), but with the nested elements you can easily specify multiple occurrences of the arguments.

Parameters

Attribute	Description	Required
href	The URL for the external documentation you wish to link to. This can be an absolute URL, or a relative file name.	Yes
offline	True if this link is not available online at the time of generating the documentation	No
packagelistLoc	The location to the directory containing the package-list file for the external documentation	One of the two if the offline attribute is true
packagelistURL	The URL of the the directory containing the package-list file for the external documentation	
resolveLink	If the link attribute is a relative file name, Ant will first try to locate the file relative to the current project's basedir and if it finds a file there use an absolute URL for the link attribute, otherwise it will pass the file name verbatim to the javadoc command.	No, default is false.

group

Separates packages on the overview page into whatever groups you specify, one group per table. This performs the same role as the group attribute. You can use either syntax (or both at once), but with the nested elements you can easily specify multiple occurrences of the arguments.

Parameters

Attribute	Description	Required
title	Title of the group	Yes, unless nested `<title>` given
packages	List of packages to include in that group. Multiple packages are separated with ':'.	Yes, unless nested `<package>`s given

The title may be specified as a nested `<title>` element with text contents, and the packages may be listed with nested `<package>` elements as for the main task.

doclet

The doclet nested element is used to specify the doclet that javadoc will use to process the input source files. A number of the standard javadoc arguments are actually arguments of the standard doclet. If these are specified in the javadoc task's attributes, they will be passed to the doclet specified in the `<doclet>` nested element. Such attributes should only be specified, therefore, if they can be interpreted by the doclet in use.

If the doclet requires additional parameters, these can be specified with `<param>` elements within the `<doclet>` element. These parameters are restricted to simple strings. An example usage of the doclet element is shown below:

```
<javadoc ... >
   <doclet name="theDoclet"
           path="path/to/theDoclet">
      <param name="-foo" value="foovalue"/>
```

```
        <param name="-bar" value="barvalue"/>
    </doclet>
  </javadoc>
```

tag

If you want to specify a standard tag using a nested tag element because you want to determine the order the tags are output, you must not set the description attribute for those tags.

Parameters

Attribute	Description	Required
name	Name of the tag (e.g. `todo`)	Yes, unless the `dir` attribute is specified.
description	Description for tag (e.g. `To do:`)	No, the javadoc executable will pick a default if this is not specified.
enabled	Whether or not the tag is enabled (defaults to `true`)	No
scope	Scope for the tag - the elements in which it can be used. This is a comma separated list of some of the elements: `overview`, `packages`, `types`, `constructors`, `methods`, `fields` or the default, `all`.	No

dir	If this attribute is specified, this element will behave as an implicit fileset. The files included by this fileset should contain each tag definition on a separate line, as described in the Javadoc reference guide:
	`ejb.bean:t:XDoclet EJB Tag`
	`todo:a:To Do`
	Note: The Javadoc reference quide has double quotes around the description part of the definition. This will not work when used in a file, as the definition is quoted again when given to the javadoc program.
	Note: If this attribute is specified, all the other attributes in this element will be ignored.
		No

taglet

The taglet nested element is used to specify custom taglets.

Parameters

Attribute	Description	Required
name	The name of the taglet class (e.g. `com.sun.tools.doclets.ToDoTaglet`)	Yes
path	A path specifying the search path for the taglet class (e.g. `/home/taglets`). The path may also be specified by a nested `<path>` element	No

sourcepath, classpath and bootclasspath

`Javadoc`'s *sourcepath*, *classpath* and *bootclasspath* attributes are PATH like structure and can also be set via nested

sourcepath, *classpath* and *bootclasspath* elements respectively.

arg

Use nested `<arg>` to specify additional arguments. See Command line arguments. *Since Ant 1.6*

Example

```
  <javadoc packagenames="com.dummy.test.*"
           sourcepath="src"
           excludepackagenames="com.dummy.test.doc-files.*"
           defaultexcludes="yes"
           destdir="docs/api"
           author="true"
           version="true"
           use="true"
           windowtitle="Test API">
    <doctitle><![CDATA[<h1>Test</h1>]]></doctitle>
    <bottom><![CDATA[<i>Copyright &#169; 2000 Dummy Corp. All Rights Reserved.</i>]]></bottom>
    <tag name="todo" scope="all" description="To do:"/>
    <group title="Group 1 Packages" packages="com.dummy.test.a*"/>
    <group title="Group 2 Packages" packages="com.dummy.test.b*:com.dummy.test.c*"/>
    <link offline="true" href="http://java.sun.com/j2se/1.5.0/docs/api/" packagelistLoc="C:\tmp"/>
    <link href="http://developer.java.sun.com/developer/products/xml/docs/api/"/>
  </javadoc>
```

is the same as

```
  <javadoc
           destdir="docs/api"
           author="true"
           version="true"
           use="true"
           windowtitle="Test API">

    <packageset dir="src" defaultexcludes="yes">
      <include name="com/dummy/test/**"/>
      <exclude name="com/dummy/test/doc-files/**"/>
    </packageset>

    <doctitle><![CDATA[<h1>Test</h1>]]></doctitle>
    <bottom><![CDATA[<i>Copyright &#169; 2000 Dummy Corp. All Rights Reserved.</i>]]></bottom>
    <tag name="todo" scope="all" description="To do:"/>
    <group title="Group 1 Packages" packages="com.dummy.test.a*"/>
    <group title="Group 2 Packages" packages="com.dummy.test.b*:com.dummy.test.c*"/>
    <link offline="true" href="http://java.sun.com/j2se/1.5.0/docs/api/" packagelistLoc="C:\tmp"/>
    <link href="http://developer.java.sun.com/developer/products/xml/docs/api/"/>
  </javadoc>
```

or

```
  <javadoc
           destdir="docs/api"
           author="true"
           version="true"
           use="true"
           windowtitle="Test API">

    <fileset dir="src" defaultexcludes="yes">
      <include name="com/dummy/test/**"/>
      <exclude name="com/dummy/test/doc-files/**"/>
    </fileset>

    <doctitle><![CDATA[<h1>Test</h1>]]></doctitle>
    <bottom><![CDATA[<i>Copyright &#169; 2000 Dummy Corp. All Rights Reserved.</i>]]></bottom>
    <tag name="todo" scope="all" description="To do:"/>
    <group title="Group 1 Packages" packages="com.dummy.test.a*"/>
    <group title="Group 2 Packages" packages="com.dummy.test.b*:com.dummy.test.c*"/>
    <link offline="true" href="http://java.sun.com/j2se/1.5.0/docs/api/" packagelistLoc="C:\tmp"/>
    <link href="http://developer.java.sun.com/developer/products/xml/docs/api/"/>
  </javadoc>
```

AntCall

Description

Call another target within the same buildfile optionally specifying some properties (params in this context). **This task must not be used outside of a `target`.**

By default, all of the properties of the current project will be available in the new project. Alternatively, you can set the *inheritAll* attribute to `false` and only "user" properties (i.e., those passed on the command-line) will be passed to the new project. In either case, the set of properties passed to the new project will override the properties that are set in the new project (See also the property task).

You can also set properties in the new project from the old project by using nested param tags. These properties are always passed to the new project and any project created in that project regardless of the setting of *inheritAll*. This allows you to parameterize your subprojects. Properties defined on the command line can not be overridden by nested `<param>` elements.

When more than one nested `<param>` element would set a property of the same name, the one declared last will win. This is for backwards compatibility reasons even so it is different from the way `<property>` tasks in build files behave.

Nested *`<reference>`* elements can be used to copy references from the calling project to the new project, optionally under a different id. References taken from nested elements will override existing references that have been defined outside of targets in the new project - but not those defined inside of targets.

When a target is invoked by antcall, all of its dependent targets will also be called within the context of any new parameters. For example. if the target "doSomethingElse" depended on the target "init", then the *antcall* of "doSomethingElse" will call "init" during the call. Of course, any properties defined in the antcall task or inherited from the calling target will be fixed and not overridable in the init task--or indeed in the "doSomethingElse" task.

The called target(s) are run in a new project; be aware that this means properties, references, etc. set by called targets will not persist back to the calling project.

If the build file changes after you've started the build, the behavior of this task is undefined.

Parameters

Attribute	Description	Required
target	The target to execute.	Yes
inheritAll	If `true`, pass all properties to the new Ant project. Defaults to `true`.	No
inheritRefs	If `true`, pass all references to the new Ant project. Defaults to `false`.	No

Note on `inheritRefs`

`<antcall>` will not override existing references, even if you set `inheritRefs` to true. As the called build files is the same build file as the calling one, this means it will not override any reference set via an `id` attribute at all. The only references that can be inherited by the child project are those defined by nested `<reference>` elements or references defined by tasks directly (not using the `id` attribute).

Parameters specified as nested elements

param

Specifies the properties to set before running the specified target. See property for usage guidelines.
These properties become equivalent to properties you define on the command line. These are special properties and they will always get passed down, even through additional <*ant*> tasks with inheritall set to false (see above).

reference

Used to choose references that shall be copied into the new project, optionally changing their id.

Attribute	Description	Required
refid	The id of the reference in the calling project.	Yes
torefid	The id of the reference in the new project.	No, defaults to the value of refid.

propertyset

You can specify a set of properties to be copied into the new project with propertysets.

since Ant 1.6.

target

You can specify multiple targets using nested <target> elements instead of using the target attribute. These will be executed as if Ant had been invoked with a single target whose dependencies are the targets so specified, in the order specified.

Attribute	Description	Required
name	The name of the called target.	Yes

since Ant 1.6.3.

Examples

```
<target name="default">
  <antcall target="doSomethingElse">
    <param name="param1" value="value"/>
  </antcall>
</target>

<target name="doSomethingElse">
  <echo message="param1=${param1}"/>
</target>
```

Will run the target 'doSomethingElse' and echo 'param1=value'.

```
<antcall ... >
  <reference refid="path1" torefid="path2"/>
</antcall>
```

will copy the parent's definition of path1 into the new project using the id path2.

Apply/*ExecOn*

The name execon is deprecated and only kept for backwards compatibility.

Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

The files and/or directories of a number of Resource Collections – including but not restricted to FileSets, DirSets (*since Ant 1.6*) or FileLists (*since Ant 1.6*) – are passed as arguments to the system command.

If you specify a nested mapper, the timestamp of each source file is compared to the timestamp of a target file which is defined by the nested mapper element and searched for in the given *dest*, if specified.

At least one fileset or filelist is required, and you must not specify more than one mapper.

Note that you cannot interact with the forked program, the only way to send input to it is via the input and inputstring attributes.

Running Ant as a background process on Unix(-like) systems

If you run Ant as a background process (like `ant &`) and use the `<apply>` task with `spawn` set to `false`, you must provide explicit input to the forked process or Ant will be suspended because it tries to read from the standard input.

Parameters

Attribute	Description	Required
executable	the command to execute without any command line arguments.	Yes
dest	the directory where the command is expected to place target files when it is executed. This attribute is valid only when used in conjunction with a nested mapper; if omitted, the target filenames returned by the mapper will be interpreted as absolute paths.	No
spawn	whether or not you want the commands to be spawned. If you spawn a command, its output will not be logged by ant. The input, output, error, and result property settings are not active when spawning a process. *since Ant 1.6*	No, default is *false*
dir	the directory in which the command should be executed.	No
relative	whether the filenames should be passed on the command line as relative pathnames (relative to the base directory of the corresponding fileset/list for source files or the *dest* attribute for target files).	No, default is *false*
forwardslash	whether the file names should be passed with forward slashes even if the operating system requires other file separator. The option is ignored if the system file separator is a forward slash.	No, default is *false*
os	list of Operating Systems on which the command may be executed.	No
osfamily	OS family as used in the <os> condition. *since Ant 1.7*	No
output	the file to which the output of the command should be redirected. If the error stream is	No

	not also redirected to a file or property, it will appear in this output.	
error	The file to which the standard error of the command should be redirected. *since Ant 1.6*	No
logError	This attribute is used when you wish to see error output in Ant's log and you are redirecting output to a file/property. The error output will not be included in the output file/property. If you redirect error with the "error" or "errorProperty" attributes, this will have no effect. *since Ant 1.6*	No
append	whether output should be appended to or overwrite an existing file. If you set parallel to false, you will probably want to set this one to true.	No, default is *false*
outputproperty	the name of a property in which the output of the command should be stored. Unless the error stream is redirected to a separate file or stream, this property will include the error output.	No
errorproperty	The name of a property in which the standard error of the command should be stored. *since Ant 1.6*	No
input	A file from which the executed command's standard input is taken. This attribute is mutually exclusive with the inputstring attribute. *since Ant 1.6*	No
inputstring	A string which serves as the input stream for the executed command. This attribute is mutually exclusive with the input attribute. *since Ant 1.6*	No
resultproperty	the name of a property in which the return code of the command should be stored. Only of interest if failonerror=false. If you set parallel to false, only the result of the first execution will be stored.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	No
failifexecutionfails	Stop the build if we can't start the program. Defaults to true.	No
skipemptyfilesets	Don't run the command, if no source files have been found or are newer than their corresponding target files. Despite its name, this attribute applies to filelists as well.	No, default is *false*
parallel	Run the command only once, appending all files as arguments. If false, command will be executed once for every file.	No, default is *false*
type	One of *file*, *dir* or *both*. If set to *file*, only the names of plain files will be sent to the command. If set to *dir*, only the names of directories are considered. **Note:** The type attribute does not apply to nested *dirset*s - *dirset*s always implicitly assume type to be *dir*.	No, default is *file*
newenvironment	Do not propagate old environment when new environment variables are specified.	No, default is *false*
vmlauncher	Run command using the Java VM's execution facilities where available. If set to false the underlying OS's shell, either directly or through the antRun scripts, will be used. Under some operating systems, this gives access to facilities not normally available through the VM including, under Windows, being able to execute scripts, rather than their associated interpreter. If you want to specify the name of the executable as a relative path to the directory given by the dir attribute, it may become necessary to set vmlauncher to false as well.	No, default is *true*
resolveExecutable	When this attribute is true, the name of the executable if resolved firstly against the project basedir and if that does not exist, against the execution directory if specified. On Unix systems, if you only want to allow execution of commands in the user's path, set this to false. *since Ant 1.6*	No, default is *false*

maxparallel	Limit the amount of parallelism by passing at most this many sourcefiles at once. Set it to <= 0 for unlimited. *Since Ant 1.6.*	No, unlimited by default
addsourcefile	Whether source file names should be added to the command automatically. *Since Ant 1.6.*	No, default is *true*
verbose	Whether to print a summary after execution or not. *Since Ant 1.6.*	No, default *false*
ignoremissing	Whether to ignore nonexistent files specified via filelists. *Since Ant 1.6.2.*	No, default is *true*
force	Whether to bypass timestamp comparisons for target files. *Since Ant 1.6.3.*	No, default is *false*

Parameters specified as nested elements

fileset

You can use any number of nested `<fileset>` elements to define the files for this task and refer to `<fileset>`s defined elsewhere.

filelist

Since Ant 1.6

You can use any number of nested `<filelist>` elements to define the files for this task and refer to `<filelist>`s defined elsewhere.

dirset

Since Ant 1.6

You can use any number of nested `<dirset>` elements to define the directories for this task and refer to `<dirset>`s defined elsewhere.

Any other Resource Collection

since Ant 1.7

You can use any number of nested resource collections.

mapper

A single `<mapper>` specifies the target files relative to the `dest` attribute for dependency checking. If the `dest` attribute is specified it will be used as a base directory for resolving relative pathnames returned by the mapper. At least one `<fileset>` or `<filelist>` is required.

arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments.

srcfile

By default the file names of the source files will be added to the end of the command line (unless you set addsourcefile to `false`). If you need to place it somewhere different, use a nested `<srcfile>` element between your `<arg>` elements to mark the insertion point.

Attribute	Description	Required
prefix	a prefix to place in front of the file name when building the command line argument. *Since Ant 1.8.0*	No.
suffix	a suffix to append to the file name when building the command line argument. *Since Ant 1.8.0*	No.

targetfile

`<targetfile>` is similar to `<srcfile>` and marks the position of the target filename on the command line. If omitted, the target filenames will not be added to the command line at all. This element can only be specified if you also define a nested mapper.

Attribute	Description	Required
prefix	a prefix to place in front of the file name when building the command line argument. *Since Ant 1.8.0*	No.
suffix	a suffix to append to the file name when building the command line argument. *Since Ant 1.8.0*	No.

env

It is possible to specify environment variables to pass to the system command via nested `<env>` elements. See the description in the section about exec

redirector

Since Ant 1.6.2

A nested I/O Redirector can be specified. <apply>'s behavior is like that of exec with regard to redirectors, with the exception that, in non-*parallel* mode, file mapping will take place with each iteration. This grants the user the capacity to receive input from, and send output to, different files for each sourcefile.

Examples

```
<apply executable="ls">
  <arg value="-l"/>
  <fileset dir="/tmp">
    <patternset>
      <exclude name="**/*.txt"/>
    </patternset>
  </fileset>
  <fileset refid="other.files"/>
</apply>
```

invokes `ls -l`, adding the absolute filenames of all files below `/tmp` not ending in `.txt` and all files of the FileSet with `id other.files` to the command line.

```
<apply executable="somecommand" parallel="false">
  <arg value="arg1"/>
  <srcfile/>
  <arg value="arg2"/>
  <fileset dir="/tmp"/>
</apply>
```

invokes `somecommand arg1 SOURCEFILENAME arg2` for each file in `/tmp` replacing SOURCEFILENAME with the absolute filename of each file in turn. If `parallel` had been set to true, SOURCEFILENAME would be replaced with the absolute filenames of all files separated by spaces.

```
<apply executable="cc" dest="src/C" parallel="false">
  <arg value="-c"/>
  <arg value="-o"/>
  <targetfile/>
  <srcfile/>
  <fileset dir="src/C" includes="*.c"/>
  <mapper type="glob" from="*.c" to="*.o"/>
</apply>
```

invokes `cc -c -o TARGETFILE SOURCEFILE` for each `.c` file that is newer than the corresponding `.o`, replacing TARGETFILE with the absolute filename of the `.o` and SOURCEFILE with the absolute name of the `.c` file.

```
<mapper id="out" type="glob"
           from="src${file.separator}*.file"
           to="dest${file.separator}*.out"/>

<apply executable="processfile" dest="dest">
  <fileset dir="src" includes="*.file"/>
  <mapper refid="out"/>
  <redirector>
    <outputmapper refid="out"/>
  </redirector>
</apply>
```

Applies the fictitious "processfile" executable to all files matching `*.file` in the `src` directory. The `out` `<mapper>` has been set up to map `*.file` to `*.out`, then this `<mapper>` is used to specify `targetfiles` for this <apply> task. A reference to `out` is then used as an `<outputmapper>` nested in a `<redirector>`, which in turn is nested beneath this `<apply>` instance. This allows us to perform dependency checking against output files--the target files in this case.

```
<apply executable="ls" parallel="true"
       force="true" dest="${basedir}" append="true" type="both">
  <path>
    <pathelement path="${env.PATH}"/>
  </path>
  <identitymapper/>
</apply>
```

Applies the "ls" executable to all directories in the PATH, effectively listing all executables that are available on the PATH.

```
<apply executable="jsmin" addsourcefile="false">
    <!-- Collect the JS-files -->
    <fileset dir="src" includes="*.js"/>
    <redirector>
        <!-- redirect STDIN; fileset collects relative to its dir, but we need -->
        <!-- relative to basedir -->
        <inputmapper type="glob" from="*" to="src/*"/>
        <!-- redirect STDOUT to file in dest-dir -->
        <outputmapper id="out" type="glob" from="*.js" to="dest/*.js"/>
    </redirector>
</apply>
```

Conversion of the command `jsmin < src/a.js > dest/a.js` but for all files in the src-directory. Because the filename itself should not be passed to the `jsmin` program, the `addsourcefile` is set to `false`.

DependSet

A task to manage arbitrary dependencies between resources.

Description

The dependset task compares a set of sources with a set of target files. If any of the sources has been modified more recently than any of the target files, all of the target files are removed.

Sources and target files are specified via nested Resource Collections; sources can be resources of any type, while targets are restricted to files only. At least one set of sources and one set of targets is required.

Use a FileSet when you want to use wildcard include or exclude patterns and don't care about missing files. Use a FileList when you want to consider the non-existence of a file as if it were out of date. If there are any non-existing files in any source or target FileList, all target files will be removed.

DependSet is useful to capture dependencies that are not or cannot be determined algorithmically. For example, the `<style>` task only compares the source XML file and XSLT stylesheet against the target file to determined whether to restyle the source. Using dependset you can extend this dependency checking to include a DTD or XSD file as well as other stylesheets imported by the main stylesheet.

Parameters

Attribute	Description	Required
verbose	Makes the task list all deleted targets files and the reason why they get deleted.	No

Parameters Specified as Nested Elements

sources

The `<sources>` element is a Union into which arbitrary resource collections can be nested. **Since Ant 1.7**

srcfileset

The nested `<srcfileset>` element specifies a FileSet. All files included in this fileset will be compared against all files included in all of the `<targetfileset>` filesets and `<targetfilelist>` filelists. Multiple `<srcfileset>` filesets may be specified.

srcfilelist

The nested `<srcfilelist>` element specifies a FileList. All files included in this filelist will be compared against all files included in all of the `<targetfileset>` filesets and `<targetfilelist>` filelists. Multiple `<srcfilelist>` filelists may be specified.

targets

The `<targets>` element is a Path and thus can include any filesystem-based resource. **Since Ant 1.7**

targetfileset

The nested `<targetfileset>` element specifies a FileSet. All files included in this fileset will be compared against all files included in all of the `<srcfileset>` filesets and `<sourcefilelist>` filelists, and if any are older, they are all deleted. Multiple `<targetfileset>` filesets may be specified.

targetfilelist

The nested `<targetfilelist>` element specifies a FileList. All files included in this filelist will be compared against all files included in all of the `<srcfileset>` filesets and `<sourcefilelist>` filelists, and if any are older, they are all deleted. Multiple `<targetfilelist>` filelists may be specified.

Examples

```
    <dependset>
       <srcfilelist
           dir   = "${dtd.dir}"
           files = "paper.dtd,common.dtd"/>
       <srcfilelist
           dir   = "${xsl.dir}"
           files = "common.xsl"/>
       <srcfilelist
           dir   = "${basedir}"
           files = "build.xml"/>
       <targetfileset
           dir      = "${output.dir}"
           includes = "**/*.html"/>
    </dependset>
```

In this example derived HTML files in the ${output.dir} directory will be removed if any are out-of-date with respect to:

1. the DTD of their source XML files
2. a common DTD (imported by the main DTD)
3. a subordinate XSLT stylesheet (imported by the main stylesheet), or
4. the buildfile

If any of the sources in the above example does not exist, all target files will also be removed. To ignore missing sources instead, use filesets instead of filelists for the sources.

Java

Description

Executes a Java class within the running (Ant) VM or forks another VM if specified.

If odd things go wrong when you run this task, set fork="true" to use a new JVM.

As of Ant 1.6.3, you can interact with a forked VM, as well as sending input to it via the `input` and `inputstring` attributes.

Running Ant as a background process on Unix(-like) systems

If you run Ant as a background process (like `ant &`) and use the `<java>` task with `spawn` set to `false` and `fork` to `true`, you must provide explicit input to the forked process or Ant will be suspended because it tries to read from the standard input.

Parameters

Attribute	Description	Required
classname	the Java class to execute.	Either `jar` or `classname`
jar	the location of the jar file to execute (must have a Main-Class entry in the manifest). Fork must be set to true if this option is selected. See notes below for more details.	Either `jar` or `classname`
args	the arguments for the class that is executed. **deprecated, use nested `<arg>` elements instead.**	No
classpath	the classpath to use.	No
classpathref	the classpath to use, given as reference to a PATH defined elsewhere.	No
fork	if enabled triggers the class execution in another VM (disabled by default)	No
spawn	if enabled allows to start a process which will outlive ant. Requires fork=true, and not compatible with timeout, input, output, error, result attributes. (disabled by default)	No
jvm	the command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by java.lang.Runtime.exec(). Ignored if fork is disabled.	No
jvmargs	the arguments to pass to the forked VM (ignored if fork is disabled). **deprecated, use nested `<jvmarg>` elements instead.**	No
maxmemory	Max amount of memory to allocate to the forked VM (ignored if fork is disabled)	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0. Default is "false" (see note)	No
resultproperty	The name of a property in which the return code of the command should be stored. Only of interest if failonerror=false and if fork=true.	No
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No
output	Name of a file to which to write the output. If the error stream is not also	No

	redirected to a file or property, it will appear in this output.	
error	The file to which the standard error of the command should be redirected.	No
logError	This attribute is used when you wish to see error output in Ant's log and you are redirecting output to a file/property. The error output will not be included in the output file/property. If you redirect error with the "error" or "errorProperty" attributes, this will have no effect.	No
append	Whether output and error files should be appended to or overwritten. Defaults to false.	No
outputproperty	The name of a property in which the output of the command should be stored. Unless the error stream is redirected to a separate file or stream, this property will include the error output.	No
errorproperty	The name of a property in which the standard error of the command should be stored.	No
input	A file from which the executed command's standard input is taken. This attribute is mutually exclusive with the inputstring attribute	No; default is to take standard input from console (unless `spawn="true"`)
inputstring	A string which serves as the input stream for the executed command. This attribute is mutually exclusive with the input attribute.	No; default is to take standard input from console (unless `spawn="true"`)
newenvironment	Do not propagate old environment when new environment variables are specified. Default is "false" (ignored if fork is disabled).	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds). **It is highly recommended to use this feature only if fork is enabled.**	No
clonevm	If set to true, then all system properties and the bootclasspath of the forked Java Virtual Machine will be the same as those of the Java VM running Ant. Default is "false" (ignored if fork is disabled). *since Ant 1.7*	No

Parameters specified as nested elements

arg and jvmarg

Use nested `<arg>` and `<jvmarg>` elements to specify arguments for the Java class and the forked VM respectively. See Command line arguments.

sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the class. These properties will be made available to the VM during the execution of the class (either ANT's VM or the forked VM). The attributes for this element are the same as for environment variables.

syspropertyset

You can specify a set of properties to be used as system properties with syspropertysets.

since Ant 1.6.

classpath

`Java`'s *classpath* attribute is a PATH like structure and can also be set via a nested *classpath* element.

bootclasspath

The location of bootstrap class files can be specified using this PATH like structure - will be ignored if *fork* is not `true` or the target VM doesn't support it (i.e. Java 1.1).

since Ant 1.6.

env

It is possible to specify environment variables to pass to the forked VM via nested *env* elements. See the description in the section about exec

Settings will be ignored if fork is disabled.

permissions

Security permissions can be revoked and granted during the execution of the class via a nested *permissions* element. For more information please see permissions

When the permission RuntimePermission exitVM has not been granted (or has been revoked) the System.exit() call will be intercepted and treated like indicated in *failonerror*.

Note:
If you do not specify permissions, a set of default permissions will be added to your Java invocation to make sure that the ant run will continue or terminated as indicated by *failonerror*. All permissions not granted per default will be checked by whatever security manager was already in place. exitVM will be disallowed.

Settings will be ignored if fork is enabled.

since Ant 1.6.

assertions

You can control enablement of Java 1.4 assertions with an `<assertions>` subelement.

Assertion statements are currently ignored in non-forked mode.

since Ant 1.6.

redirector

Since Ant 1.6.2

A nested I/O Redirector can be specified. In general, the attributes of the redirector behave as the corresponding attributes available at the task level. The most notable peculiarity stems from the retention of the `<java>` attributes for backwards compatibility. Any file mapping is done using a `null` sourcefile; therefore not all Mapper types will return results. When no results are returned, redirection specifications will fall back to the task level attributes. In practice this means that defaults can be specified for input, output, and error output files.

Errors and return codes

By default the return code of a `<java>` is ignored. Alternatively, you can set `resultproperty` to the name of a property and have it assigned to the result code (barring immutability, of course). When you set `failonerror="true"`, the only possible value for `resultproperty` is 0. Any non-zero response is treated as an error and would mean the build exits.

Similarly, if `failonerror="false"` and `fork="false"` , then `<java>` **must** return 0 otherwise the build will exit, as the class was run by the build JVM.

JAR file execution

The parameter of the `jar` attribute is of type `File`; that is, the parameter is resolved to an absolute file relative to the base directory of the project, *not* the directory in which the Java task is run. If you need to locate a JAR file relative to the directory the task will be run in, you need to explicitly create the full path to the JAR file.

When using the `jar` attribute, all classpath settings are ignored according to Sun's specification.

Examples

```
<java classname="test.Main">
  <arg value="-h"/>
  <classpath>
    <pathelement location="dist/test.jar"/>
    <pathelement path="${java.class.path}"/>
  </classpath>
</java>
```

Run a class in this JVM with a new jar on the classpath

```
<java jar="dist/test.jar"
    fork="true"
    failonerror="true"
    maxmemory="128m"
    >
  <arg value="-h"/>
  <classpath>
    <pathelement location="dist/test.jar"/>
    <pathelement path="${java.class.path}"/>
  </classpath>
</java>
```

Run the JAR test.jar in this project's dist/lib directory. using the manifest supplied entry point, forking (as required), and with a maximum memory of 128MB. Any non zero return code breaks the build.

```
<java
    dir="${exec.dir}"
    jar="${exec.dir}/dist/test.jar"
    fork="true"
    failonerror="true"
    maxmemory="128m"
    >
  <arg value="-h"/>
  <classpath>
    <pathelement location="dist/test.jar"/>
    <pathelement path="${java.class.path}"/>
  </classpath>
</java>
```

Run the JAR dist/test.jar relative to the directory `${exec.dir}`, this being the same directory into which the JVM is to start up.

```
<java classname="test.Main"/>
```

Runs a given class with the current classpath.

```
  <java classname="test.Main"
        fork="yes" >
    <sysproperty key="DEBUG" value="true"/>
    <arg value="-h"/>
    <jvmarg value="-Xrunhprof:cpu=samples,file=log.txt,depth=3"/>
  </java>
```

Add system properties and JVM-properties to the JVM as in `java ="-Xrunhprof:cpu=samples,file=log.txt,depth=3 -DDEBUG=true test.Main`

```
  <java classname="ShowJavaVersion" classpath="."
        jvm="path-to-java14-home/bin/java" fork="true"
        taskname="java1.4" >
```

Use a given Java implementation (another the one Ant is currently using) to run the class. For documentation in the log `taskname` is used to change the `[java]` log-prefix to `[java1.4]`.

Note: you can not specify the (highly deprecated) MSJVM, "jview.exe" as the JVM, as it takes different parameters for other JVMs, That JVM can be started from `<exec>` if required.

Parallel

Description

Executes nested tasks in parallel with no guarantees of thread safety. Every task will run in its own thread, with the likelihood of concurrency problems scaling with the number of CPUs on the host system.

Warning: While the Ant core is believed to be thread safe, no such guarantees are made about tasks, which are not tested for thread safety during Ant's test process. Third party tasks may or may not be thread safe, and some of Ant's core tasks, such as `<javac>` are definitely not re-entrant. This is because they use libraries that were never designed to be used in a multithreaded environment.

The primary use case for `<parallel>` is to run external programs such as an application server, and the JUnit or TestNG test suites at the same time. Anyone trying to run large Ant task sequences in parallel, such as javadoc and javac at the same time, is implicitly taking on the task of identifying and fixing all concurrency bugs the tasks that they run.

Accordingly, while this task has uses, it should be considered an advanced task which should be used in certain batch-processing or testing situations, rather than an easy trick to speed up build times on a multiway CPU.

Parameters

Attribute	Description	Required
threadCount	Maximum numbers of thread to use.	No
threadsPerProcessor	Maximum number of threads to use per available processor (Java 1.4+)	No, defers to threadCount
timeout	Number of milliseconds before execution is terminated	No
failonany	If any of the nested tasks fails, execution of the task completes at that point without waiting for any other tasks to complete.	No
pollInterval	Currently has no effect	No, default is 1000

Parallel tasks have a number of uses in an Ant build file including:

- Taking advantage of available processing resources to execute external programs simultaneously.
- Testing servers, where the server can be run in one thread and the test harness is run in another thread.

Any valid Ant task may be embedded within a parallel task, including other parallel tasks, though there is no guarantee that the tasks will be thread safe in such an environment.

While the tasks within the parallel task are being run, the main thread will be blocked waiting for all the child threads to complete. If execution is terminated by a timeout or a nested task failure when the `failonany` flag is set, the parallel task will complete without waiting for other nested tasks to complete in other threads.

If any of the tasks within the `<parallel>` task fails and failonany is not set, the remaining tasks in other threads will continue to run until all threads have completed. In this situation, the parallel task will also fail.

The parallel task may be combined with the sequential task to define sequences of tasks to be executed on each thread within the parallel block

The `threadCount` attribute can be used to place a maximum number of available threads for the execution. When not present all child tasks will be executed at once. When present then the maximum number of concurrently executing tasks will not exceed the number of threads specified. Furthermore, each task will be started in the order they are given. But no guarantee is made as to the speed of execution or the order of completion of the tasks, only that each will be started before the next.

If you are using Java 1.4 or later you can also use the `threadsPerProcessor` and the number of available threads will be the stated multiple of the number of processors (there is no affinity to a particular processor however). This will override the value in `threadCount`. If `threadsPerProcessor` is specified on any older JVM, then the value in `threadCount` will be used as is.

When using `threadCount` and `threadsPerProcessor` care should be taken to ensure that the build does not deadlock. This can be caused by tasks such as `waitfor` taking up all available threads before the tasks that would unlock the `waitfor` would occur. This is not a replacement for Java Language level thread semantics and is best used for "embarassingly parallel" tasks.

Parameters specified as nested elements

daemons

The parallel task supports a `<daemons>` nested element. This is a list of tasks which are to be run in parallel daemon threads. The parallel task will not wait for these tasks to complete. Being daemon threads, however, they will not prevent Ant from completing, whereupon the threads are terminated. Failures in daemon threads which occur before the parallel task itself finishes will be reported and can cause parallel to throw an exception. Failures which occur after parallel has completed are not reported.

Daemon tasks can be used, for example, to start test servers which might not be easily terminated from Ant. By using `<daemons>` such servers do not halt the build.

Examples

```
<parallel>
  <wlrun ... >
  <sequential>
    <sleep seconds="30"/>
    <junit fork="true" forkmode="once" ... >
    <wlstop/>
  </sequential>
</parallel>
```

This example represents a typical pattern for testing a server application. In one thread the server is started (the `<wlrun>` task). The other thread consists of a three tasks which are performed in sequence. The `<sleep>` task is used to give the server time to come up. Another task which is capable of validating that the server is available could be used in place of the `<sleep>` task. The `<junit>` test harness then runs, again in its own JVM. Once the tests are complete, the server is stopped (using `<wlstop>` in this example), allowing both threads to complete. The `<parallel>` task will also complete at this time and the build will then continue.

```
<parallel>
  <javac fork="true"...> <!-- compiler servlet code -->
  <wljspc ...> <!-- precompile JSPs -->
</parallel>
```

This example shows two independent tasks being run to achieve better resource utilization during the build. In this instance, some servlets are being compiled in one thead and a set of JSPs is being precompiled in another. Developers need to be careful that the two tasks are independent, both in terms of their dependencies and in terms of their potential interactions in Ant's external environment. Here we set `fork="true"` for the `<javac>` task, so that it runs in a new process; if the `<wljspc>` task used the javac compiler in-VM (it may), concurrency problems may arise.

```
 <macrodef name="dbpurge">
    <attribute file="file"/>
   <sequential>
      <java jar="utils/dbpurge.jar" fork="true" >
        <arg file="@{file} />
      </java>
   </sequential>
</macrodef>

<parallel threadCount='4'>
  <dbpurge file="db/one" />
  <dbpurge file="db/two" />
  <dbpurge file="db/three" />
  <dbpurge file="db/four" />
  <dbpurge file="db/five" />
  <dbpurge file="db/six" />
  <dbpurge file="db/seven" />
  <dbpurge file="db/eight" />
  <!-- repeated about 40 times -->
</parallel>
```

This example represents a typical need for use of the threadCount and threadsPerProcessor attributes. Spinning up all 40 of those tasks could cripple the system for memory and CPU time. By limiting the number of concurrent executions you can reduce contention for CPU, memory and disk IO, and so actually finish faster. This is also a good candidiate for use of threadCount (and possibly threadsPerProcessor) because each task is independent (every new JVM is forked) and has no dependencies on the other tasks.

Sleep

Description

A task for sleeping a short period of time, useful when a build or deployment process requires an interval between tasks.

Parameters

Attribute	Description	Required
hours	hours to to add to the sleep time	No
minutes	minutes to add to the sleep time	No
seconds	seconds to add to the sleep time	No
milliseconds	milliseconds to add to the sleep time	No
failonerror	flag controlling whether to break the build on an error.	No

The sleep time is the sum of specified values, hours, minutes seconds and milliseconds. A negative value can be supplied to any of them provided the total sleep time is positive

Note that sleep times are always hints to be interpred by the OS how it feels - small times may either be ignored or rounded up to a minimum timeslice. Note also that the system clocks often have a fairly low granularity too, which complicates measuring how long a sleep actually took.

Examples

```
<sleep milliseconds="10"/>
```

Sleep for about 10 mS.

```
<sleep seconds="2"/>
```

Sleep for about 2 seconds.

```
<sleep hours="1" minutes="-59" seconds="-58"/>
```

Sleep for one hour less 59:58, or two seconds again

```
<sleep/>
```

Sleep for no time at all. This may yield the CPU time to another thread or process.

Subant Task

Calls a given target for all defined sub-builds.

Description

Calls a given target for all defined sub-builds. This is an extension of ant for bulk project execution.
This task must not be used outside of a `target` if it invokes the same build file it is part of.

Since Ant 1.6

Use with directories

subant can be used with directory sets to execute a build from different directories. 2 different options are offered :

- to run the same build file `/somepath/otherpath/mybuild.xml` with different base directories, use the genericantfile attribute
- if you want to run `directory1/mybuild.xml`, `directory2/mybuild.xml`, `....`, use the antfile attribute. The subant task does not set the base directory for you in this case, because you can specify it in each build file.

Parameters

Attribute	Description	Type	Requirement
antfile	Build file name, to use in conjunction with directories. Defaults to "build.xml". If `genericantfile` is set, this attribute is ignored.	String	Optional
buildpath	Set the buildpath to be used to find sub-projects.	Path	
buildpathref	Buildpath to use, by reference.	Reference	
failonerror	Sets whether to fail with a build exception on error, or go on.	boolean	
genericantfile	Build file path, to use in conjunction with directories. Use `genericantfile`, in order to run the same build file with different basedirs. If this attribute is set, `antfile` is ignored.	File	
inheritall	Corresponds to `<ant>`'s `inheritall` attribute but defaults to false in this task..	boolean	
inheritrefs	Corresponds to `<ant>`'s `inheritrefs` attribute.	boolean	
output	Corresponds to `<ant>`'s `output` attribute.	String	
target		String	
verbose	Enable/ disable log messages showing when each sub-build path is entered/ exited. The default value is false.	boolean	

Parameters as nested elements

any filesystem based resource collection

This includes <fileset>, <dirset> and <filelist> which are the nested resource collections supported prior to Ant 1.7.

dirset (org.apache.tools.ant.types.DirSet)

Adds a directory set to the implicit build path.

Note that the directories will be added to the build path in no particular order, so if order is significant, one should use a file list instead!

filelist (org.apache.tools.ant.types.FileList)

Adds an ordered file list to the implicit build path.

Note that contrary to file and directory sets, file lists can reference non-existent files or directories!

fileset (org.apache.tools.ant.types.FileSet)

Adds a file set to the implicit build path.

Note that the directories will be added to the build path in no particular order, so if order is significant, one should use a file list instead!

property (org.apache.tools.ant.taskdefs.Property)

Corresponds to <ant>'s nested <property> element.

When more than one nested <property> element would set a property of the same name, the one declared last will win. This is for backwards compatibility reasons even so it is different from the way <property> tasks in build files behave.

propertyset (org.apache.tools.ant.types.PropertySet)

Corresponds to <ant>'s nested <propertyset> element.

buildpath (org.apache.tools.ant.types.Path)

Creates a nested build path, and add it to the implicit build path.

buildpathelement (org.apache.tools.ant.types.Path.PathElement)

Creates a nested <buildpathelement>, and add it to the implicit build path.

target (org.apache.tools.ant.taskdefs.Ant.TargetElement)

You can specify multiple targets using nested <target> elements instead of using the target attribute. These will be executed as if Ant had been invoked with a single target whose dependencies are the targets so specified, in the order specified.

Attribute	Description	Required

name	The name of the called target.	Yes

since Ant 1.7.

Examples

```
<project name="subant" default="subant1">
    <property name="build.dir" value="subant.build"/>
    <target name="subant1">
        <subant target="">
            <property name="build.dir" value="subant1.build"/>
            <property name="not.overloaded" value="not.overloaded"/>
            <fileset dir="." includes="*/build.xml"/>
        </subant>
    </target>
</project>
```

this snippet build file will run ant in each subdirectory of the project directory, where a file called build.xml can be found. The property build.dir will have the value subant1.build in the ant projects called by subant.

```
  <subant target="">
      <propertyset>
          <propertyref prefix="toplevel"/>
          <mapper type="glob" from="foo*" to="bar*"/>
      </propertyset>
      <fileset dir="." includes="*/build.xml"/>
  </subant>
```

this snippet build file will run ant in each subdirectory of the project directory, where a file called build.xml can be found. All properties whose name starts with "foo" are passed, their names are changed to start with "bar" instead

```
  <subant target="compile" genericantfile="/opt/project/build1.xml">
      <dirset dir="." includes="projects*"/>
  </subant>
```

assuming the subdirs of the project dir are called projects1, projects2, projects3 this snippet will execute the compile target of /opt/project/build1.xml, setting the basedir to projects1, projects2, projects3

Now a little more complex - but useful - scenario. Assume that we have a directory structure like this:

```
root
   |  common.xml
   |  build.xml
   |
   +-- modules
         +-- modA
         |     +-- src
         +-- modB
               +-- src
```

common.xml:

```
<project>
    <property name="src.dir"       value="src"/>
    <property name="build.dir"     value="build"/>
    <property name="classes.dir"   value="${build.dir}/classes"/>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <!-- more targets -->
```

```
</project>
```

build.xml:

```
<project>

    <macrodef name="iterate">
        <attribute name="target"/>
        <sequential>
            <subant target="@{target}">
                <fileset dir="modules" includes="*/build.xml"/>
            </subant>
        </sequential>
    </macrodef>


    <target name="compile">
        <iterate target="compile"/>
    </target>

    <!-- more targets -->
</project>
```

modules/modA/build.xml:

```
<project name="modA">
    <import file="../../common.xml"/>
</project>
```

This results in very small buildfiles in the modules, maintainable buildfile (common.xml) and a clear project structure. Additionally the root buildfile is capable to run the whole build over all modules.

```
<subant failonerror="false">
    <fileset dir="." includes="**/build.xml" excludes="build.xml"/>
    <target name="clean"/>
    <target name="build"/>
</subant>
```

Does a "clean build" for each subproject.

Hint: because buildfiles are plain xml, you could generate the masterbuildfile from the common buildfile by using a XSLT transformation:

```
<xslt in="common.xml"
      out="master.xml"
      style="${ant.home}/etc/common2master.xsl"
/>
```

Waitfor

Description

Blocks execution until a set of specified conditions become true. This is intended to be used with the parallel task to synchronize a set of processes.

The conditions to wait for are defined in nested elements, if multiple conditions are specified, then the task will wait until all conditions are true..

If both maxwait and maxwaitunit are not specified, the maxwait is 3 minutes (180000 milliseconds).

If the `timeoutproperty` attribute has been set, a property of that name will be created if the condition didn't come true within the specified time.

Parameters

Attribute	Description	Required
maxwait	The maximum amount of time to wait for all the required conditions to become true before failing the task. Defaults to 180000 maxwaitunits.	No
maxwaitunit	The unit of time that must be used to interpret the value of the maxwait attribute. Defaults to millisecond. Valid Values are • millisecond • second • minute • hour • day • week	No
checkevery	The amount of time to wait between each test of the conditions. Defaults to 500 checkeveryunits.	No
checkeveryunit	The unit of time that must be used to interpret the value of the checkevery attribute. Defaults to millisecond. Valid Values are • millisecond • second • minute • hour • day • week	No
timeoutproperty	the name of the property to set if maxwait has been exceeded.	No

Nested Elements

The available conditions that satisfy the `<waitfor>` task are the same as those for the `<condition>` task. See here for the full list.

Examples

```
<waitfor maxwait="30" maxwaitunit="second">
        <available file="errors.log"/>
</waitfor>
```

waits up to 30 seconds for a file called errors.log to appear.

```
<waitfor maxwait="3" maxwaitunit="minute" checkevery="500">
        <http url="http://localhost/myapp/index.html"/>
</waitfor>
```

waits up to 3 minutes (and checks every 500 milliseconds) for a web server on localhost to serve up the specified URL.

```
<waitfor maxwait="10" maxwaitunit="second">
        <and>
            <socket server="dbserver" port="1521"/>
            <http url="http://webserver/mypage.html"/>
        </and>
</waitfor>
```

waits up to 10 seconds for a server on the dbserver machine to begin listening on port 1521 and for the http://webserver/mypage.html web page to become available.

Attrib

Since Ant 1.6.

Description

Changes the attributes of a file or all files inside specified directories. Right now it has effect only under Windows. Each of the 4 possible permissions has its own attribute, matching the arguments for the attrib command.

FileSets, DirSets or FileLists can be specified using nested `<fileset>`, `<dirset>` and `<filelist>` elements.

Starting with Ant 1.7, this task supports arbitrary Resource Collections as nested elements.

By default this task will use a single invocation of the underlying attrib command. If you are working on a large number of files this may result in a command line that is too long for your operating system. If you encounter such problems, you should set the maxparallel attribute of this task to a non-zero value. The number to use highly depends on the length of your file names (the depth of your directory tree), so you'll have to experiment a little.

By default this task won't do anything unless it detects it is running on a Windows system. If you know for sure that you have a "attrib" executable on your PATH that is command line compatible with the Windows command, you can use the task's os attribute and set its value to your current os.

Parameters

Attribute	Description	Required
file	the file or directory of which the permissions must be changed.	Yes or nested `<fileset/list>` elements.
readonly	the readonly permission.	at least one of the four.
archive	the archive permission.	
system	the system permission.	
hidden	the hidden permission.	
type	One of *file*, *dir* or *both*. If set to *file*, only the permissions of plain files are going to be changed. If set to *dir*, only the directories are considered. **Note:** The type attribute does not apply to nested *dirset*s - *dirset*s always implicitly assume type to be *dir*.	No, default is *file*
verbose	Whether to print a summary after execution or not. Defaults to `false`.	No
parallel	process all specified files using a single `chmod` command. Defaults to true.	No
maxparallel	Limit the amount of parallelism by passing at most this many sourcefiles at once. Set it to <= 0 for unlimited. Defaults to unlimited. *Since Ant 1.6.*	No
os	list of Operating Systems on which the command may be executed.	No
osfamily	OS family as used in the <os> condition.	No - defaults to "windows"

Examples

```
<attrib file="${dist}/run.bat" readonly="true" hidden="true"/>
```

makes the "run.bat" file read-only and hidden.

```
<attrib readonly="false">
  <fileset dir="${meta.inf}" includes="**/*.xml"/>
</attrib>
```

makes all ".xml" files below `${meta.inf}` readable.

```
<attrib readonly="true" archive="true">
  <fileset dir="shared/sources1">
    <exclude name="**/trial/**"/>
  </fileset>
  <fileset refid="other.shared.sources"/>
</attrib>
```

makes all files below `shared/sources1` (except those below any directory named trial) read-only and archived. In addition all files belonging to a FileSet with `id` `other.shared.sources` get the same attributes.

Checksum

Description

Generates checksum for files. This task can also be used to perform checksum verifications.

Note that many popular message digest functions - including MD5 and SHA-1 - have been broken recently. If you are going to use the task to create checksums used in an environment where security is important, please take some time to investigate the algorithms offered by your JCE provider. Note also that some JCE providers like the one by The Legion of the Bouncy Castle, the GNU project or the Technical University Graz offer more digest algorithms than those built-in into your JDK.

Warning: the case of the extension is that of the algorithm used. If you ask for "SHA1", you get a .SHA1 extension; if you ask for "sha1", you get a file ending in .sha1. The Java Crypto Engines are case-insensitive in matching algorithms, so choose a name to match your desired output extension, or set the `fileext` attribute.

Parameters

Attribute	Description	Required
file	The file to generate checksum for.	One of either *file* or at least one nested (filesystem-only) resource collection.
todir	The root directory where checksums should be written.	No. If not specified, checksum files will be written to the same directory as the files themselves. *since Ant 1.6*
algorithm	Specifies the algorithm to be used to compute the checksum. Defaults to "MD5". Other popular algorithms like "SHA" or "SHA-512" may be used as well.	No
provider	Specifies the provider of the algorithm.	No
fileext	The generated checksum file's name will be the original filename with the fileext added to it. Defaults to a "." and the algorithm name being used.	No
property	This attribute can mean two different things, it depends on the presence of the verifyproperty attribute. **If you don't set the verifyproperty attribute**, property specifies the name of the property to be set with the generated checksum value. **If you set the verifyproperty attribute**, property specifies the checksum you expect to be generated (the checksum itself, not a name of a property containing the checksum). This cannot be specified when fileext is being used or when the number of files for which checksums is to be generated is greater than 1.	No
pattern	Specifies the pattern to use as a pattern suitable for MessageFormat where `{0}` is replaced with the checksum and `{1}` with the file name. *Since Ant 1.7.0*	No - default is "{0}".
format	Specifies the pattern to use as one of a well-known format. Supported values are	No - default is "CHECKSUM".

	<table><tr><th>name</th><th>pattern</th><th>description</th></tr><tr><td>CHECKSUM</td><td>{0}</td><td>only the checksum itself</td></tr><tr><td>MD5SUM</td><td>{0} *{1}</td><td>the format of GNU textutils md5sum</td></tr><tr><td>SVF</td><td>MD5 ({1}) = {0}</td><td>the format of BSDs md5 command</td></tr></table> *Since Ant 1.7.0*	
totalproperty	If specified, this attribute specifies the name of the property that will hold a checksum of all the checksums and file paths. The individual checksums and the relative paths to the files within the resource collections in which they are defined will be used to compute this checksum. (The file separators in the paths will be converted to '/' before computation to ensure platform portability). *since Ant 1.6*	No
forceoverwrite	Overwrite existing files even if the destination files are newer. Defaults to "no".	No
verifyproperty	Specifies the name of the property to be set with "true" or "false" depending upon whether the generated checksum matches the existing checksum. When this is set, the generated checksum is not written to a file or property, but rather, the content of the file or property is used to check against the generated checksum.	No
readbuffersize	The size of the buffer (in bytes) to use when reading a file. Defaults to "8192" - you may get a better performance on big files if you increase this value.	No

Parameters specified as nested elements

resource collection

Resource collections are used to select files for which checksums should be generated.

Examples

Example 1

```
<checksum file="foo.bar"/>
```

Generates a MD5 checksum for foo.bar and stores the checksum in the destination file foo.bar.MD5. foo.bar.MD5 is overwritten only if foo.bar is newer than itself.

Example 2

```
<checksum file="foo.bar" forceOverwrite="yes"/>
```

Generates a MD5 checksum for foo.bar and stores the checksum in foo.bar.MD5. If foo.bar.MD5 already exists, it is overwritten.

Example 3

```
<checksum file="foo.bar" property="foobarMD5"/>
```

Generates a MD5 checksum for foo.bar and stores it in the Project Property foobarMD5.

Example 4

```
<checksum file="foo.bar" verifyProperty="isMD5ok"/>
```

Generates a MD5 checksum for foo.bar, compares it against foo.bar.MD5 and sets isMD5ok to either true or false, depending upon the result.

Example 5

```
<checksum file="foo.bar" algorithm="SHA-512" fileext="asc"/>
```

Generates a SHA-512 checksum for foo.bar and stores the checksum in the destination file foo.bar.asc. foo.bar.asc is overwritten only if foo.bar is newer than itself.

Example 6

```
<checksum file="foo.bar" property="${md5}" verifyProperty="isEqual"/>
```

Generates a MD5 checksum for foo.bar, compares it against the value of the property md5, and sets isEqual to either true or false, depending upon the result.

Example 7

```
<checksum>
  <fileset dir=".">
    <include name="foo*"/>
  </fileset>
</checksum>
```

Works just like Example 1, but generates a .MD5 file for every file that begins with the name foo.

Example 8

```
<condition property="isChecksumEqual">
  <checksum>
    <fileset dir=".">
      <include name="foo.bar"/>
    </fileset>
  </checksum>
</condition>
```

Works like Example 4, but only sets isChecksumEqual to true, if the checksum matches - it will never be set to false. This example demonstrates use with the Condition task.

Note:

When working with more than one file, if condition and/or verifyproperty is used, the result will be true only if the checksums matched correctly for all files being considered.

Chgrp

Since Ant 1.6.

Description

Changes the group of a file or all files inside specified directories. Right now it has effect only under Unix. The group attribute is equivalent to the corresponding argument for the chgrp command.

FileSets, DirSets or FileLists can be specified using nested `<fileset>`, `<dirset>` and `<filelist>` elements.

Starting with Ant 1.7, this task supports arbitrary Resource Collections as nested elements.

By default this task will use a single invocation of the underlying chgrp command. If you are working on a large number of files this may result in a command line that is too long for your operating system. If you encounter such problems, you should set the maxparallel attribute of this task to a non-zero value. The number to use highly depends on the length of your file names (the depth of your directory tree) and your operating system, so you'll have to experiment a little. POSIX recommends command line length limits of at least 4096 characters, this may give you an approximation for the number you could use as initial value for these experiments.

By default this task won't do anything unless it detects it is running on a Unix system. If you know for sure that you have a "chgrp" executable on your PATH that is command line compatible with the Unix command, you can use the task's os attribute and set its value to your current os.

Parameters

Attribute	Description	Required
file	the file or directory of which the group must be changed.	Yes, unless nested `<fileset|filelist|dirset>` elements are specified
group	the new group.	Yes
parallel	process all specified files using a single `chgrp` command. Defaults to true.	No
type	One of *file*, *dir* or *both*. If set to *file*, only the group of plain files are going to be changed. If set to *dir*, only the directories are considered. **Note:** The type attribute does not apply to nested *dirset*s - *dirset*s always implicitly assume type to be *dir*.	No, default is *file*
maxparallel	Limit the amount of parallelism by passing at most this many sourcefiles at once. Set it to <= 0 for unlimited. Defaults to unlimited.	No
verbose	Whether to print a summary after execution or not. Defaults to `false`.	No
os	list of Operating Systems on which the command may be executed.	No
osfamily	OS family as used in the <os> condition.	No - defaults to "unix"

Examples

```
<chgrp file="${dist}/start.sh" group="coders"/>
```

makes the "start.sh" file belong to the coders group on a UNIX system.

```
<chgrp group="coders">
  <fileset dir="${dist}/bin" includes="**/*.sh"/>
</chgrp>
```

makes all ".sh" files below `${dist}/bin` belong to the coders group on a UNIX system.

```
<chgrp group="coders">
  <fileset dir="shared/sources1">
    <exclude name="**/trial/**"/>
  </fileset>
  <fileset refid="other.shared.sources"/>
</chgrp>
```

makes all files below `shared/sources1` (except those below any directory named trial) belong to the coders group on a UNIX system. In addition all files belonging to a FileSet with `id` `other.shared.sources` get the same group.

```
<chgrp group="webdev" type="file">
  <fileset dir="/web">
    <include name="**/*.test.jsp"/>
    <include name="**/*.new"/>
  </fileset>
  <dirset dir="/web">
    <include name="**/test_*"/>
  </dirset>
</chmod>
```

makes all `.test.jsp`, and `.new` files belong to group webdev. Directories beginning with `test_` also will belong to webdev, but if there is a directory that ends in `.new` or a file that begins with `test_` it will be unaffected.

Chmod

Description

Changes the permissions of a file or all files inside specified directories. Right now it has effect only under Unix or NonStop Kernel (Tandem). The permissions are also UNIX style, like the argument for the chmod command.

See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task holds an implicit FileSet and supports all of FileSet's attributes and nested elements directly. More sets can be specified using nested `<fileset>` or `<dirset>` (*since Ant 1.6*) elements.

Starting with Ant 1.6, this task also supports nested filelists.

Starting with Ant 1.7, this task supports arbitrary Resource Collections as nested elements.

By default this task will use a single invocation of the underlying chmod command. If you are working on a large number of files this may result in a command line that is too long for your operating system. If you encounter such problems, you should set the maxparallel attribute of this task to a non-zero value. The number to use highly depends on the length of your file names (the depth of your directory tree) and your operating system, so you'll have to experiment a little. POSIX recommends command line length limits of at least 4096 characters, this may give you an approximation for the number you could use as initial value for these experiments.

By default this task won't do anything unless it detects it is running on a Unix system. If you know for sure that you have a "chmod" executable on your PATH that is command line compatible with the Unix command, you can use the task's os attribute and set its value to your current os.

Parameters

Attribute	Description	Required
file	the file or single directory of which the permissions must be changed.	exactly one of the two or nested `<fileset/list>` elements.
dir	the directory which holds the files whose permissions must be changed. **Note:** for backwards compatibility reasons `<chmod dir="some-dir"/>` will only change the permissions on "some-dir" but not recurse into it, unless you also specify any patterns.	
perm	the new permissions.	Yes
includes	comma- or space-separated list of patterns of files that must be included.	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
parallel	process all specified files using a single `chmod` command. Defaults to true.	No
type	One of *file*, *dir* or *both*. If set to *file*, only the permissions of plain files are going to be changed. If set to *dir*, only the directories are considered. **Note:** The type attribute does not apply to nested *dirset*s - *dirset*s always implicitly assume type to be *dir*.	No, default is *file*
maxparallel	Limit the amount of parallelism by passing at most this many sourcefiles at once. Set it to <= 0 for unlimited. Defaults to unlimited. *Since Ant 1.6.*	No

verbose	Whether to print a summary after execution or not. Defaults to `false`. *Since Ant 1.6.*	No
os	list of Operating Systems on which the command may be executed.	No
osfamily	OS family as used in the <os> condition.	No - defaults to "unix"

Examples

```
<chmod file="${dist}/start.sh" perm="ugo+rx"/>
```

makes the "start.sh" file readable and executable for anyone on a UNIX system.

```
<chmod file="${dist}/start.sh" perm="700"/>
```

makes the "start.sh" file readable, writable and executable only for the owner on a UNIX system.

```
<chmod dir="${dist}/bin" perm="ugo+rx"
       includes="**/*.sh"/>
```

makes all ".sh" files below `${dist}/bin` readable and executable for anyone on a UNIX system.

```
<chmod perm="g+w">
  <fileset dir="shared/sources1">
    <exclude name="**/trial/**"/>
  </fileset>
  <fileset refid="other.shared.sources"/>
</chmod>
```

makes all files below `shared/sources1` (except those below any directory named trial) writable for members of the same group on a UNIX system. In addition all files belonging to a FileSet with `id` `other.shared.sources` get the same permissions.

```
<chmod perm="go-rwx" type="file">
  <fileset dir="/web">
    <include name="**/*.cgi"/>
    <include name="**/*.old"/>
  </fileset>
  <dirset dir="/web">
    <include name="**/private_*"/>
  </dirset>
</chmod>
```

keeps non-owners from touching cgi scripts, files with a `.old` extension or directories beginning with `private_`. A directory ending in `.old` or a file beginning with private_ would remain unaffected.

Note on maxparallel attribute

Some shells have a limit of the number of characters that a command line may contain. This maximum limit varies from shell to shell and from operating system to operating system. If one has a large number of files to change mode on, consider using the *maxparallel* attribute. For example when using AIX and the limit is reached, the system responds with a warning: "Warning: UNIXProcess.forkAndExec native error: The parameter or environment lists are too long". A value of about 300 seems to result in a command line that is acceptable.

Chown

Since Ant 1.6.

Description

Changes the owner of a file or all files inside specified directories. Right now it has effect only under Unix. The owner attribute is equivalent to the corresponding argument for the chown command.

FileSets, DirSets or FileLists can be specified using nested `<fileset>`, `<dirset>` and `<filelist>` elements.

Starting with Ant 1.7, this task supports arbitrary Resource Collections as nested elements.

By default this task will use a single invocation of the underlying chown command. If you are working on a large number of files this may result in a command line that is too long for your operating system. If you encounter such problems, you should set the maxparallel attribute of this task to a non-zero value. The number to use highly depends on the length of your file names (the depth of your directory tree) and your operating system, so you'll have to experiment a little. POSIX recommends command line length limits of at least 4096 characters, this may give you an approximation for the number you could use as initial value for these experiments.

By default this task won't do anything unless it detects it is running on a Unix system. If you know for sure that you have a "chown" executable on your PATH that is command line compatible with the Unix command, you can use the task's os attribute and set its value to your current os.

Parameters

Attribute	Description	Required
file	the file or directory of which the owner must be changed.	Yes or nested `<fileset/list>` elements.
owner	the new owner.	Yes
parallel	process all specified files using a single `chown` command. Defaults to true.	No
type	One of *file*, *dir* or *both*. If set to *file*, only the owner of plain files are going to be changed. If set to *dir*, only the directories are considered. **Note:** The type attribute does not apply to nested *dirset*s - *dirset*s always implicitly assume type to be *dir*.	No, default is *file*
maxparallel	Limit the amount of parallelism by passing at most this many sourcefiles at once. Set it to <= 0 for unlimited. Defaults to unlimited.	No
verbose	Whether to print a summary after execution or not. Defaults to `false`.	No
os	list of Operating Systems on which the command may be executed.	No
osfamily	OS family as used in the <os> condition.	No - defaults to "unix"

Examples

```
<chown file="${dist}/start.sh" owner="coderjoe"/>
```

makes the "start.sh" file belong to coderjoe on a UNIX system.

```
    <chown owner="coderjoe">
      <fileset dir="${dist}/bin" includes="**/*.sh"/>
    </chown>
```

makes all ".sh" files below `${dist}/bin` belong to coderjoe on a UNIX system.

```
<chown owner="coderjoe">
  <fileset dir="shared/sources1">
    <exclude name="**/trial/**"/>
  </fileset>
  <fileset refid="other.shared.sources"/>
</chown>
```

makes all files below `shared/sources1` (except those below any directory named trial) belong to coderjoe on a UNIX system. In addition all files belonging to a FileSet with `id` `other.shared.sources` get the same owner.

```
<chown owner="webadmin" type="file">
  <fileset dir="/web">
    <include name="**/*.cgi"/>
    <include name="**/*.old"/>
  </fileset>
  <dirset dir="/web">
    <include name="**/private_*"/>
  </dirset>
</chmod>
```

makes cgi scripts, files with a `.old` extension or directories beginning with `private_` belong to the user named webadmin. A directory ending in `.old` or a file beginning with `private_` would remain unaffected.

Concat

Description

Concatenates one or more resources to a single file or to the console. The destination file will be created if it does not exist unless the resource list is empty and ignoreempty is true.

Since Ant 1.7.1, this task can be used as a Resource Collection that will return exactly one resource.

Resource Collections are used to select which resources are to be concatenated. There is no singular attribute to specify a single resource to cat.

Parameters

Attribute	Description	Required
destfile	The destination file for the concatenated stream. If not specified the console will be used instead.	No
append	Specifies whether or not the file specified by 'destfile' should be appended. Defaults to "no".	No
force	Specifies whether or not the file specified by 'destfile' should be written to even if it is newer than all source files. **deprecated, use the overwrite attribute instead** Defaults to "yes".	No
overwrite	Specifies whether or not the file specified by 'destfile' should be written to even if it is newer than all source files. *since Ant 1.8.2*. Defaults to "yes".	No
forceReadOnly	Overwrite read-only destination files. *since Ant 1.8.2*	No; defaults to false.
encoding	Specifies the encoding for the input files. Please see http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html for a list of possible values. Defaults to the platform's default character encoding.	No
outputencoding	The encoding to use when writing the output file *since Ant 1.6*. Defaults to the value of the encoding attribute if given or the default JVM encoding otherwise.	No
fixlastline	Specifies whether or not to check if each file concatenated is terminated by a new line. If this attribute is "yes" a new line will be appended to the stream if the file did not end in a new line. *since Ant 1.6*. Defaults to "no". This attribute does not apply to embedded text.	No
eol	Specifies what the end of line character are for use by the fixlastline attribute. *since Ant 1.6* Valid values for this property are: • cr: a single CR • lf: a single LF • crlf: the pair CRLF • mac: a single CR • unix: a single LF • dos: the pair CRLF The default is platform dependent. For Unix platforms, the default is "lf". For DOS based systems (including Windows), the default is "crlf". For Mac OS, the default is "cr".	No

binary	*since Ant 1.6.2* If this attribute is set to true, the task concatenates the files in a byte by byte fashion. If this attribute is false, concat will not normally work for binary files due to character encoding issues. If this option is set to true, the destfile attribute must be set, and the task cannot used nested text. Also the attributes encoding, outputencoding, filelastline cannot be used. The default is "false".	No
ignoreempty	*Since Ant 1.8.0* Specifies whether or not the file specified by 'destfile' should be created if the source resource list is empty. Defaults to "true".	No

Parameters specified as nested elements

Resource Collection

since Ant 1.7.

Any of the various Resource Collection types can specify the resources to be concatenated.

filterchain

since Ant 1.6.

The concat task supports nested FilterChains.

header, footer

since Ant 1.6.

Used to prepend or postpend text into the concatenated stream.

The text may be in-line or be in a file.

Attribute	Description	Required
filtering	Whether to filter the text provided by this sub element, default is "yes".	No
file	A file to place at the head or tail of the concatenated text.	No
trim	Whether to trim the value, default is "no"	No
trimleading	Whether to trim leading white space on each line, default is "no"	No

Examples

Concatenate a string to a file:

```
<concat destfile="README">Hello, World!</concat>
```

Concatenate a series of files to the console:

```
<concat>
  <fileset dir="messages" includes="*important*"/>
</concat>
```

Concatenate a single file, appending if the destination file exists:

```
<concat destfile="NOTES" append="true">
```

```
  <filelist dir="notes" files="note.txt"/>
</concat>
```

Concatenate a series of files, update the destination file only if is older that all the source files:

```
<concat destfile="${docbook.dir}/all-sections.xml"
        force="no">
  <filelist dir="${docbook.dir}/sections"
       files="introduction.xml,overview.xml"/>
  <fileset dir="${docbook.dir}"
       includes="sections/*.xml"
       excludes="introduction.xml,overview.xml"/>
</concat>
```

Concatenate a series of files, expanding ant properties

```
 <concat destfile="${build.dir}/subs">
    <path>
      <fileset dir="${src.dir}" includes="*.xml"/>
      <pathelement location="build.xml"/>
    </path>
    <filterchain>
      <expandproperties/>
    </filterchain>
 </concat>
```

Filter the lines containing project from build.xml and output them to report.output, prepending with a header

```
 <concat destfile="${build.dir}/report.output">
    <header filtering="no" trimleading="yes">
        Lines that contain project
        ==========================
    </header>
    <path path="build.xml"/>
    <filterchain>
       <linecontains>
         <contains value="project"/>
       </linecontains>
    </filterchain>
 </concat>
```

Concatenate a number of binary files.

```
 <concat destfile="${build.dir}/dist.bin" binary="yes">
   <fileset file="${src.dir}/scripts/dist.sh" />
   <fileset file="${build.dir}/dist.tar.bz2" />
 </concat>
```

Copy

Description

Copies a file or resource collection to a new file or directory. By default, files are only copied if the source file is newer than the destination file, or when the destination file does not exist. However, you can explicitly overwrite files with the `overwrite` attribute.

Resource Collections are used to select a group of files to copy. To use a resource collection, the `todir` attribute must be set. **Note** that some resources (for example the file resource) return absolute paths as names and the result of using them without using a nested mapper (or the flatten attribute) may not be what you expect.

Note: If you employ filters in your copy operation, you should limit the copy to text files. Binary files will be corrupted by the copy operation. This applies whether the filters are implicitly defined by the filter task or explicitly provided to the copy operation as filtersets. *See encoding note*.

Parameters

Attribute	Description	Required
file	The file to copy.	Yes, unless a nested resource collection element is used.
preservelastmodified	Give the copied files the same last modified time as the original source files.	No; defaults to false.
tofile	The file to copy to.	With the `file` attribute, either `tofile` or `todir` can be used. With nested resource collection elements, if the number of included files is greater than 1, or if only the `dir` attribute is specified in the `<fileset>`, or if the `file` attribute is also specified, then only `todir` is allowed.
todir	The directory to copy to.	
overwrite	Overwrite existing files even if the destination files are newer.	No; defaults to false.
force	Overwrite read-only destination files. *since Ant 1.8.2*	No; defaults to false.
filtering	Indicates whether token filtering using the global build-file filters should take place during the copy. *Note*: Nested `<filterset>` elements will always be used, even if this attribute is not specified, or its value is `false` (`no`, or `off`).	No; defaults to false.
flatten	Ignore the directory structure of the source files, and copy all files into the directory specified by the `todir` attribute. Note that you can achieve the same effect by using a flatten mapper.	No; defaults to false.
includeEmptyDirs	Copy any empty directories included in the FileSet(s).	No; defaults to true.
failonerror	If false, log a warning message, but do not stop the build, when the file to copy does not exist or one of	No; defaults to true.

	the nested filesets points to a directory that doesn't exist or an error occurs while copying.	
verbose	Log the files that are being copied.	No; defaults to false.
encoding	The encoding to assume when filter-copying the files. *since Ant 1.5*.	No - defaults to default JVM encoding
outputencoding	The encoding to use when writing the files. *since Ant 1.6*.	No - defaults to the value of the encoding attribute if given or the default JVM encoding otherwise.
enablemultiplemappings	If true the task will process to all the mappings for a given source path. If false the task will only process the first file or directory. This attribute is only relevant if there is a mapper subelement. *since Ant 1.6*.	No - defaults to false.
granularity	The number of milliseconds leeway to give before deciding a file is out of date. This is needed because not every file system supports tracking the last modified time to the millisecond level. Default is 1 second, or 2 seconds on DOS systems. This can also be useful if source and target files live on separate machines with clocks being out of sync. *since Ant 1.6.2*.	No

Parameters specified as nested elements

fileset or any other resource collection

Resource Collections are used to select groups of files to copy. To use a resource collection, the `todir` attribute must be set.

Prior to Ant 1.7 only `<fileset>` has been supported as a nested element.

mapper

You can define filename transformations by using a nested mapper element. The default mapper used by `<copy>` is the identity mapper.

Since Ant 1.6.3, one can use a filenamemapper type in place of the mapper element.

Note that the source name handed to the mapper depends on the resource collection you use. If you use `<fileset>` or any other collection that provides a base directory, the name passed to the mapper will be a relative filename, relative to the base directory. In any other case the absolute filename of the source will be used.

filterset

FilterSets are used to replace tokens in files that are copied. To use a FilterSet, use the nested `<filterset>` element.

It is possible to use more than one filterset.

filterchain

The Copy task supports nested FilterChains.

If `<filterset>` and `<filterchain>` elements are used inside the same `<copy>` task, all `<filterchain>` elements are processed first followed by `<filterset>` elements.

Examples

Copy a single file

```
<copy file="myfile.txt" tofile="mycopy.txt"/>
```

Copy a single file to a directory

```
<copy file="myfile.txt" todir="../some/other/dir"/>
```

Copy a directory to another directory

```
<copy todir="../new/dir">
  <fileset dir="src_dir"/>
</copy>
```

Copy a set of files to a directory

```
<copy todir="../dest/dir">
  <fileset dir="src_dir">
    <exclude name="**/*.java"/>
  </fileset>
</copy>

<copy todir="../dest/dir">
  <fileset dir="src_dir" excludes="**/*.java"/>
</copy>
```

Copy a set of files to a directory, appending `.bak` to the file name on the fly

```
<copy todir="../backup/dir">
  <fileset dir="src_dir"/>
  <globmapper from="*" to="*.bak"/>
</copy>
```

Copy a set of files to a directory, replacing @TITLE@ with Foo Bar in all files.

```
<copy todir="../backup/dir">
  <fileset dir="src_dir"/>
  <filterset>
    <filter token="TITLE" value="Foo Bar"/>
  </filterset>
</copy>
```

Collect all items from the current CLASSPATH setting into a destination directory, flattening the directory structure.

```
<copy todir="dest" flatten="true">
  <path>
    <pathelement path="${java.class.path}"/>
  </path>
</copy>
```

Copies some resources to a given directory.

```
<copy todir="dest" flatten="true">
  <resources>
    <file file="src_dir/file1.txt"/>
    <url url="http://ant.apache.org/index.html"/>
  </resources>
</copy>
```

If the example above didn't use the flatten attribute, the `<file>` resource would have returned its full path as source and target name and would not have been copied at all. In general it is a good practice to use an explicit mapper together with resources that use an absolute path as their names.

Copies the two newest resources into a destination directory.

```
<copy todir="dest" flatten="true">
  <first count="2">
    <sort>
      <date xmlns="antlib:org.apache.tools.ant.types.resources.comparators"/>
      <resources>
        <file file="src_dir/file1.txt"/>
        <file file="src_dir/file2.txt"/>
        <file file="src_dir/file3.txt"/>
        <url url="http://ant.apache.org/index.html"/>
      </resources>
    </sort>
  </first>
</copy>
```

The paragraph following the previous example applies to this example as well.

Unix Note: File permissions are not retained when files are copied; they end up with the default `UMASK` permissions instead. This is caused by the lack of any means to query or set file permissions in the current Java runtimes. If you need a permission-preserving copy function, use `<exec executable="cp" ... >` instead.

Windows Note: If you copy a file to a directory where that file already exists, but with different casing, the copied file takes on the case of the original. The workaround is to delete the file in the destination directory before you copy it.

Important Encoding Note: The reason that binary files when filtered get corrupted is that filtering involves reading in the file using a Reader class. This has an encoding specifing how files are encoded. There are a number of different types of encoding - UTF-8, UTF-16, Cp1252, ISO-8859-1, US-ASCII and (lots) others. On Windows the default character encoding is Cp1252, on Unix it is usually UTF-8. For both of these encoding there are illegal byte sequences (more in UTF-8 than for Cp1252).

How the Reader class deals with these illegal sequences is up to the implementation of the character decoder. The current Sun Java implemenation is to map them to legal characters. Previous Sun Java (1.3 and lower) threw a MalformedInputException. IBM Java 1.4 also thows this exception. It is the mapping of the characters that cause the corruption.

On Unix, where the default is normally UTF-8, this is a *big* problem, as it is easy to edit a file to contain non US Ascii characters from ISO-8859-1, for example the Danish oe character. When this is copied (with filtering) by Ant, the character get converted to a question mark (or some such thing).

There is not much that Ant can do. It cannot figure out which files are binary - a UTF-8 version of Korean will have lots of bytes with the top bit set. It is not informed about illegal character sequences by current Sun Java implementions.

One trick for filtering containing only US-ASCII is to use the ISO-8859-1 encoding. This does not seem to contain illegal character sequences, and the lower 7 bits are US-ASCII. Another trick is to change the LANG environment variable from something like "us.utf8" to "us".

Copydir

Deprecated

This task has been deprecated. Use the Copy task instead.

Description

Copies a directory tree from the source to the destination.

It is possible to refine the set of files that are being copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `src`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Parameters

Attribute	Description	Required
src	the directory to copy.	Yes
dest	the directory to copy to.	Yes
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
filtering	indicates whether token filtering should take place during the copy	No
flatten	ignore directory structure of source directory, copy all files into a single directory, specified by the `dest` attribute (default is `false`).	No
forceoverwrite	overwrite existing files even if the destination files are newer (default is false).	No

Examples

```
<copydir src="${src}/resources"
         dest="${dist}"
/>
```

copies the directory `${src}/resources` to `${dist}`.

```
<copydir src="${src}/resources"
         dest="${dist}"
```

```
          includes="**/*.java"
          excludes="**/Test.java"
  />
```

copies the directory `${src}/resources` to `${dist}` recursively. All java files are copied, except for files with the name `Test.java`.

```
  <copydir src="${src}/resources"
           dest="${dist}"
           includes="**/*.java"
           excludes="mypackage/test/**"/>
```

copies the directory `${src}/resources` to `${dist}` recursively. All java files are copied, except for the files under the `mypackage/test` directory.

Copyfile

Deprecated

This task has been deprecated. Use the Copy task instead.

Description

Copies a file from the source to the destination. The file is only copied if the source file is newer than the destination file, or when the destination file does not exist.

Parameters

Attribute	Description	Required
src	the filename of the file to copy.	Yes
dest	the filename of the file where to copy to.	Yes
filtering	indicates whether token filtering should take place during the copy	No
forceoverwrite	overwrite existing files even if the destination files are newer (default is false).	No

Examples

```
<copyfile src="test.java" dest="subdir/test.java"/>

<copyfile src="${src}/index.html" dest="${dist}/help/index.html"/>
```

Delete

Description

Deletes a single file, a specified directory and all its files and subdirectories, or a set of files specified by one or more resource collections. The literal implication of `<fileset>` is that directories are not included; however the removal of empty directories can be triggered when using nested filesets by setting the `includeEmptyDirs` attribute to *true*. Note that this attribute is meaningless in the context of any of the various resource collection types that *do* include directories, but that no attempt will be made to delete non-empty directories in any case. Whether a directory is empty or not is decided by looking into the filesystem - include or exclude patterns don't apply here.

If you use this task to delete temporary files created by editors and it doesn't seem to work, read up on the default exclusion set in **Directory-based Tasks**, and see the `defaultexcludes` attribute below.

For historical reasons `<delete dir="x"/>` is different from `<delete><fileset dir="x"/></delete>`, it will try to remove everything inside "x" including "x" itself, not taking default excludes into account, blindly following all symbolic links. If you need more control, use a nested `<fileset>`.

Parameters

Attribute	Description	Required
file	The file to delete, specified as either the simple filename (if the file exists in the current base directory), a relative-path filename, or a full-path filename.	At least one of the two, unless nested resource collections are specified
dir	The directory to delete, including all its files and subdirectories. **Note:** `dir` is *not* used to specify a directory name for `file`; `file` and `dir` are independent of each other. **WARNING:** Do **not** set `dir` to `"."`, `"${basedir}"`, or the full-pathname equivalent unless you truly *intend* to recursively remove the entire contents of the current base directory (and the base directory itself, if different from the current working directory).	
verbose	Whether to show the name of each deleted file.	No, default "false"
quiet	If the specified file or directory does not exist, do not display a diagnostic message (unless Ant has been invoked with the `-verbose` or `-debug` switches) or modify the exit status to reflect an error. When set to "true", if a file or directory cannot be deleted, no error is reported. This setting emulates the `-f` option to the Unix *rm* command. Setting this to "true" implies setting `failonerror` to "false".	No, default "false"
failonerror	Controls whether an error (such as a failure to delete a file) stops the build or is merely reported to the screen. Only relevant if `quiet` is "false".	No, default "true"
includeemptydirs	Whether to delete empty directories when using filesets.	No, default "false"
includes	*Deprecated.* Use resource collections. Comma- or space-separated list of patterns of files that must be deleted. All files are relative to the directory specified in `dir`.	No
includesfile	*Deprecated.* Use resource collections. The name of a file. Each line of	No

	this file is taken to be an include pattern.	
excludes	*Deprecated.* Use resource collections. Comma- or space-separated list of patterns of files that must be excluded from the deletion list. All files are relative to the directory specified in `dir`. No files (except default excludes) are excluded when omitted.	No
excludesfile	*Deprecated.* Use resource collections. The name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	*Deprecated.* Use resource collections. Whether to use default excludes.	No, default "true"
deleteonexit	Indicates whether to use File#deleteOnExit() if there is a failure to delete a file, this causes the jvm to attempt to delete the file when the jvm process is terminating. *Since Ant 1.6.2*	No, default "false"
removeNotFollowedSymlinks	Whether symbolic links (not the files/directories they link to) should be removed if they haven't been followed because followSymlinks was false or the maximum number of symbolic links was too big. *Since Ant 1.8.0*	No, default "false"

Examples

```
<delete file="/lib/ant.jar"/>
```

deletes the file `/lib/ant.jar`.

```
<delete dir="lib"/>
```

deletes the `lib` directory, including all files and subdirectories of `lib`.

```
<delete>
  <fileset dir="." includes="**/*.bak"/>
</delete>
```

deletes all files with the extension `.bak` from the current directory and any subdirectories.

```
<delete includeEmptyDirs="true">
  <fileset dir="build"/>
</delete>
```

deletes all files and subdirectories of `build`, including `build` itself.

```
<delete includeemptydirs="true">
  <fileset dir="build" includes="**/*"/>
</delete>
```

deletes all files and subdirectories of `build`, without `build` itself.

```
<delete includeemptydirs="true">
  <fileset dir="src" includes="**/.svn" defaultexcludes="false"/>
</delete>
```

deletes the subversion metadata directories under `src`. Because `.svn` is on of the default excludes you have to use the `defaultexcludes` flag, otherwise Ant wont delete these directories and the files in it.

Deltree

Deprecated

This task has been deprecated. Use the Delete task instead.

Description

Deletes a directory with all its files and subdirectories.

Parameters

Attribute	Description	Required
dir	the directory to delete.	Yes

Examples

```
<deltree dir="dist"/>
```

deletes the directory `dist`, including its files and subdirectories.

```
<deltree dir="${dist}"/>
```

deletes the directory `${dist}`, including its files and subdirectories.

FixCRLF

Description

Adjusts a text file to local conventions.

The set of files to be adjusted can be refined with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. Patterns provided through the *includes* or *includesfile* attributes specify files to be included. Patterns provided through the *exclude* or *excludesfile* attribute specify files to be excluded. Additionally, default exclusions can be specified with the *defaultexcludes* attribute. See the section on directory-based tasks, for details of file inclusion/exclusion patterns and their usage.

This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

The output file is only written if it is a new file, or if it differs from the existing file. This prevents spurious rebuilds based on unchanged files which have been regenerated by this task.

Since **Ant 1.7**, this task can be used in a filterchain.

Parameters

Attribute	Description	Required	
		As Task	As Filter
srcDir	Where to find the files to be fixed up.	One of these	
file	Name of a single file to fix. **Since Ant 1.7**		
destDir	Where to place the corrected files. Defaults to srcDir (replacing the original file).	No	
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No	
includesfile	the name of a file. Each line of this file is taken to be an include pattern.	No	
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No	
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern.	No	
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No	
encoding	The encoding of the files.	No; defaults to default JVM encoding.	
outputencoding	The encoding to use when writing the files. **Since Ant 1.7**	No; defaults to the value of the encoding attribute.	
preservelastmodified	Whether to preserve the last modified date of source files. **Since Ant 1.6.3**	No; default is *false*	

eol	Specifies how end-of-line (EOL) characters are to be handled. The EOL characters are CR, LF and the pair CRLF. Valid values for this property are: • asis: leave EOL characters alone • cr: convert all EOLs to a single CR • lf: convert all EOLs to a single LF • crlf: convert all EOLs to the pair CRLF • mac: convert all EOLs to a single CR • unix: convert all EOLs to a single LF • dos: convert all EOLs to the pair CRLF Default is based on the platform on which you are running this task. For Unix platforms (including Mac OS X), the default is "lf". For DOS-based systems (including Windows), the default is "crlf". For Mac environments other than OS X, the default is "cr". This is the preferred method for specifying EOL. The "***cr***" attribute (see below) is now deprecated. *N.B.*: One special case is recognized. The three characters CR-CR-LF are regarded as a single EOL. Unless this property is specified as "asis", this sequence will be converted into the specified EOL type.	No
cr	***Deprecated.*** Specifies how CR characters are to be handled at end-of-line (EOL). Valid values for this property are: • asis: leave EOL characters alone. • add: add a CR before any single LF characters. The intent is to convert all EOLs to the pair CRLF. • remove: remove all CRs from the file. The intent is to convert all EOLs to a single LF. Default is based on the platform on which you are running this task. For Unix platforms, the default is "remove". For DOS based systems (including Windows), the default is "add". *N.B.*: One special case is recognized. The three characters CR-CR-LF are regarded as a single EOL. Unless this property is specified as "asis", this sequence will be converted into the specified EOL type.	No
javafiles	Used only in association with the "***tab***" attribute (see below), this boolean attribute indicates whether the fileset is a set of java source files ("yes"/"no"). Defaults to "no". See notes in section on "tab".	No
tab	Specifies how tab characters are to be handled. Valid values for this property are: • add: convert sequences of spaces which span a tab stop to tabs • asis: leave tab and space characters alone • remove: convert tabs to spaces Default for this parameter is "asis". *N.B.*: When the attribute "***javafiles***" (see above) is "true", literal TAB characters occurring within Java string or character constants are never modified. This functionality also requires the recognition of Java-style	No

	comments. *N.B.*: There is an incompatibility between this and the previous version in the handling of white space at the end of lines. This version does ***not*** remove trailing whitespace on lines.	
tablength	TAB character interval. Valid values are between 2 and 80 inclusive. The default for this parameter is 8.	No
eof	Specifies how DOS end of file (control-Z) characters are to be handled. Valid values for this property are: • add: ensure that there is an EOF character at the end of the file • asis: leave EOF characters alone • remove: remove any EOF character found at the end Default is based on the platform on which you are running this task. For Unix platforms, the default is remove. For DOS based systems (including Windows), the default is asis.	No
fixlast	Whether to add a missing EOL to the last line of a processed file. **Since Ant 1.6.1**	No; default is *true*

Examples

```
<fixcrlf srcdir="${src}" includes="**/*.sh"
         eol="lf" eof="remove" />
```

Replaces EOLs with LF characters and removes eof characters from the shell scripts. Tabs and spaces are left as is.

```
<fixcrlf srcdir="${src}"
         includes="**/*.bat" eol="crlf" />
```

Replaces all EOLs with cr-lf pairs in the batch files. Tabs and spaces are left as is. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems.

```
<fixcrlf srcdir="${src}"
         includes="**/Makefile" tab="add" />
```

Sets EOLs according to local OS conventions, and converts sequences of spaces and tabs to the minimal set of spaces and tabs which will maintain spacing within the line. Tabs are set at 8 character intervals. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems. Many versions of make require tabs prior to commands.

```
<fixcrlf srcdir="${src}" includes="**/*.java"
         tab="remove" tablength="3"
         eol="lf" javafiles="yes" />
```

Converts all EOLs in the included java source files to a single LF. Replace all TAB characters except those in string or character constants with spaces, assuming a tab width of 3. If run on a unix system, any CTRL-Z EOF characters at the end of the file are removed. On DOS/Windows, any such EOF characters will be left untouched.

```
<fixcrlf srcdir="${src}"
         includes="**/README*" tab="remove" />
```

Sets EOLs according to local OS conventions, and converts all tabs to spaces, assuming a tab width of 8. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems. You never know what editor a user will use to browse READMEs.

Get

Description

Gets files from URLs. When the verbose option is "on", this task displays a '.' for every 100 Kb retrieved. Any URL schema supported by the runtime is valid here, including http:, ftp: and jar:;

The *usetimestamp* option enables you to control downloads so that the remote file is only fetched if newer than the local copy. If there is no local copy, the download always takes place. When a file is downloaded, the timestamp of the downloaded file is set to the remote timestamp. NB: This timestamp facility only works on downloads using the HTTP protocol.

A username and password can be specified, in which case basic 'slightly encoded plain text' authentication is used. This is only secure over an HTTPS link.

Proxies. Since Ant 1.7.0, Ant running on Java1.5 or later can use the proxy settings of the operating system if enabled with the `-autoproxy` option. There is also the <setproxy> task for earlier Java versions. With proxies turned on, `<get>` requests against localhost may not work as expected, if the request is relayed to the proxy.

Parameters

Attribute	Description	Required
src	the URL from which to retrieve a file.	Yes or a nested resource collection
dest	the file or directory where to store the retrieved file(s).	Yes
verbose	show verbose progress information ("on"/"off").	No; default "false"
ignoreerrors	Log errors but don't treat as fatal.	No; default "false"
usetimestamp	conditionally download a file based on the timestamp of the local copy. HTTP only	No; default "false"
username	username for 'BASIC' http authentication	if password is set
password	password: required	if username is set
maxtime	Maximum time in seconds a single download may take, otherwise it will be interrupted and treated like a download error. *Since Ant 1.8.0*	No: default 0 which means no maximum time
retries	the per download number of retries on error *since Ant 1.8.0*	No; default "3"
skipexisting	skip files that already exist on the local filesystem *since Ant 1.8.0*	No; default "false"
httpusecaches	HTTP only - if true, allow caching at the HttpUrlConnection level. if false, turn caching off. **Note** this is only a hint to the underlying UrlConnection class, implementations and proxies are free to ignore the setting.	No; default "true"

Parameters specified as nested elements

any resource collection

Resource Collections are used to select groups of URLs to download. If the collection contains more than one resource, the dest attribute must point to a directory if it exists or a directory will be created if it doesn't exist. The destination file name use the last part of the path of the source URL unless you also specify a mapper.

mapper

You can define name transformations by using a nested mapper element. You can also use any filenamemapper type in place of the mapper element.

The mapper will receive the resource's name as argument. Any resource for which the mapper returns no or more than one mapped name will be skipped. If the returned name is a relative path, it will be considered relative to the *dest* attribute.

Examples

```
<get src="http://ant.apache.org/" dest="help/index.html"/>
```

Gets the index page of http://ant.apache.org/, and stores it in the file `help/index.html`.

```
<get src="http://www.apache.org/dist/ant/KEYS"
  dest="KEYS"
  verbose="true"
  usetimestamp="true"/>
```

Gets the PGP keys of Ant's (current and past) release managers, if the local copy is missing or out of date. Uses the verbose option for progress information.

```
<get src="https://insecure-bank.org/statement/user=1214"
  dest="statement.html"
  username="1214";
  password="secret"/>
```

Fetches some file from a server with access control. Because https is being used the fact that basic auth sends passwords in plaintext is moot if you ignore the fact that it is part of your build file which may be readable by third parties. If you need more security, consider using the input task to query for a password.

Using a macro like the following

```
<macrodef name="get-and-checksum">
  <attribute name="url"/>
  <attribute name="dest"/>
  <sequential>
    <local name="destdir"/>
    <dirname property="destdir" file="@{dest}"/>
    <get dest="${destdir}">
      <url url="@{url}"/>
      <url url="@{url}.sha1"/>
      <firstmatchmapper>
        <globmapper from="@{url}.sha1" to="@{dest}.sha"/>
        <globmapper from="@{url}" to="@{dest}"/>
      </firstmatchmapper>
    </get>
    <local name="checksum.matches"/>
    <local name="checksum.matches.fail"/>
    <checksum file="@{dest}" algorithm="sha" fileext=".sha"
              verifyproperty="checksum.matches"/>
    <condition property="checksum.matches.fail">
      <equals arg1="${checksum.matches}" arg2="false"/>
    </condition>
    <fail if="checksum.matches.fail">Checksum error</fail>
  </sequential>
</macrodef>
```

it is possible to download an artifacts together with its SHA1 checksum (assuming a certain naming convention for the checksum file, of course) and validate the checksum on the fly.

```
<get dest="downloads">
  <url url="http://ant.apache.org/index.html"/>
  <url url="http://ant.apache.org/faq.html"/>
</get>
```

Gets the index and FAQ pages of http://ant.apache.org/, and stores them in the directory `downloads` which will be created if necessary.

Mkdir

Description

Creates a directory. Also non-existent parent directories are created, when necessary. Does nothing if the directory already exist.

Parameters

Attribute	Description	Required
dir	the directory to create.	Yes

Examples

```
<mkdir dir="${dist}"/>
```

creates a directory `${dist}`.

```
<mkdir dir="${dist}/lib"/>
```

creates a directory `${dist}/lib`.

Move

Description

Moves a file to a new file or directory, or collections of files to a new directory. By default, the destination file is overwritten if it already exists. When *overwrite* is turned off, then files are only moved if the source file is newer than the destination file, or when the destination file does not exist.

Resource Collections are used to select a group of files to move. Only file system based resource collections are supported, this includes filesets, filelist and path. Prior to Ant 1.7 only `<fileset>` has been supported as a nested element. To use a resource collection, the `todir` attribute must be set.

Since Ant 1.6.3, the *file* attribute may be used to move (rename) an entire directory. If *tofile* denotes an existing file, or there is a directory by the same name in *todir*, the action will fail.

Parameters

Attribute	Description	Required
file	the file or directory to move	One of *file* or at least one nested resource collection element
preservelastmodified	Give the moved files the same last modified time as the original source files. (*Note*: Ignored on Java 1.1)	No; defaults to false.
tofile	the file to move to	With the *file* attribute, either *tofile* or *todir* can be used. With nested filesets, if the fileset size is greater than 1 or if the only entry in the fileset is a directory or if the *file* attribute is already specified, only *todir* is allowed
todir	the directory to move to	
overwrite	overwrite existing files even if the destination files are newer (default is "true")	No
force	Overwrite read-only destination files. *since Ant 1.8.2*	No; defaults to false.
filtering	indicates whether token filtering should take place during the move. See the filter task for a description of how filters work.	No
flatten	ignore directory structure of source directory, copy all files into a single directory, specified by the *todir* attribute (default is "false").Note that you can achieve the same effect by using a flatten mapper	No
includeEmptyDirs	Copy empty directories included with the nested FileSet(s). Defaults to "yes".	No
failonerror	If false, log a warning message, but do not stop the build, when the file to copy does not exist or one of the nested filesets points to a directory that doesn't exist or an error occurs while moving.	No; defaults to true.
verbose	Log the files that are being moved.	No; defaults to false.
encoding	The encoding to assume when filter-copying the files.	No - defaults to default JVM

	since Ant 1.5.	encoding
outputencoding	The encoding to use when writing the files. *since Ant 1.6*.	No - defaults to the value of the encoding attribute if given or the default JVM encoding otherwise.
enablemultiplemapping	If true the task will process to all the mappings for a given source path. If false the task will only process the first file or directory. This attribute is only relevant if there is a mapper subelement. *since Ant 1.6*.	No - defaults to false.
granularity	The number of milliseconds leeway to give before deciding a file is out of date. This is needed because not every file system supports tracking the last modified time to the millisecond level. Default is 0 milliseconds, or 2 seconds on DOS systems. This can also be useful if source and target files live on separate machines with clocks being out of sync. *since Ant 1.6*.	

Parameters specified as nested elements

mapper

You can define file name transformations by using a nested mapper element. The default mapper used by `<move>` is the identity.

Note that the source name handed to the mapper depends on the resource collection you use. If you use `<fileset>` or any other collection that provides a base directory, the name passed to the mapper will be a relative filename, relative to the base directory. In any other case the absolute filename of the source will be used.

filterchain

The Move task supports nested FilterChains.

If `<filterset>` and `<filterchain>` elements are used inside the same `<move>` task, all `<filterchain>` elements are processed first followed by `<filterset>` elements.

Examples

Move a single file (rename a file)

```
<move file="file.orig" tofile="file.moved"/>
```

Move a single file to a directory

```
<move file="file.orig" todir="dir/to/move/to"/>
```

Move a directory to a new directory

```
<move todir="new/dir/to/move/to">
  <fileset dir="src/dir"/>
</move>
```

or, since Ant 1.6.3:

```
<move file="src/dir" tofile="new/dir/to/move/to"/>
```

Move a set of files to a new directory

```
<move todir="some/new/dir">
  <fileset dir="my/src/dir">
    <include name="**/*.jar"/>
    <exclude name="**/ant.jar"/>
  </fileset>
</move>
```

Move a list of files to a new directory

```
<move todir="some/new/dir">
  <filelist dir="my/src/dir">
    <file name="file1.txt"/>
    <file name="file2.txt"/>
  </filelist>
</move>
```

Append `".bak"` to the names of all files in a directory.

```
<move todir="my/src/dir" includeemptydirs="false">
  <fileset dir="my/src/dir">
    <exclude name="**/*.bak"/>
  </fileset>
  <mapper type="glob" from="*" to="*.bak"/>
</move>
```

Patch

Description

Applies a diff file to originals. ; requires "patch" to be on the execution path.

Parameters

Attribute	Description	Required
patchfile	the file that includes the diff output	Yes
originalfile	the file to patch	No, tries to guess it from the diff file
destfile	the file to send the output to instead of patching the file(s) in place. *since Ant 1.6*	No.
backups	Keep backups of the unpatched files	No
quiet	Work silently unless an error occurs	No
reverse	Assume patch was created with old and new files swapped.	No
ignorewhitespace	Ignore whitespace differences.	No
strip	Strip the smallest prefix containing *num* leading slashes from filenames.	No
dir	The directory in which to run the patch command.	No, default is the project's basedir.
failonerror	Stop the buildprocess if the command exits with a return code signaling failure. Defaults to false. *since Ant 1.8.0*	No

Examples

```
  <patch patchfile="module.1.0-1.1.patch"/>
```

applies the diff included in *module.1.0-1.1.patch* to the files in base directory guessing the filename(s) from the diff output.

```
  <patch patchfile="module.1.0-1.1.patch" strip="1"/>
```

like above but one leading directory part will be removed. i.e. if the diff output looked like

```
--- a/mod1.0/A  Mon Jun  5 17:28:41 2000
+++ a/mod1.1/A  Mon Jun  5 17:28:49 2000
```

the leading *a/* will be stripped.

Rename

Deprecated

This task has been deprecated. Use the Move task instead.

Description

Renames a given file.

Parameters

Attribute	Description	Required
src	file to rename.	Yes
dest	new name of the file.	Yes
replace	Enable replacing of existing file (default: on).	No

Examples

```
  <rename src="foo.jar" dest="${name}-${version}.jar"/>
```

Renames the file `foo.jar` to `${name}-${version}.jar` (assuming `name` and `version` being predefined properties). If a file named `${name}-${version}.jar` already exists, it will be removed prior to renaming `foo.jar`.

RenameExtensions

Deprecated

This task has been deprecated. Use the move task with a glob mapper instead.

Description

Renames files in the `srcDir` directory ending with the `fromExtension` string so that they end with the `toExtension` string. Files are only replaced if `replace` is true

See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns. This task forms an implicit FileSet and supports most attributes of `<fileset>` (`dir` becomes `srcDir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Parameters

Attribute	Description	Required
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
fromExtention	The string that files must end in to be renamed	Yes
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
replace	Whether the file being renamed to should be replaced if it already exists	No
srcDir	The starting directory for files to search in	Yes
toExtension	The string that renamed files will end with on completion	Yes

Examples

```
<renameext srcDir="/source/project1" includes="**" excludes="**/samples/*"
fromExtension=".java.keep" toExtension=".java" replace="true"/>
```

Replace

Description

Replace is a directory based task for replacing the occurrence of a given string with another string in selected file.

If you want to replace a text that crosses line boundaries, you must use a nested `<replacetoken>` element.

The output file is only written if it differs from the existing file. This prevents spurious rebuilds based on unchanged files which have been regenerated by this task.

Parameters

Attribute	Description	Required
file	file for which the token should be replaced.	Exactly one of the two.
dir	The base directory to use when replacing a token in multiple files.	
encoding	The encoding of the files upon which replace operates.	No - defaults to default JVM encoding
token	the token which must be replaced.	Yes, unless a nested `replacetoken` element or the replacefilterfile attribute is used.
value	the new value for the token. When omitted, an empty string ("") is used.	No
summary	Indicates whether a summary of the replace operation should be produced, detailing how many token occurrences and files were processed	No, by default no summary is produced
propertyFile	valid property file from which properties specified using nested `<replacefilter>` elements are drawn.	Yes only if *property* attribute of `<replacefilter>` is used.
replacefilterfile	valid property file. Each property will be treated as a replacefilter where `token` is the name of the property and `value` is the properties value.	No.
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
preserveLastModified	Keep the file timestamp(s) even if the file(s) is(are) modified. *since Ant 1.8.0.*	No, defaults to false
failOnNoReplacements	Whether to fail the build if the task didn't do anything.	No, defaults to false

	since Ant 1.8.0.	

Examples

```
  <replace file="${src}/index.html" token="@@@" value="wombat"/>
```

replaces occurrences of the string "@@@" with the string "wombat", in the file `${src}/index.html`.

Parameters specified as nested elements

This task forms an implicit FileSet and supports most attributes of `<fileset>` as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Since Ant 1.8.0 this task supports any filesystem based resource collections as nested elements.

replacetoken and replacevalue

If either the text you want to replace or the replacement text cross line boundaries, you can use nested elements to specify them.

The elements support attributes:

Attribute	Description	Required
expandProperties	Whether to expand properties in the nested text. *since Ant 1.8.0.*	No, defaults to true.

Examples

```
<replace dir="${src}" value="wombat">
  <include name="**/*.html"/>
  <replacetoken><![CDATA[multi line
token]]></replacetoken>
</replace>
```

replaces occurrences of the string "multi line*n*token" with the string "wombat", in all HTML files in the directory `${src}`.Where *n* is the platform specific line separator.

```
<replace file="${src}/index.html">
  <replacetoken><![CDATA[two line
token]]></replacetoken>
  <replacevalue><![CDATA[two line
token]]></replacevalue>
</replace>
```

replacefilter

In addition to allowing for multiple replacements, optional nested `<replacefilter>` elements allow replacement values to be extracted from a property file. The name of this file is specified using the `<replace>` attribute *propertyFile*.

Attribute	Description	Required
token	The string to search for.	Yes unless a nested replacetoken is specified
value	The replacement string.	Either may be specified, but not both. Both can be omitted, if desired.
property	Name of the property whose value is to serve as the replacement value.	

Since Ant 1.8.0 token and value can be specified as nested elements just like in the task itself.

If neither *value* nor *property* is used, the value provided using the `<replace>` attribute *value* and/or the `<replacevalue>` element is used. If no value was specified using either of these options, the token is replaced with an empty string.

Examples

```
<replace
    file="configure.sh"
    value="defaultvalue"
    propertyFile="src/name.properties">
  <replacefilter
    token="@token1@"/>
  <replacefilter
    token="@token2@"
    value="value2"/>
  <replacefilter
    token="@token3@"
    property="property.key"/>
  <replacefilter>
    <replacetoken>@token4@</replacetoken>
    <replacevalue>value4</replacevalue>
  </replacefilter>
</replace>
```

In file `configure.sh`, replace all instances of "@token1@" with "defaultvalue", all instances of "@token2@" with "value2", and all instances of "@token3@" with the value of the property "property.key", as it appears in property file `src/name.properties`.

Note: It is possible to use either the *token*/`<replacetoken>` and *value*/`<replacevalue>` attributes/elements, the nested replacefilter elements, or both in the same operation.

ReplaceRegExp

Description

ReplaceRegExp is a directory based task for replacing the occurrence of a given regular expression with a substitution pattern in a selected file or set of files.

The output file is only written if it differs from the existing file. This prevents spurious rebuilds based on unchanged files which have been regenerated by this task.

Similar to regexp type mappers this task needs a supporting regular expression library and an implementation of `org.apache.tools.ant.util.regexp.Regexp`. See details in the documentation of the Regexp Type.

Parameters

Attribute	Description	Required
file	file for which the regular expression should be replaced.	Yes if no nested `<fileset>` is used
match	The regular expression pattern to match in the file(s)	Yes, if no nested `<regexp>` is used
replace	The substitution pattern to place in the file(s) in place of the regular expression.	Yes, if no nested `<substitution>` is used
flags	The flags to use when matching the regular expression. For more information, consult the Perl5 syntax g : Global replacement. Replace all occurrences found i : Case Insensitive. Do not consider case in the match m : Multiline. Treat the string as multiple lines of input, using "^" and "$" as the start or end of any line, respectively, rather than start or end of string. s : Singleline. Treat the string as a single line of input, using "." to match any character, including a newline, which normally, it would not match.	No
byline	Process the file(s) one line at a time, executing the replacement on one line at a time (*true/false*). This is useful if you want to only replace the first occurrence of a regular expression on each line, which is not easy to do when processing the file as a whole. Defaults to *false*.	No
encoding	The encoding of the file. *since Ant 1.6*	No - defaults to default JVM encoding
preserveLastModified	Keep the file timestamp(s) even if the file(s) is(are) modified. *since Ant 1.8.0.*	No, defaults to false

Examples

```
<replaceregexp file="${src}/build.properties"
               match="OldProperty=(.*)"
               replace="NewProperty=\1"
               byline="true"
```

```
/>
```

replaces occurrences of the property name "OldProperty" with "NewProperty" in a properties file, preserving the existing value, in the file `${src}/build.properties`

Parameters specified as nested elements

This task supports a nested FileSet element.

Since Ant 1.8.0 this task supports any filesystem based resource collections as nested elements.

This task supports a nested *Regexp* element to specify the regular expression. You can use this element to refer to a previously defined regular expression datatype instance.

```
<regexp id="id" pattern="alpha(.+)beta"/>
<regexp refid="id"/>
```

This task supports a nested *Substitution* element to specify the substitution pattern. You can use this element to refer to a previously defined substitution pattern datatype instance.

```
<substitution id="id" expression="beta\1alpha"/>
<substitution refid="id"/>
```

Examples

```
<replaceregexp byline="true">
  <regexp pattern="OldProperty=(.*)"/>
  <substitution expression="NewProperty=\1"/>
  <fileset dir=".">
    <include name="*.properties"/>
  </fileset>
</replaceregexp>
```

replaces occurrences of the property name "OldProperty" with "NewProperty" in a properties file, preserving the existing value, in all files ending in `.properties` in the current directory

```
<replaceregexp match="\s+" replace=" " flags="g" byline="true">
    <fileset dir="${html.dir}" includes="**/*.html"/>
</replaceregexp>
```

replaces all whitespaces (blanks, tabs, etc) by one blank remaining the line separator. So with input

```
<html>    <body>
<<TAB>><h1>    T E S T   </h1>  <<TAB>>
<<TAB>> </body></html>
```

would converted to

```
<html> <body>
 <h1> T E S T </h1> </body></html>
```

Sync

Since Ant 1.6

Description

Synchronize a target directory from the files defined in one or more Resource Collections.

Any file in the target directory that has not been matched by at least one of the nested resource collections gets removed. I.e. if you exclude a file in your sources and a file of that name is present in the target dir, it will get removed from the target.

Parameters

Attribute	Description	Required
todir	the target directory to sync with the resource collections	Yes
overwrite	Overwrite existing files even if the destination files are newer.	No; defaults to false.
includeEmptyDirs	Copy any empty directories included in the resource collection(s). **Note** this attribute also controls the behavior for any nested <preserveintarget> element. If this attribute is false (the default) empty directories that only exist in the target directory will be removed even if they are matched by the patterns of <preserveintarget>. This can be overridden by <preserveintarget>'s preserveEmptyDirs attribute.	No; defaults to false.
failonerror	If is set to false, log a warning message, but do not stop the build, when one of the nested filesets points to a directory that doesn't exist.	No; defaults to true.
verbose	Log the files that are being copied.	No; defaults to false.
granularity	The number of milliseconds leeway to give before deciding a file is out of date. This is needed because not every file system supports tracking the last modified time to the millisecond level. Default is 0 milliseconds, or 2 seconds on DOS systems. This can also be useful if source and target files live on separate machines with clocks being out of sync. *since Ant 1.6.2*.	No.

Parameters specified as nested elements

fileset or any other resource collection

Resource Collections are used to select groups of files to copy. To use a resource collection, the `todir` attribute must be set.

Prior to Ant 1.7 only `<fileset>` has been supported as a nested element.

preserveInTarget

Since Ant 1.7.0

Specifies files or directories that should be kept in the target directory even if they are not present in one of the source directories.

This nested element is like a FileSet except that it doesn't support the dir attribute and the usedefaultexcludes attribute defaults to false.

Additional Parameters

Attribute	Description	Required
preserveEmptyDirs	Overrules the includeEmptydirs setting for directories matched by this element. If you want to preserve empty directories that are not in your source directory you can either set the task's includeemptydirs attribute or this one. If the two attribute values conflict, this attribute "wins".	No, defaults to the value of the task's includeemptydirs attribute

Examples

```
<sync todir="site">
  <fileset dir="generated-site"/>
</sync>
```

overwrites all files in *site* with newer files from *generated-site*, deletes files from *site* that are not present in *generated-site*.

```
<sync todir="site">
  <fileset dir="generated-site"/>
  <preserveintarget>
    <include name="**/CVS/**"/>
  </preserveintarget>
</sync>
```

overwrites all files in *site* with newer files from *generated-site*, deletes files from *site* that are not present in *generated-site* but keeps all files in any *CVS* sub-directory.

Tempfile Task

This task sets a property to the name of a temporary file.

Description

This task sets a property to the name of a temporary file. Unlike `java.io.File.createTempFile`, this task does not actually create the temporary file, but it does guarantee that the file did not exist when the task was executed.

Examples:

```
<tempfile property="temp.file"/>
```

create a temporary file

```
<tempfile property="temp.file" suffix=".xml"/>
```

create a temporary file with the `.xml` suffix

```
<tempfile property="temp.file" destDir="build"/>
```

create a temporary file in the `build` subdirectory

Parameters

Attribute	Description	Type	Requirement
property	Sets the property you wish to assign the temporary file to.	String	Required
destdir	Sets the destination directory. If not set, the basedir directory is used instead.	File	Optional
prefix	Sets the optional prefix string for the temp file.	String	
suffix	Sets the optional suffix string for the temp file.	String	
deleteonexit	Whether the temp file will be marked for deletion on normal exit of the Java Virtual Machine (even though the file may never be created); default *false*. Since Ant 1.7	boolean	
createfile	Whether the temp file should be created by this task; default *false*.Since Ant 1.8	boolean	

Parameters as nested elements

Touch

Description

Changes the modification time of a resource and possibly creates it at the same time. In addition to working with a single file, this Task can also work on resources and resource collections (which also includes directories). Prior to Ant 1.7 only FileSet or Filelist (since Ant 1.6) have been supported.

Parameters

Attribute	Description	Required
file	The name of the file.	Unless a nested resource collection element has been specified.
millis	Specifies the new modification time of the file in milliseconds since midnight Jan 1 1970.	No--datetime takes precedence, however if both are omitted the current time is assumed.
datetime	Specifies the new modification time of the file. The special value "now" indicates the current time (now supported since Ant 1.8).	
pattern	SimpleDateFormat-compatible pattern string. Defaults to MM/DD/YYYY HH:MM AM_or_PM or MM/DD/YYYY HH:MM:SS AM_or_PM. **Since Ant 1.6.3**	No
mkdirs	Whether to create nonexistent parent directories when touching new files. **Since Ant 1.6.3**	No, default *false*.
verbose	Whether to log the creation of new files. **Since Ant 1.6.3**	No, default *true*.

Parameters specified as nested elements

any resource collection

You can use any number of nested resource collection elements to define the resources for this task and refer to resources defined elsewhere. **Note:** resources passed to this task must implement the `org.apache.tools.ant.types.resources.Touchable` interface, this is true for all filesystem-based resources like those returned by path, fileset ot filelist.

For backwards compatibility directories matched by nested filesets will be "touched" as well, use a <type> selector to suppress this. This only applies to filesets nested into the task directly, not to filesets nested into a path or any other resource collection.

mapper

Since Ant 1.6.3, a nested mapper can be specified. Files specified via nested `filesets`, `filelists`, or the `file` attribute are mapped using the specified mapper. For each file mapped, the resulting files are touched. If no time has been specified and the original file exists its timestamp will be used. If no time has been specified and the original file does not exist the current time is used. Since Ant 1.8 the task settings (`millis`, and `datetime`) have priority over the timestamp of the original file.

Examples

```
  <touch file="myfile"/>
```

creates `myfile` if it doesn't exist and changes the modification time to the current time.

```
  <touch file="myfile" datetime="06/28/2000 2:02 pm"/>
```

creates `myfile` if it doesn't exist and changes the modification time to Jun, 28 2000 2:02 pm (14:02 for those used to 24 hour times).

```
  <touch datetime="09/10/1974 4:30 pm">
    <fileset dir="src_dir"/>
  </touch>
```

changes the modification time to Oct, 09 1974 4:30 pm of all files and directories found in `src_dir`.

```
  <touch file="myfile" datetime="06/28/2000 2:02:17 pm"/>
```

creates `myfile` if it doesn't exist and changes the modification time to Jun, 28 2000 2:02:17 pm (14:02:17 for those used to 24 hour times), if the filesystem allows a precision of one second - a time close to it otherwise.

```
  <touch file="foo">
    <mapper type="glob" from="foo" to="bar" />
  </touch>
```

creates `bar` if it doesn't exist and changes the modification time to that of `foo`.

```
  <touch file="foo" datetime="now">
    <mapper type="regexp" from="^src(.*)\.java" to="shadow\1.empty" />
  </touch>
```

creates files in the `shadow` directory for every java file in the `src` directory if it doesn't exist and changes the modification time of those files to the current time.

jarlib-available

Description

Check whether an extension is present in a fileset or an extensionSet. If the extension is present then a property is set.

Note that this task works with extensions as defined by the "Optional Package" specification. For more information about optional packages, see the document *Optional Package Versioning* in the documentation bundle for your Java2 Standard Edition package, in file `guide/extensions/versioning.html` or online at http://java.sun.com/j2se/1.3/docs/guide/extensions/versioning.html.

See the Extension and ExtensionSet documentation for further details

Parameters

Attribute	Description	Required
property	The name of property to set if extensions is available.	Yes
file	The file to check for extension	No, one of file, nested ExtensionSet or nested fileset must be present.

Parameters specified as nested elements

extension

Extension the extension to search for.

fileset

FileSets are used to select sets of files to check for extension.

extensionSet

ExtensionSets is the set of extensions to search for extension in.

Examples

Search for extension in single file

```
<jarlib-available property="myext.present" file="myfile.jar">
  <extension
    extensionName="org.apache.tools.ant"
    specificationVersion="1.4.9"
    specificationVendor="Apache Software Foundation"/>
</jarlib-available>
```

Search for extension in single file referencing external Extension

```
<extension id="myext"
  extensionName="org.apache.tools.ant"
  specificationVersion="1.4.9"
```

```
    specificationVendor="Apache Software Foundation"/>

  <jarlib-available property="myext.present" file="myfile.jar">
    <extension refid="myext"/>
  </jarlib-available>
```

Search for extension in fileset

```
  <extension id="myext"
    extensionName="org.apache.tools.ant"
    specificationVersion="1.4.9"
    specificationVendor="Apache Software Foundation"/>

  <jarlib-available property="myext.present">
    <extension refid="myext"/>
    <fileset dir="lib">
      <include name="*.jar"/>
    </fileset>
  </jarlib-available>
```

Search for extension in extensionSet

```
  <extension id="myext"
    extensionName="org.apache.tools.ant"
    specificationVersion="1.4.9"
    specificationVendor="Apache Software Foundation"/>

  <jarlib-available property="myext.present">
    <extension refid="myext"/>
    <extensionSet id="exts3">
      <libfileset
        includeUrl="false"
        includeImpl="true"
        dir="lib">
        <include name="*.jar"/>
      </libfileset>
    </extensionSet>
  </jarlib-available>
```

jarlib-display

Description

Display the "Optional Package" and "Package Specification" information contained within the specified jars.

Note that this task works with extensions as defined by the "Optional Package" specification. For more information about optional packages, see the document *Optional Package Versioning* in the documentation bundle for your Java2 Standard Edition package, in file `guide/extensions/versioning.html` or online at http://java.sun.com/j2se/1.3/docs/guide/extensions/versioning.html.

See the Extension and ExtensionSet documentation for further details

Parameters

Attribute	Description	Required
file	The file to display extension information about.	No, but one of file or fileset must be present.

Parameters specified as nested elements

fileset

FileSets contain list of files to display Extension information about.

Examples

Display Extension info for a single file

```
<jarlib-display file="myfile.jar">
```

Display Extension info for a fileset

```
<jarlib-display>
  <fileset dir="lib">
    <include name="*.jar"/>
  </fileset>
</jarlib-display>
```

jarlib-manifest

Description

Task to generate a manifest that declares all the dependencies in manifest. The dependencies are determined by looking in the specified path and searching for Extension / "Optional Package" specifications in the manifests of the jars.

Note that this task works with extensions as defined by the "Optional Package" specification. For more information about optional packages, see the document *Optional Package Versioning* in the documentation bundle for your Java2 Standard Edition package, in file `guide/extensions/versioning.html` or online at http://java.sun.com/j2se/1.3/docs/guide/extensions/versioning.html.

See the Extension and ExtensionSet documentation for further details

Parameters

Attribute	Description	Required
destfile	The file to generate Manifest into	Yes.

Parameters specified as nested elements

extension

Extension the extension that this library implements.

depends

ExtensionSets containing all dependencies for jar.

options

ExtensionSets containing all optional dependencies for jar. (Optional dependencies will be used if present else they will be ignored)

Examples

Basic Manifest generated for single Extension

```
<extension id="e1"
    extensionName="MyExtensions"
    specificationVersion="1.0"
    specificationVendor="Peter Donald"
    implementationVendorID="vv"
    implementationVendor="Apache"
    implementationVersion="2.0"
    implementationURL="http://somewhere.com"/>

<jarlib-manifest destfile="myManifest.txt">
    <extension refid="e1"/>
</jarlib-manifest>
```

Search for extension in fileset

A large example with required and optional dependencies

```
<extension id="e1"
    extensionName="MyExtensions"
    specificationVersion="1.0"
    specificationVendor="Peter Donald"
    implementationVendorID="vv"
    implementationVendor="Apache"
    implementationVersion="2.0"
    implementationURL="http://somewhere.com"/>

<extensionSet id="option.ext">
    <libfileset dir="lib/option">
        <include name="**/*.jar"/>
    </libfileset>
</extensionSet>

<extensionSet id="depends.ext">
    <libfileset dir="lib/required">
        <include name="*.jar"/>
    </libfileset>
</extensionSet>

<jarlib-manifest destfile="myManifest.txt">
    <extension refid="e1"/>
    <depends refid="depends.ext"/>
    <options refid="option.ext"/>
</jarlib-manifest>
```

jarlib-resolve

Description

Try to locate a jar to satisfy an extension and place location of jar into property. The task allows you to add a number of resolvers that are capable of locating a library for a specific extension. Each resolver will be attempted in specified order until library is found or no resolvers are left. If no resolvers are left and failOnError is true then a BuildException will be thrown.

Note that this task works with extensions as defined by the "Optional Package" specification. For more information about optional packages, see the document *Optional Package Versioning* in the documentation bundle for your Java2 Standard Edition package, in file `guide/extensions/versioning.html` or online at http://java.sun.com/j2se/1.3/docs/guide/extensions/versioning.html.

See the Extension and ExtensionSet documentation for further details

Parameters

Attribute	Description	Required
property	The name of property to set to library location.	Yes
failOnError	True if failure to locate library should result in build exception.	No, defaults to true.
checkExtension	True if libraries returned by nested resolvers should be checked to see if they supply extension.	No, defaults to true.

Parameters specified as nested elements

extension

Extension the extension to resolve. Must be present

location

The location sub element allows you to look for a library in a location relative to project directory.

Attribute	Description	Required
location	The pathname of library.	Yes

url

The url resolver allows you to download a library from a URL to a local file.

Attribute	Description	Required
url	The URL to download.	Yes
destfile	The file to download URL into.	No, But one of destfile or destdir must be present
destdir	The directory in which to place downloaded file.	No, But one of destfile or destdir must be present

ant

The ant resolver allows you to run a ant build file to generate a library.

Attribute	Description	Required
antfile	The build file.	Yes
destfile	The file that the ant build creates.	Yes
target	The target to run in build file.	No

Examples

Resolve Extension to file. If file does not exist or file does not implement extension then throw an exception.

```
<extension id="dve.ext"
  extensionName="org.realityforge.dve"
  specificationVersion="1.2"
  specificationVendor="Peter Donald"/>

<jarlib-resolve property="dve.library">
  <extension refid="dve.ext"/>
  <location location="/opt/jars/dve.jar"/>
</jarlib-resolve>
```

Resolve Extension to url. If url does not exist or can not write to destfile or files does not implement extension then throw an exception.

```
<extension id="dve.ext"
  extensionName="org.realityforge.dve"
  specificationVersion="1.2"
  specificationVendor="Peter Donald"/>

<jarlib-resolve property="dve.library">
  <extension refid="dve.ext"/>
  <url url="http://www.realityforge.net/jars/dve.jar" destfile="lib/dve.jar"/>
</jarlib-resolve>
```

Resolve Extension to file produce by ant build. If file does not get produced or ant file is missing or build fails then throw an exception (Note does not check that library implements extension).

```
<extension id="dve.ext"
  extensionName="org.realityforge.dve"
  specificationVersion="1.2"
  specificationVendor="Peter Donald"/>

<jarlib-resolve property="dve.library" checkExtension="false">
  <extension refid="dve.ext"/>
  <ant antfile="../dve/build.xml" target="main" destfile="lib/dve.jar"/>
</jarlib-resolve>
```

Resolve Extension via multiple methods. First check local file to see if it implements extension. If it does not then try to build it from source in parallel directory. If that fails then finally try to download it from a website. If all steps fail then throw a build exception.

```
<extension id="dve.ext"
  extensionName="org.realityforge.dve"
  specificationVersion="1.2"
  specificationVendor="Peter Donald"/>

<jarlib-resolve property="dve.library">
  <extension refid="dve.ext"/>
  <location location="/opt/jars/dve.jar"/>
  <ant antfile="../dve/build.xml" target="main" destfile="lib/dve.jar"/>
  <url url="http://www.realityforge.net/jars/dve.jar" destfile="lib/dve.jar"/>
</jarlib-resolve>
```

Record

Description

A recorder is a listener to the current build process that records the output to a file.

Several recorders can exist at the same time. Each recorder is associated with a file. The filename is used as a unique identifier for the recorders. The first call to the recorder task with an unused filename will create a recorder (using the parameters provided) and add it to the listeners of the build. All subsequent calls to the recorder task using this filename will modify that recorders state (recording or not) or other properties (like logging level).

Some technical issues: the file's print stream is flushed for "finished" events (buildFinished, targetFinished and taskFinished), and is closed on a buildFinished event.

Parameters

Attribute	Description	Required
name	The name of the file this logger is associated with.	yes
action	This tells the logger what to do: should it start recording or stop? The first time that the recorder task is called for this logfile, and if this attribute is not provided, then the default for this attribute is "start". If this attribute is not provided on subsequent calls, then the state remains as previous. [Values = {start|stop}, Default = no state change]	no
append	Should the recorder append to a file, or create a new one? This is only applicable the first time this task is called for this file. [Values = {yes|no}, Default=no]	no
emacsmode	Removes `[task]` banners like Ant's `-emacs` command line switch if set to *true*.	no, default is *false*
loglevel	At what logging level should this recorder instance record to? This is not a once only parameter (like `append` is) -- you can increase or decrease the logging level as the build process continues. [Values= {error|warn|info|verbose|debug}, Default = no change]	no

Examples

The following build.xml snippet is an example of how to use the recorder to record just the `<javac>` task:

```
...
<compile >
    <record name="log.txt" action="start"/>
    <javac ...
    <record name="log.txt" action="stop"/>
<compile/>
...
```

The following two calls to `<record>` set up two recorders: one to file "records-simple.log" at logging level `info` (the default) and one to file "ISO.log" using logging level of `verbose`.

```
...
<record name="records-simple.log"/>
<record name="ISO.log" loglevel="verbose"/>
...
```

Notes

There is some functionality that I would like to be able to add in the future. They include things like the following:

Attribute	Description	Required
listener	A classname of a build listener to use from this point on instead of the default listener.	no
includetarget	A comma-separated list of targets to automatically record. If this value is "all", then all targets are recorded. [Default = all]	no
excludetarget		no
includetask	A comma-separated list of task to automatically record or not. This could be difficult as it could conflict with the `includetarget/excludetarget`. (e.g.: `includetarget="compile" exlcudetask="javac"`, what should happen?)	no
excludetask		no
action	add greater flexibility to the action attribute. Things like `close` to close the print stream.	no

Mail

Description

A task to send SMTP email.

This task can send mail using either plain text, UU encoding, or MIME format mail, depending on what is available.

SMTP auth and SSL/TLS require JavaMail and are only available in MIME format.

Attachments may be sent using nested `<attachments>` elements, which are path-like structures. This means any filesystem based resource or resource collection can be used to point to attachments. Prior to Ant 1.7 only `<fileset>` has been supported as a nested element, you can still use this directly without an `<attachments>` container.

Note: This task may depend on external libraries that are not included in the Ant distribution. See Library Dependencies for more information.

Parameters

Attribute	Description	Required
from	Email address of sender.	Either a `from` attribute, or a `<from>` element.
replyto	Replyto email address.	No
tolist	Comma-separated list of recipients.	At least one of these, or the equivalent elements.
cclist	Comma-separated list of recipients to carbon copy	
bcclist	Comma-separated list of recipients to blind carbon copy	
message	Message to send in the body of the email.	One of these or a `<message>` element.
messagefile	File to send as the body of the email. Property values in the file will be expanded.	
messagemimetype	The content type of the message. The default is `text/plain`.	No
files	Files to send as attachments to the email. Separate multiple file names using a comma or space. You can also use `<fileset>` elements to specify files.	No
failonerror	flag to indicate whether to halt the build on any error. The default value is `true`.	No.
includefilenames	Include filename(s) before file contents. Valid only when the `plain` encoding is used. The default value is `false`.	No
mailhost	Host name of the SMTP server. The default value is `localhost`.	No
mailport	TCP port of the SMTP server. The default value is 25.	No
user	user name for SMTP auth	Yes, if SMTP auth is required on your SMTP server the email message will be then sent using Mime and requires JavaMail

password	password for SMTP auth	Yes, if SMTP auth is required on your SMTP server the email message will be then sent using Mime and requires JavaMail
ssl	"true", "on" or "yes" accepted here indicates whether you need TLS/SSL	No
encoding	Specifies the encoding to use for the content of the email. Values are `mime`, `uu`, `plain`, or `auto`. The default value is `auto`. `uu` or `plain` are not compatible with SMTP auth	No
charset	Character set of the email. You can also set the charset in the message nested element. These options are mutually exclusive.	No
subject	Email subject line.	No
ignoreInvalidRecipients	Boolean. Whether the task should try to send the message to as many recipients as possible and should only fail if neither is reachable. *Since Ant 1.8.0*.	No, default is false
enableStartTLS	"true", "on" or "yes" accepted here whether the STARTTLS command used to switch to an encrypted connection for authentication should be supported. Requires JavaMail. *Since Ant 1.8.0*	No

Note regarding the attributes containing email addresses

Since Ant 1.6, the attributes from, replyto, tolist, cclist, bcclist can contain email addresses of the form :

- address@xyz.com
- name <address@xyz.com>
- <address@xyz.com> name
- (name) address@xyz.com
- address@xyz.com (name)

You need to enter the angle brackets as XML entities `>` and `<`.

Parameters specified as nested elements

to / cc / bcc / from/ replyto

Adds an email address element. It takes the following attributes:

Attribute	Description	Required
name	The display name for the address.	No
address	The email address.	Yes

message

Specifies the message to include in the email body. It takes the following attributes:

Attribute	Description	Required
src	The file to use as the message.	No
mimetype	The content type to use for the message.	No
charset	Character set of the message You can also set the charset as attribute of the enclosing mail task. These options are mutually exclusive.	No

If the `src` attribute is not specified, then text can be added inside the `<message>` element. Property expansion will occur in the message, whether it is specified as an external file or as text within the `<message>` element.

header

Since Ant 1.7, arbitrary mail headers can be added by specifying these attributes on one or more nested header elements:

Attribute	Description	Required
name	The name associated with this mail header.	Yes
value	The value to assign to this mail header.	Yes

It is permissible to duplicate the name attribute amongst multiple headers.

Examples

```
<mail from="me"
      tolist="you"
      subject="Results of nightly build"
      files="build.log"/>
```

Sends an email from *me* to *you* with a subject of *Results of nightly build* and includes the contents of the file *build.log* in the body of the message.

```
<mail mailhost="smtp.myisp.com" mailport="1025" subject="Test build">
  <from address="config@myisp.com"/>
  <replyto address="me@myisp.com"/>
  <to address="all@xyz.com"/>
  <message>The ${buildname} nightly build has completed</message>
  <attachments>
    <fileset dir="dist">
      <include name="**/*.zip"/>
    </fileset>
  </attachments>
</mail>
```

Sends an eMail from *config@myisp.com* to *all@xyz.com* with a subject of *Test Build*. Replies to this email will go to *me@myisp.com*. Any zip files from the dist directory are attached. The task will attempt to use JavaMail and fall back to UU encoding or no encoding in that order depending on what support classes are available. `${buildname}` will be replaced with the `buildname` property's value.

```
<property name="line2" value="some_international_message"/>
<echo message="${line2}"/>

<mail mailhost="somehost@xyz.com" mailport="25" subject="Test build"  charset="utf-8">
  <from address="me@myist.com"/>
  <to address="all@xyz.com"/>
  <message>some international text:${line2}</message>
```

```
</mail>
```

Sends an eMail from *me@myisp.com* to *all@xyz.com* with a subject of *Test Build*, the message body being coded in UTF-8

MimeMail

Deprecated

This task has been deprecated. Use the mail task instead.

Description

Sends SMTP mail with MIME attachments. JavaMail and Java Activation Framework are required for this task.

Multiple files can be attached using FileSets.

Parameters

Attribute	Description	Required
message	The message body	No, but only one of of 'message' or 'messageFile' may be specified. If not specified, a fileset must be provided.
messageFile	A filename to read and used as the message body	
messageMimeType	MIME type to use for 'message' or 'messageFile' when attached.	No, defaults to "text/plain"
tolist	Comma-separated list of To: recipients	Yes, at least one of 'tolist', 'cclist', or 'bcclist' must be specified.
cclist	Comma-separated list of CC: recipients	
bcclist	Comma-separated list of BCC: recipients	
mailhost	Host name of the mail server.	No, default to "localhost"
subject	Email subject line.	No
from	Email address of sender.	Yes
failonerror	Stop the build process if an error occurs sending the e-mail.	No, default to "true"

Examples

Send a single HTML file as the body of a message

```
    <mimemail messageMimeType="text/html" messageFile="overview-summary.html"
        tolist="you" subject="JUnit Test Results: ${TODAY}" from="me"/>
```

Sends all files in a directory as attachments

```
    <mimemail message="See attached files" tolist="you" subject="Attachments" from="me">
        <fileset dir=".">
            <include name="dist/*.*"/>
        </fileset>
    </mimemail>
```

DefaultExcludes

since Ant 1.6

Description

Alters the default excludes for all subsequent processing in the build, and prints out the current default excludes if desired.

Parameters

Attribute	Description	Required
echo	whether or not to print out the default excludes.(defaults to false)	attribute "true" required if no other attribute specified
default	go back to hard wired default excludes	attribute "true" required if no if no other attribute is specified
add	the pattern to add to the default excludes	if no other attribute is specified
remove	remove the specified pattern from the default excludes	if no other attribute is specified

Examples

Print out the default excludes

```
<defaultexcludes echo="true"/>
```

Print out the default excludes and exclude all *.bak files in **all** further processing

```
<defaultexcludes echo="true" add="**/*.bak"/>
```

Silently allow several fileset based tasks to operate on emacs backup files and then restore normal behavior

```
<defaultexcludes remove="**/*~"/>

(do several fileset based tasks here)

<defaultexcludes default="true"/>
```

Notes

By default the pattern `**/.svn` and `**/.svn/**` are set as default excludes. With version 1.3 Subversion supports the "_svn hack". That means, that the svn-libraries evaluate environment variables and use *.svn* or *_svn* directory regarding to that value. We had chosen not to evaluate environment variables to get a more reliable build. Instead you have to change the settings by yourself by changing the exclude patterns:

```
<defaultexcludes remove="**/.svn"/>
<defaultexcludes remove="**/.svn/**"/>
<defaultexcludes add="**/_svn"/>
<defaultexcludes add="**/_svn/**"/>
```

Echo

Description

Echoes a message to the current loggers and listeners which means `System.out` unless overridden. A `level` can be specified, which controls at what logging level the message is filtered at.

The task can also echo to a file, in which case the option to append rather than overwrite the file is available, and the `level` option is ignored

Parameters

Attribute	Description	Required
message	the message to echo.	No. Text may also be included in a character section within this element. If neither is included a blank line will be emitted in the output.
file	the file to write the message to.	Optionally one of these may be specified.
output	the Resource to write the message to (see note). **Since Ant 1.8**	
append	Append to an existing file (or open a new file / overwrite an existing file)? Default *false*.	No; ignored unless *output* indicates a filesystem destination.
level	Control the level at which this message is reported. One of "error", "warning", "info", "verbose", "debug" (decreasing order)	No - default is "warning".
encoding	encoding to use, default is ""; the local system encoding. *since Ant 1.7*	No
force	Overwrite read-only destination files. *since Ant 1.8.2*	No; defaults to false.

Examples

```
<echo message="Hello, world"/>
```

```
<echo message="Embed a line break:${line.separator}"/>
```

```
<echo>Embed another:${line.separator}</echo>
```

```
<echo>This is a longer message stretching over
two lines.
</echo>
```

```
<echo>
This is a longer message stretching over
three lines; the first line is a blank
</echo>
```

The newline immediately following the <echo> tag will be part of the output.
Newlines in character data within the content of an element are not discarded by XML parsers.
See W3C Recommendation 04 February 2004 / End of Line handling for more details.

```
<echo message="Deleting drive C:" level="debug"/>
```

A message which only appears in `-debug` mode.

```
<echo level="error">
Imminent failure in the antimatter containment facility.
Please withdraw to safe location at least 50km away.
</echo>
```

A message which appears even in `-quiet` mode.

```
<echo file="runner.csh" append="false">#\!/bin/tcsh
java-1.3.1 -mx1024m ${project.entrypoint} $$*
</echo>
```

Generate a shell script by echoing to a file. Note the use of a double $ symbol to stop Ant filtering out the single $ during variable expansion

Depending on the loglevel Ant runs, messages are print out or silently ignored:

Ant-Statement	-quiet, -q	*no statement*	-verbose, -v	-debug, -d
`<echo message="This is error message." level="error" />`	ok	ok	ok	ok
`<echo message="This is warning message." />`	ok	ok	ok	ok
`<echo message="This is warning message." level="warning" />`	ok	ok	ok	ok
`<echo message="This is info message." level="info" />`	not logged	ok	ok	ok
`<echo message="This is verbose message." level="verbose" />`	not logged	not logged	ok	ok
`<echo message="This is debug message." level="debug" />`	not logged	not logged	not logged	ok

Fail

Description

Exits the current build (just throwing a BuildException), optionally printing additional information.

The message of the Exception can be set via the message attribute or character data nested into the element.

Parameters

Attribute	Description	Required
message	A message giving further information on why the build exited	No
if	Only fail if a property of the given name exists in the current project	No
unless	Only fail if a property of the given name doesn't exist in the current project	No
status	Exit using the specified status code; assuming the generated Exception is not caught, the JVM will exit with this status. *Since Ant 1.6.2*	No

Parameters specified as nested elements

As an alternative to the *if*/*unless* attributes, conditional failure can be achieved using a single nested `<condition>` element, which should contain exactly one core or custom condition. For information about conditions, see here.
Since Ant 1.6.2

Examples

```
  <fail/>
```

will exit the current build with no further information given.

```
BUILD FAILED
build.xml:4: No message
```

```
  <fail message="Something wrong here."/>
```

will exit the current build and print something like the following to wherever your output goes:

```
BUILD FAILED
build.xml:4: Something wrong here.
```

```
  <fail>Something wrong here.</fail>
```

will give the same result as above.

```
  <fail unless="thisdoesnotexist"/>
```

will exit the current build and print something like the following to wherever your output goes:

```
BUILD FAILED
build.xml:2: unless=thisdoesnotexist
```

Using a condition to achieve the same effect:

```
  <fail>
     <condition>
       <not>
         <isset property="thisdoesnotexist"/>
       </not>
     </condition>
   </fail>
```

Output:

```
BUILD FAILED

build.xml:2: condition satisfied
```

```
<fail message="Files are missing.">
    <condition>
        <not>
            <resourcecount count="2">
                <fileset id="fs" dir="." includes="one.txt,two.txt"/>
            </resourcecount>
        </not>
    </condition>
</fail>
```

Will check that both files *one.txt* and *two.txt* are present otherwise the build will fail.

GenKey

Description

Generates a key in a keystore.

Parameters

Attribute	Description	Required
alias	the alias to add under	Yes.
storepass	password for keystore integrity. Must be at least 6 characters long	Yes.
keystore	keystore location	No
storetype	keystore type	No
keypass	password for private key (if different)	No
sigalg	the algorithm to use in signing	No
keyalg	the method to use when generating name-value pair	No
verbose	(true | false) verbose output when signing	No
dname	The distinguished name for entity	Yes if dname element unspecified
validity	(integer) indicates how many days certificate is valid	No
keysize	(integer) indicates the size of key generated	No

Alternatively you can specify the distinguished name by creating a sub-element named dname and populating it with param elements that have a name and a value. When using the subelement it is automatically encoded properly and commas (",") are replaced with "\,".

The following two examples are identical:

Examples

```
<genkey alias="apache-group" storepass="secret"
  dname="CN=Ant Group, OU=Jakarta Division, O=Apache.org, C=US"/>
```

```
<genkey alias="apache-group" storepass="secret" >
  <dname>
    <param name="CN" value="Ant Group"/>
    <param name="OU" value="Jakarta Division"/>
    <param name="O"  value="Apache.Org"/>
    <param name="C"  value="US"/>
  </dname>
</genkey>
```

HostInfo

Description

Sets the `NAME`, `DOMAIN`, `ADDR4`, and `ADDR6` properties in the current project.

The `NAME` contains the host part of the canonical name of the host.
If the host is not found, the host will contain the name as provided to the task, or `localhost` if no host was provided, and no name for the local host was found.
The `DOMAIN` contains the domain part of the canonical name of the host.
If the host is not found, the domain will contain the domain as provided to the task, or `localdomain` if no host / domain was provided.
The `ADDR4` contains the IPv4 address of the host with the widest meaning.
If no IPv4 address is found and a host has been provided the address `0.0.0.0` is returned, when no host was provided the address `127.0.0.1` is returned.
The `ADDR6` contains the IPv6 address of the host with the widest meaning.
If no IPv6 address is found and a host has been provided the address `::` is returned, when no host was provided the address `::1` is returned.

These properties can be used in the build-file, for instance, to create host-stamped filenames, or used to replace placeholder tags inside documents to indicate, for example, the host where the build was performed on. The best place for this task is probably in an initialization target.

Parameters

Attribute	Description	Required
prefix	Prefix used for all properties set. The default is no prefix.	No
host	The host to retrieve the information for, default is to retrieve information for the host the task is running on.	No

Examples

```
<hostinfo/>
```

Sets the `NAME`, `DOMAIN`, `ADDR4`, and `ADDR6` for the local host, using the most "global" address available.

```
<hostinfo prefix="remotehost" host="www.apache.org"/>
```

Sets the properties `remotehost.NAME` to `eos`, `remotehost.DOMAIN` to `apache.org`, `remotehost.ADDR4` to `140.211.11.130` and `remotehost.ADDR6` to `::` for the host with the name www.apache.org (provided the canonical name and ip addresses do not change).

Input

Description

Allows user interaction during the build process by prompting for input. To do so, it uses the configured InputHandler.

The prompt can be set via the message attribute or as character data nested into the element.

Optionally a set of valid input arguments can be defined via the validargs attribute. Input task will not accept a value that doesn't match one of the predefined.

Optionally a property can be created from the value entered by the user. This property can then be used during the following build run. Input behaves according to property task which means that existing properties cannot be overriden. Since Ant 1.6, `<input>` will not prompt for input if a property should be set by the task that has already been set in the project (and the task wouldn't have any effect).

Historically, a regular complaint about this task has been that it echoes characters to the console, this is a critical security defect, we must fix it immediately, etc, etc. This problem was due to the lack in early versions of Java of a (fully functional) facility for handling secure console input. In Java 1.6 that shortcoming in Java's API was addressed and Ant versions 1.7.1 and 1.8 have added support for Java 1.6's secure console input feature (see handler type).

IDE behaviour depends upon the IDE: some hang waiting for input, some let you type it in. For this situation, place the password in a (secured) property file and load in before the input task.

Parameters

Attribute	Description	Required
message	the Message which gets displayed to the user during the build run.	No
validargs	comma separated String containing valid input arguments. If set, input task will reject any input not defined here. Validargs are compared case sensitive. If you want 'a' and 'A' to be accepted you will need to define both arguments within validargs.	No
addproperty	the name of a property to be created from input. Behaviour is equal to property task which means that existing properties cannot be overridden.	No
defaultvalue	Defines the default value of the property to be created from input. Property value will be set to default if no input is received.	No

Parameters Specified as Nested Elements

Handler

Since **Ant 1.7**, a nested <handler> element can be used to specify an InputHandler, so that different InputHandlers may be used among different Input tasks.

Attribute	Description	Required
type	one of "default","propertyfile", "greedy", or "secure" (since Ant 1.8).	One of these
refid	Reference to an `InputHandler` defined elsewhere in the project.	
classname	The name of an `InputHandler` subclass.	
classpath	The classpath to use with *classname*.	No

classpathref	The refid of a classpath to use with *classname*.	No
loaderref	The refid of a classloader to use with *classname*.	No

The classpath can also be specified by means of one or more nested <classpath> elements.

Examples

```
<input/>
```

Will pause the build run until return key is pressed when using the default InputHandler, the concrete behavior is defined by the InputHandler implementation you use.

```
<input>Press Return key to continue...</input>
```

Will display the message "Press Return key to continue..." and pause the build run until return key is pressed (again, the concrete behavior is implementation dependent).

```
<input
  message="Press Return key to continue..."
/>
```

Will display the message "Press Return key to continue..." and pause the build run until return key is pressed (see above).

```
<input
  message="All data is going to be deleted from DB continue (y/n)?"
  validargs="y,n"
  addproperty="do.delete"
/>
<condition property="do.abort">
  <equals arg1="n" arg2="${do.delete}"/>
</condition>
<fail if="do.abort">Build aborted by user.</fail>
```

Will display the message "All data is going to be deleted from DB continue (y/n)?" and require 'y' to continue build or 'n' to exit build with following message "Build aborted by user.".

```
<input
  message="Please enter db-username:"
  addproperty="db.user"
/>
```

Will display the message "Please enter db-username:" and set the property `db.user` to the value entered by the user.

```
<input
  message="Please enter db-username:"
  addproperty="db.user"
  defaultvalue="Scott-Tiger"
/>
```

Same as above, but will set `db.user` to the value *Scott- Tiger* if the user enters no value (simply types <return>).

Script

Description

Execute a script in a Apache BSF or JSR 223 supported language.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information.

The task may use the BSF scripting manager or the JSR 223 manager that is included in JDK6 and higher. This is controlled by the `manager` attribute. The JSR 223 scripting manager is indicated by "javax".

All items (tasks, targets, etc) of the running project are accessible from the script, using either their `name` or `id` attributes (as long as their names are considered valid Java identifiers, that is). This is controlled by the "setbeans" attribute of the task. The name "project" is a pre-defined reference to the Project, which can be used instead of the project name. The name "self" is a pre-defined reference to the actual `<script>`-Task instance.
From these objects you have access to the Ant Java API, see the JavaDoc (especially for Project and Script) for more information.

If you are using JavaScript under BSF, a good resource is http://www.mozilla.org/rhino/doc.html as we are using their JavaScript interpreter.

Scripts can do almost anything a task written in Java could do.

Rhino provides a special construct - the *JavaAdapter*. With that you can create an object which implements several interfaces, extends classes and for which you can overwrite methods. Because this is an undocumented feature (yet), here is the link to an explanation: Groups@Google: "Rhino, enum.js, JavaAdapter?" by Norris Boyd in the newsgroup *netscape.public.mozilla.jseng*.

If you are creating Targets programmatically, make sure you set the Location to a useful value. In particular all targets should have different location values.

Parameters

Attribute	Description	Required
language	The programming language the script is written in. Must be a supported Apache BSF or JSR 223 language	Yes
manager	*Since: Ant 1.7*. The script engine manager to use. This can have one of three values ("auto", "bsf" or "javax"). The default value is "auto". • "bsf" use the BSF scripting manager to run the language. • "javax" use the *javax.scripting* manager to run the language. (This will only work for JDK6 and higher). • "auto" use the BSF engine if it exists, otherwise use the *javax.scripting* manager.	No
src	The location of the script as a file, if not inline	No
setbeans	This attribute controls whether to set variables for all properties, references and targets in the running script. If this attribute is false, only the the "project" and "self" variables are set. If this attribute is true all the variables are set. The default value of this attribute is "true". *Since Ant 1.7*	No

classpath	The classpath to pass into the script. *Since Ant 1.7*	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere. *Since Ant 1.7*	No

Parameters specified as nested elements

classpath

Since Ant 1.7

`Script`'s `classpath` attribute is a path-like structure and can also be set via a nested `<classpath>` element.

If a classpath is set, it will be used as the current thread context classloader, and as the classloader given to the BSF manager. This means that it can be used to specify the classpath containing the language implementation for BSF or for JSR 223 managers. This can be useful if one wants to keep ${user.home}/.ant/lib free of lots of scripting language specific jar files.

NB: (Since Ant 1.7.1) This classpath *can* be used to specify the location of the BSF jar file and/or languages that have engines in the BSF jar file. This includes the javascript, jython, netrexx and jacl languages.

Examples

The following snippet shows use of five different languages:

```
    <property name="message" value="Hello world"/>

    <script language="groovy">
      println("message is " + message)
    </script>

    <script language="beanshell">
      System.out.println("message is " + message);
    </script>

    <script language="judoscript">
        println 'message is ', message
    </script>

    <script language="ruby">
        print 'message is ', $message, "\n"
    </script>

    <script language="jython">
print "message is %s" % message
    </script>
```

Note that for the *jython* example, the script contents **must** start on the first column.

Note also that for the *ruby* example, the names of the set variables are prefixed by a '$'.

The following script shows a little more complicated jruby example:

```
<script language="ruby">
  xmlfiles = Dir.new(".").entries.delete_if { |i| ! (i =~ /\.xml$/) }
  xmlfiles.sort.each { |i| $self.log(i) }
</script>
```

The same example in groovy is:

```
<script language="groovy">
  xmlfiles = new java.io.File(".").listFiles().findAll{ it =~ "\.xml$"}
  xmlfiles.sort().each { self.log(it.toString())}
</script>
```

The following example shows the use of classpath to specify the location of the beanshell jar file.

```
<script language="beanshell" setbeans="true">
  <classpath>
    <fileset dir="${user.home}/lang/beanshell" includes="*.jar" />
  </classpath>
  System.out.println("Hello world");
</script>
```

The following script uses javascript to create a number of echo tasks and execute them.

```
<project name="squares" default="main" basedir=".">

  <target name="main">

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

      for (i=1; i<=10; i++) {
        echo = squares.createTask("echo");
        echo.setMessage(i*i);
        echo.perform();
      }

    ]]> </script>

  </target>

</project>
```

generates

```
main:
1
4
9
16
25
36
49
64
81
100

BUILD SUCCESSFUL
```

Now a more complex example using the Java API and the Ant API. The goal is to list the filesizes of all files a `<fileset/>` caught.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyProject" basedir="." default="main">

  <property name="fs.dir" value="src"/>
  <property name="fs.includes" value="**/*.txt"/>
  <property name="fs.excludes" value="**/*.tmp"/>

  <target name="main">
    <script language="javascript"> <![CDATA[

      // import statements
      // importPackage(java.io);
      importClass(java.io.File);

      // Access to Ant-Properties by their names
      dir      = project.getProperty("fs.dir");
      includes = MyProject.getProperty("fs.includes");
      excludes = self.getProject()  .getProperty("fs.excludes");

      // Create a <fileset dir="" includes=""/>
      fs = project.createDataType("fileset");
      fs.setDir( new File(dir) );
      fs.setIncludes(includes);
      fs.setExcludes(excludes);

      // Get the files (array) of that fileset
      ds = fs.getDirectoryScanner(project);
```

```
      srcFiles = ds.getIncludedFiles();

      // iterate over that array
      for (i=0; i<srcFiles.length; i++) {

        // get the values via Java API
        var basedir  = fs.getDir(project);
        var filename = srcFiles[i];
        var file = new File(basedir, filename);
        var size = file.length();

        // create and use a Task via Ant API
        echo = MyProject.createTask("echo");
        echo.setMessage(filename + ": " + size + " byte");
        echo.perform();
      }
    ]]></script>
  </target>
</project>
```

We want to use the Java API. Because we don't want always typing the package signature we do an import. Rhino knows two different methods for import statements: one for packages and one for a single class. By default only the *java* packages are available, so *java.lang.System* can be directly imported with `importClass/importPackage`. For other packages you have to prefix the full classified name with *Packages*. For example Ant's *FileUtils* class can be imported with `importClass(`**`Packages`**`.org.apache.tools.ant.util.FileUtils)`
The `<script>` task populates the Project instance under the name *project*, so we can use that reference. Another way is to use its given name or getting its reference from the task itself.
The Project provides methods for accessing and setting properties, creating DataTypes and Tasks and much more.
After creating a FileSet object we initialize that by calling its set-methods. Then we can use that object like a normal Ant task (`<copy>` for example).
For getting the size of a file we instantiate a `java.io.File`. So we are using normal Java API here.
Finally we use the `<echo>` task for producing the output. The task is not executed by its execute() method, because the perform() method (implemented in Task itself) does the appropriate logging before and after invoking execute().

Here is an example of using beanshell to create an ant task. This task will add filesets and paths to a referenced path. If the path does not exist, it will be created.

```
<!--
       Define addtopath task
 -->
<script language="beanshell">
    import org.apache.tools.ant.Task;
    import org.apache.tools.ant.types.Path;
    import org.apache.tools.ant.types.FileSet;
    public class AddToPath extends Task {
        private Path path;
        public void setRefId(String id) {
            path = getProject().getReference(id);
            if (path == null) {
                path = new Path(getProject());
                getProject().addReference(id, path);
            }
        }
        public void add(Path c) {
            path.add(c);
        }
        public void add(FileSet c) {
            path.add(c);
        }
        public void execute() {
            // Do nothing
        }
    }
    project.addTaskDefinition("addtopath", AddToPath.class);
</script>
```

An example of using this task to create a path from a list of directories (using antcontrib's <for> task) follows:

```
<path id="main.path">
  <fileset dir="build/classes"/>
</path>
```

```
<ac:for param="ref" list="commons,fw,lps"
        xmlns:ac="antlib:net.sf.antcontrib">
  <sequential>
    <addtopath refid="main.path">
      <fileset dir="${dist.dir}/@{ref}/main"
               includes="**/*.jar"/>
    </addtopath>
  </sequential>
</ac:for>
```

Sound

Description

Plays a sound-file at the end of the build, according to whether the build failed or succeeded. You can specify either a specific sound-file to play, or, if a directory is specified, the `<sound>` task will randomly select a file to play. Note: At this point, the random selection is based on all the files in the directory, not just those ending in appropriate suffixes for sound-files, so be sure you only have sound-files in the directory you specify.

More precisely `<sound>` registers a hook that is triggered when the build finishes. Therefore you have to place this task as top level or inside a target which is always executed.

Unless you are running on Java 1.3 or later, you need the Java Media Framework on the classpath (javax.sound).

Nested Elements

success

Specifies the sound to be played if the build succeeded.

fail

Specifies the sound to be played if the build failed.

Nested Element Parameters

The following attributes may be used on the `<success>` and `<fail>` elements:

Attribute	Description	Required
source	the path to a sound-file directory, or the name of a specific sound-file, to be played. If this file does not exist, an error message will be logged.	Yes
loops	the number of extra times to play the sound-file; default is `0`.	No
duration	the amount of time (in milliseconds) to play the sound-file.	No

Examples

```
<target name="fun" if="fun" unless="fun.done">
  <sound>
    <success source="${user.home}/sounds/bell.wav"/>
    <fail source="${user.home}/sounds/ohno.wav" loops="2"/>
  </sound>
  <property name="fun.done" value="true"/>
</target>
```

plays the `bell.wav` sound-file if the build succeeded, or the `ohno.wav` sound-file if the build failed, three times, if the `fun` property is set to `true`. If the target is a dependency of an "initialization" target that other targets depend on, the `fun.done` property prevents the target from being executed more than once.

```
<target name="fun" if="fun" unless="fun.done">
  <sound>
    <success source="//intranet/sounds/success"/>
```

```
    <fail source="//intranet/sounds/failure"/>
  </sound>
  <property name="fun.done" value="true"/>
</target>
```

randomly selects a sound-file to play when the build succeeds or fails.

Splash

by Les Hughes (leslie.hughes@rubus.com)

Description

This task creates a splash screen. The splash screen is displayed for the duration of the build and includes a handy progress bar as well. Use in conjunction with the sound task to provide interest whilst waiting for your builds to complete...

Parameters

Attribute	Description	Required	Default
imageurl	A URL pointing to an image to display.	No	antlogo.gif from the classpath
showduration	Initial period to pause the build to show the splash in milliseconds.	No	5000 ms
progressregexp	Progress regular expression which is used to parse the output and dig out current progress. Exactly one group pattern must exists, and it represents the progress number (0-100) (i.e "Progress: (.*)%") *since Ant 1.8.0*	No	progress is increased every action and log output line
displaytext	display text presented in the splash window *since Ant 1.8.0*	No	Building ...

Deprecated properties

The following properties can be used to configure the proxy settings to retrieve an image from behind a firewall. However, the settings apply not just to this task, but to all following tasks. Therefore they are now mostly deprecated in preference to the `<setproxy>` task, that makes it clear to readers of the build exactly what is going on. We say mostly as this task's support includes proxy authentication, so you may still need to use its proxy attributes.

useproxy	Use a proxy to access imgurl. Note: Only tested on JDK 1.2.2 and above	No	None
proxy	IP or hostname of the proxy server	No	None
port	Proxy portnumber	No	None
user	User to authenticate to the proxy as.	No	None
password	Proxy password	No	None

Examples

```
<splash/>
```

Splash `images/ant_logo_large.gif` from the classpath.

```
<splash imageurl="http://jakarta.apache.org/images/jakarta-logo.gif"
           useproxy="true"
           showduration="5000"/>
```

Splashes the jakarta logo, for an initial period of 5 seconds.

Splash with controlled progress and nondefault text

```
        <target name="test_new_features">
                <echo>New features</echo>
                <splash progressRegExp="Progress: (.*)%" showduration="0"
displayText="Test text"/>
                <sleep seconds="1"/>
                <echo>Progress: 10%</echo>
                <sleep seconds="1"/>
                <echo>Progress: 20%</echo>
                <sleep seconds="1"/>
                <echo>Progress: 50%</echo>
                <sleep seconds="1"/>
                <echo>Progress: 70%</echo>
                <sleep seconds="1"/>
                <echo>Progress: 100%</echo>
                <sleep seconds="3"/>
        </target>
```

Sql

Description

Executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file using the *src* attribute or from between the enclosing SQL tags.

Multiple statements can be provided, separated by semicolons (or the defined *delimiter*). Individual lines within the statements can be commented using either --, // or REM at the start of the line.

The *autocommit* attribute specifies whether auto-commit should be turned on or off whilst executing the statements. If auto-commit is turned on each statement will be executed and committed. If it is turned off the statements will all be executed as one transaction.

The *onerror* attribute specifies how to proceed when an error occurs during the execution of one of the statements. The possible values are: **continue** execution, only show the error; **stop** execution, log the error but don't fail the task and **abort** execution and transaction and fail task.

Proxies. Some JDBC drivers (including the Oracle thin driver), use the JVM's proxy settings to route their JDBC operations to the database. Since Ant1.7, Ant running on Java1.5 or later defaults to using the proxy settings of the operating system. Accordingly, the OS proxy settings need to be valid, or Ant's proxy support disabled with `-noproxy` option.

Parameters

Attribute	Description	Required
driver	Class name of the jdbc driver	Yes
url	Database connection url	Yes
userid	Database user name	Yes
password	Database password	Yes
src	File containing SQL statements	Yes, unless statements enclosed within tags
encoding	The encoding of the files containing SQL statements	No - defaults to default JVM encoding
delimiter	String that separates SQL statements	No, default ";"
autocommit	Auto commit flag for database connection (default false)	No, default "false"
print	Print result sets from the statements (default false)	No, default "false"
showheaders	Print headers for result sets from the statements (default true)	No, default "true"
showtrailers	Print trailer for number of rows affected (default true)	No, default "true"
output	Output file for result sets (defaults to System.out) **Since Ant 1.8** can specify	No (print to

	any Resource that supports output (see note).	System.out by default)
append	whether output should be appended to or overwrite an existing file. Defaults to false.	No, ignored if *output* does not specify a filesystem destination.
classpath	Classpath used to load driver	No (use system classpath)
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No (use system classpath)
onerror	Action to perform when statement fails: continue, stop, abort	No, default "abort"
rdbms	Execute task only if this rdbms	No (no restriction)
version	Execute task only if rdbms version match	No (no restriction)
caching	Should the task cache loaders and the driver?	No (default=true)
delimitertype	Control whether the delimiter will only be recognized on a line by itself. Can be "normal" -anywhere on the line, or "row", meaning it must be on a line by itself	No (default:normal)
keepformat	Control whether the format of the sql will be preserved. Useful when loading packages and procedures.	No (default=false)
escapeprocessing	Control whether the Java statement object will perform escape substitution. See Statement's API docs for details. *Since Ant 1.6.*	No (default=true)
expandproperties	Set to true to turn on property expansion in nested SQL, inline in the task or nested transactions. *Since Ant 1.7.*	No (default=true)
rawblobs	If true, will write raw streams rather than hex encoding when printing BLOB results. *Since Ant 1.7.1.*	No, default *false*
failOnConnectionError	If false, will only print a warning message and not execute any statement if the task fails to connect to the database. *Since Ant 1.8.0.*	No, default *true*
strictDelimiterMatching	If false, delimiters will be searched for in a case-insesitive manner (i.e. delimer="go" matches "GO") and surrounding whitespace will be ignored (delimter="go" matches "GO "). *Since Ant 1.8.0.*	No, default *true*
showWarnings	If true, SQLWarnings will be logged at the WARN level. *Since Ant 1.8.0.* **Note:** even if the attribute is set to false, warnings that apply to the connection will be logged at the verbose level.	No, default *false*
treatWarningsAsErrors	If true, SQLWarnings will be treated like errors - and the logic selected via the onError attribute applies. *Since Ant 1.8.0.*	No, default *false*
csvColumnSeparator	The column separator used when printing the results. *Since Ant 1.8.0.*	No, default *','*
csvQuoteCharacter	The character used to quote column values. If set, columns that contain either the column separator or the quote character itself will be surrounded by the quote character. The quote character itself will be doubled if it appears inside of the column's value. **Note:** BLOB values will never be quoted. *Since Ant 1.8.0.*	No, default is not set (i.e. no quoting ever occurs)
errorproperty	The name of a property to set in the event of an error. *Since Ant 1.8.0*	No

warningproperty	The name of a property to set in the event of an warning. *Since Ant 1.8.0*	No
rowcountproperty	The name of a property to set to the number of rows updated by the first statement/transaction that actually returned a row count. *Since Ant 1.8.0*	No

Parameters specified as nested elements

transaction

Use nested `<transaction>` elements to specify multiple blocks of commands to the executed executed in the same connection but different transactions. This is particularly useful when there are multiple files to execute on the same schema.

Attribute	Description	Required
src	File containing SQL statements	Yes, unless statements enclosed within tags

The `<transaction>` element supports any resource or single element resource collection as nested element to specify the resource containing the SQL statements.

any resource or resource collection

You can specify multiple sources via nested resource collection elements. Each resource of the collection will be run in a transaction of its own. Prior to Ant 1.7 only filesets were supported. Use a sort resource collection to get a predictable order of transactions.

classpath

`Sql`'s *classpath* attribute is a PATH like structure and can also be set via a nested *classpath* element. It is used to load the JDBC classes.

connectionProperty

Since Ant 1.8.0

Use nested `<connectionProperty>` elements to specify additional JDBC properties that need to be set when connecting to the database.

Attribute	Description	Required
name	Name of the property	Yes
value	Value of the property	Yes

Examples

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    src="data.sql"
/>
```

Connects to the database given in *url* as the sa user using the org.database.jdbcDriver and executes the SQL statements contained within the file data.sql

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    src="data.sql">
  <connectionProperty name="internal_logon" value="SYSDBA">
</sql>
```

Connects to the database given in *url* as the sa user using the org.database.jdbcDriver and executes the SQL statements contained within the file data.sql. Also sets the property *internal_logon* to the value *SYSDBA*.

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    >
insert
into table some_table
values(1,2,3,4);

truncate table some_other_table;
</sql>
```

Connects to the database given in *url* as the sa user using the org.database.jdbcDriver and executes the two SQL statements inserting data into some_table and truncating some_other_table. Ant Properties in the nested text will not be expanded.

Note that you may want to enclose your statements in `<![CDATA[ ... ]]>` sections so you don't need to escape `<`, `>` `&` or other special characters. For example:

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    ><![CDATA[

update some_table set column1 = column1 + 1 where column2 < 42;

]]></sql>
```

The following command turns property expansion in nested text on (it is off purely for backwards compatibility), then creates a new user in the HSQLDB database using Ant properties.

```
<sql
    driver="org.hsqldb.jdbcDriver";
    url="jdbc:hsqldb:file:${database.dir}"
    userid="sa"
    password=""
    expandProperties="true"
    >
  <transaction>
    CREATE USER ${newuser} PASSWORD ${newpassword}
  </transaction>
</sql>
```

The following connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the SQL statements contained within the files data1.sql, data2.sql and data3.sql and then executes the truncate operation on *some_other_table*.

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass" >
  <transaction  src="data1.sql"/>
  <transaction  src="data2.sql"/>
  <transaction  src="data3.sql"/>
```

```
  <transaction>
    truncate table some_other_table;
  </transaction>
</sql>
```

The following example does the same as (and may execute additional SQL files if there are more files matching the pattern `data*.sql`) but doesn't guarantee that data1.sql will be run before `data2.sql`.

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass">
  <path>
    <fileset dir=".">
      <include name="data*.sql"/>
    </fileset>
  </path>
  <transaction>
    truncate table some_other_table;
  </transaction>
</sql>
```

The following connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the SQL statements contained within the file data.sql, with output piped to outputfile.txt, searching /some/jdbc.jar as well as the system classpath for the driver class.

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    src="data.sql"
    print="yes"
    output="outputfile.txt"
    >
<classpath>
        <pathelement location="/some/jdbc.jar"/>
</classpath>
</sql>
```

The following will only execute if the RDBMS is "oracle" and the version starts with "8.1."

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"
    src="data.sql"
    rdbms="oracle"
    version="8.1."
    >
insert
into table some_table
values(1,2,3,4);

truncate table some_other_table;
</sql>
```

Typedef

Description

Adds a task or a data type definition to the current project such that this new type or task can be used in the current project.

A Task is any class that extends org.apache.tools.ant.Task or can be adapted as a Task using an adapter class.

Data types are things like paths or filesets that can be defined at the project level and referenced via their ID attribute. Custom data types usually need custom tasks to put them to good use.

Two attributes are needed to make a definition: the name that identifies this data type uniquely, and the full name of the class (including its package name) that implements this type.

You can also define a group of definitions at once using the file or resource attributes. These attributes point to files in the format of Java property files or an xml format.

For property files each line defines a single data type in the format:

```
    typename=fully.qualified.java.classname
```

The xml format is described in the Antlib section.

Parameters

Attribute	Description	Required
name	the name of the data type	Yes, unless the file or resource type attributes have been specified.
classname	the full class name implementing the data type	Yes, unless file or resource have been specified.
file	Name of the file to load definitions from.	No
resource	Name of the resource to load definitions from. If multiple resources by this name are found along the classpath, and the format is "properties", the first resource will be loaded; otherwise all such resources will be loaded.	No
format	The format of the file or resource. The values are "properties" or "xml". If the value is "properties" the file/resource is a property file contains name to classname pairs. If the value is "xml", the file/resource is an xml file/resource structured according to Antlib. The default is "properties" unless the file/resource name ends with ".xml", in which case the format attribute will have the value "xml". **since Ant 1.6**	No
classpath	the classpath to use when looking up `classname`.	No

classpathref	a reference to a classpath to use when looking up `classname`.	No
loaderRef	the name of the loader that is used to load the class, constructed from the specified classpath. Use this to allow multiple tasks/types to be loaded with the same loader, so they can call each other. **since Ant 1.5**	No
onerror	The action to take if there was a failure in defining the type. The values are *fail*: cause a build exception; *report*: output a warning, but continue; *ignore*: do nothing. **since Ant 1.6** An additional value is *failall*: cause all behavior of fail, as well as a build exception for the resource or file attribute if the resource or file is not found. **since Ant 1.7** The default is *fail*.	No
adapter	A class that is used to adapt the defined class to another interface/class. The adapter class must implement the interface "org.apache.tools.ant.TypeAdapter". The adapter class will be used to wrap the defined class unless the defined class implements/extends the class defined by the attribute "adaptto". If "adaptto" is not set, the defined class will always be wrapped. **since Ant 1.6**	No
adaptto	This attribute is used in conjunction with the adapter attribute. If the defined class does not implement/extend the interface/class specified by this attribute, the adaptor class will be used to wrap the class. **since Ant 1.6**	No
uri	The uri that this definition should live in. **since Ant 1.6**	No

Parameters specified as nested elements

classpath

`Typedef`'s *classpath* attribute is a path-like structure and can also be set via a nested *classpath* element.

Examples

The following fragment defines define a type called *urlset*.

```
    <typedef name="urlset" classname="com.mydomain.URLSet"/>
```

The data type is now available to Ant. The class `com.mydomain.URLSet` implements this type.

Assuming a class *org.acme.ant.RunnableAdapter* that extends Task and implements *org.apache.tools.ant.TypeAdapter*, and in the execute method invokes *run* on the proxied object, one may use a Runnable class as an Ant task. The following fragment defines a task called *runclock*.

```
    <typedef name="runclock"
             classname="com.acme.ant.RunClock"
             adapter="org.acme.ant.RunnableAdapter"/>
```

The following fragment shows the use of the classpathref and loaderref to load up two definitions.

```
    <path id="lib.path">
      <fileset dir="lib" includes="lib/*.jar"/>
    </path>

    <typedef name="filter1"
             classname="org.acme.filters.Filter1"
             classpathref="lib.path"
             loaderref="lib.path.loader"
             />
    <typedef name="filter2"
             classname="org.acme.filters.Filter2"
             loaderref="lib.path.loader"
             />
```

If you want to load an antlib into a special xml-namespace, the `uri` attribute is important:

```
<project xmlns:antcontrib="antlib:net.sf.antcontrib">
   <taskdef uri="antlib:net.sf.antcontrib"
            resource="net/sf/antcontrib/antlib.xml"
            classpath="path/to/ant-contrib.jar"/>
```

XMLValidate

Description

This task checks that XML files are valid (or only well formed). The task uses the SAX2 parser implementation provided by JAXP by default (probably the one that is used by Ant itself), but one can specify any SAX1/2 parser if needed.

This task supports the use of nested

- `<xmlcatalog>` elements
- `<dtd>` elements which are used to resolve DTDs and entities
- `<attribute>` elements which are used to set features on the parser. These can be any number of `http://xml.org/sax/features/` or other features that your parser may support.
- `<property>` elements, containing string properties

Warning : JAXP creates by default a non namespace aware parser. The `"http://xml.org/sax/features/namespaces"` feature is set by default to `false` by the JAXP implementation used by ant. To validate a document containing namespaces, set the namespaces feature to `true` explicitly by nesting the following element:

```
  <attribute name="http://xml.org/sax/features/namespaces" value="true"/>
```

If you are using for instance a `xsi:noNamespaceSchemaLocation` attribute in your XML files, you will need this namespace support feature.

If you are using a parser not generated by JAXP, by using the `classname` attribute of xmlvalidate, this warning may not apply.

Parameters

Attribute	Description	Required
file	the file(s) you want to check. (optionally can use an embedded fileset)	No
lenient	if true, only check the XML document is well formed (ignored if the specified parser is a SAX1 parser)	No
classname	the parser to use.	No
classpathref	where to find the parser class. Optionally can use an embedded `<classpath>` element.	No
failonerror	fails on a error if set to true (defaults to true).	No
warn	log parser warn events.	No

Nested Elements

dtd

`<dtd>` is used to specify different locations for DTD resolution.

Attribute	Description	Required
publicId	Public ID of the DTD to resolve	Yes

location	Location of the DTD to use, which can be a file, a resource, or a URL	Yes

xmlcatalog

The <xmlcatalog> element is used to perform entity resolution.

attribute

The <attribute> element is used to set parser features.
Features usable with the xerces parser are defined here : Setting features
SAX features are defined here: http://xml.org/sax/features/

Attribute	Description	Required
name	The name of the feature	Yes
value	The boolean value of the feature	Yes

property

The <property> element is used to set properties. These properties are defined here for the xerces XML parser implementation : XML Parser properties Properties can be used to set the schema used to validate the XML file.

Attribute	Description	Required
name	The name of the feature	Yes
value	The string value of the property	Yes

Examples

```
<xmlvalidate file="toto.xml"/>
```

Validate toto.xml

```
<xmlvalidate failonerror="no" lenient="yes" warn="yes"
             classname="org.apache.xerces.parsers.SAXParser">
             classpath="lib/xerces.jar">
  <fileset dir="src" includes="style/*.xsl"/>
</xmlvalidate>
```

Validate all .xsl files in src/style, but only warn if there is an error, rather than halt the build.

```
<xmlvalidate file="struts-config.xml" warn="false">
  <dtd publicId="-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
       location="struts-config_1_0.dtd"/>
</xmlvalidate>
```

Validate a struts configuration, using a local copy of the DTD.

```
<xmlvalidate failonerror="no">
  <fileset dir="${project.dir}" includes="**/*.xml"/>
  <xmlcatalog refid="mycatalog"/>
</xmlvalidate>
```

Scan all XML files in the project, using a predefined catalog to map URIs to local files.

```
<xmlvalidate failonerror="no">
  <fileset dir="${project.dir}" includes="**/*.xml"/>
  <xmlcatalog>
       <dtd
```

```
        publicId="-//ArielPartners//DTD XML Article V1.0//EN"
        location="com/arielpartners/knowledgebase/dtd/article.dtd"/>
  </xmlcatalog>
</xmlvalidate>
```

Scan all XML files in the project, using the catalog defined inline.

```
<xmlvalidate failonerror="yes" lenient="no" warn="yes">
  <fileset dir="xml" includes="**/*.xml"/>
  <attribute name="http://xml.org/sax/features/validation" value="true"/>
  <attribute name="http://apache.org/xml/features/validation/schema"  value="true"/>
  <attribute name="http://xml.org/sax/features/namespaces" value="true"/>
</xmlvalidate>
```

Validate all .xml files in xml directory with the parser configured to perform XSD validation. Note: The parser must support the feature `http://apache.org/xml/features/validation/schema`. The schemavalidate task is better for validating W3C XML Schemas, as it extends this task with the right options automatically enabled, and makes it easy to add a list of schema files/URLs to act as sources.

```
<pathconvert dirsep="/" property="xsd.file">
<path>
   <pathelement location="xml/doc.xsd"/>
</path>
</pathconvert>

<xmlvalidate file="xml/endpiece-noSchema.xml" lenient="false"
  failonerror="true" warn="true">
  <attribute name="http://apache.org/xml/features/validation/schema"
  value="true"/>
  <attribute name="http://xml.org/sax/features/namespaces" value="true"/>
  <property
  name="http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation"
  value="${xsd.file}"/>
</xmlvalidate>
```

Validate the file xml/endpiece-noSchema.xml against the schema xml/doc.xsd.

ANTLR

Description

Invokes the ANTLR Translator generator on a grammar file.

To use the ANTLR task, set the *target* attribute to the name of the grammar file to process. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise ANTLR writes the generated files to the directory containing the grammar file.

This task only invokes ANTLR if the grammar file (or the supergrammar specified by the glib attribute) is newer than the generated files.

Antlr 2.7.1 Note: *To successfully run ANTLR, your best option is probably to build the whole jar with the provided script* ***mkalljar*** *and drop the resulting jar (about 300KB) into ${ant.home}/lib. Dropping the default jar (70KB) is probably not enough for most needs and your only option will be to add ANTLR home directory to your classpath as described in ANTLR* `install.html` *document.*

Antlr 2.7.2 Note: *Instead of the above, you will need antlrall.jar that can be created by the* ***antlr-all.jar*** *target of the Makefile provided with the download.*

Parameters

Attribute	Description	Required
target	The grammar file to process.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
glib	An optional super grammar file that the target grammar overrides. This feature is only needed for advanced vocabularies.	No
debug	When set to "yes", this flag adds code to the generated parser that will launch the ParseView debugger upon invocation. The default is "no". Note: ParseView is a separate component that needs to be installed or your grammar will have compilation errors.	No
html	Emit an html version of the grammar with hyperlinked actions.	No
diagnostic	Generates a text file with debugging information based on the target grammar.	No
trace	Forces **all** rules to call traceIn/traceOut if set to "yes". The default is "no".	No
traceParser	Only forces parser rules to call traceIn/traceOut if set to "yes". The default is "no".	No
traceLexer	Only forces lexer rules to call traceIn/traceOut if set to "yes". The default is "no".	No
traceTreeWalker	Only forces tree walker rules to call traceIn/traceOut if set to "yes". The default is "no".	No
dir	The directory to invoke the VM in.	No

Nested Elements

`ANTLR` supports a nested `<classpath>` element, that represents a PATH like structure. It is given as a convenience if you have to specify the original ANTLR directory. In most cases, dropping the appropriate ANTLR jar in the normal Ant lib repository will be enough.

jvmarg

Additional parameters may be passed to the new VM via nested `<jvmarg>` attributes, for example:

```
<antlr target="...">
  <jvmarg value="-Djava.compiler=NONE"/>
  ...
</antlr>
```

would run ANTLR in a VM without JIT.

`<jvmarg>` allows all attributes described in Command line arguments.

Example

```
<antlr
    target="etc/java.g"
    outputdirectory="build/src"
/>
```

This invokes ANTLR on grammar file etc/java.g, writing the generated files to build/src.

AntStructure

Description

Generates an DTD for Ant buildfiles which contains information about all tasks currently known to Ant.

Actually the DTD will not be a real DTD for buildfiles since Ant's usage of XML cannot be captured with a DTD. Several elements in Ant can have different attribute lists depending on the element that contains them. "fail" for example can be the task or the nested child element of the sound task. Don't consider the generated DTD something to rely upon.

Also note that the DTD generated by this task is incomplete, you can always add XML entities using `<taskdef>` or `<typedef>`. See here for a way to get around this problem.

This task doesn't know about required attributes, all will be listed as `#IMPLIED`.

Since Ant 1.7 custom structure printers can be used instead of the one that emits a DTD. In order to plug in your own structure, you have to implement the interface `org.apache.tools.ant.taskdefs.AntStructure.StructurePrinter` and `<typedef> your class and use the new type as a nested element of this task - see the example below.`

Parameters

Attribute	Description	Required
output	file to write the DTD to.	Yes

Examples

```
<antstructure output="project.dtd" />
```

Emitting your own structure instead of a DTD

First you need to implement the interface

```
package org.example;
import org.apache.tools.ant.taskdefs.AntStructure;
public class MyPrinter implements AntStructure.StructurePrinter {
    ...
}
```

and then use it via typedef

```
  <typedef name="myprinter" classname="org.example.MyPrinter" />
  <antstructure output="project.my">
    <myprinter />
  </antstructure>
```

Your own StructurePrinter can accept attributes and nested elements just like any other Ant type or task.

Include

Description

Include another build file into the current project.

since Ant 1.8.0

Note this task heavily relies on the ProjectHelper implementation and doesn't really perform any work of its own. If you have configured Ant to use a ProjectHelper other than Ant's default, this task may or may not work.

On execution it will read another Ant file into the same Project rewriting the included target names and depends lists. This is different from Entity Includes as explained in the Ant FAQ in so far as the target names get prefixed by the included project's name or the as attribute and do not appear as if the file was contained in the including file.

The include task may only be used as a top-level task. This means that it may not be used in a target.

There are two further functional aspects that pertain to this task and that are not possible with entity includes:

- target rewriting
- special properties

Target rewriting

Any target in the included file will be renamed to *prefix.name* where *name* is the original target's name and *prefix* is either the value of the *as* attribute or the *name* attribute of the *project* tag of the included file.

The depends attribute of all included targets is rewritten so that all target names are prefixed as well. This makes the included file self-contained.

Note that prefixes nest, so if a build file includes a file with prefix "a" and the included file includes another file with prefix "b", then the targets of that last build file will be prefixed by "a.b.".

`<import>` contribute to the prefix as well, but only if their `as` attribute has been specified.

Special Properties

Included files are treated as they are present in the main buildfile. This makes it easy to understand, but it makes it impossible for them to reference files and resources relative to their path. Because of this, for every included file, Ant adds a property that contains the path to the included buildfile. With this path, the included buildfile can keep resources and be able to reference them relative to its position.

So if I include for example a *docsbuild.xml* file named **builddocs**, I can get its path as **ant.file.builddocs**, similarly to the **ant.file** property of the main buildfile.

Note that "builddocs" is not the filename, but the name attribute present in the included project tag.

If the included file does not have a name attribute, the ant.file.projectname property will not be set.

If you need to know whether the current build file's source has been a file or an URL you can consult the property **ant.file.type.*projectname*** (using the same example as above **ant.file.type.builddocs**) which either have the value "file" or "url".

Resolving files against the included file

Suppose your main build file called `including.xml` includes a build file `included.xml`, located anywhere on the file system, and `included.xml` reads a set of properties from `included.properties`:

```
<!-- including.xml -->
<project name="including" basedir="." default="...">
  <include file="${path_to_included}/included.xml"/>
</project>

<!-- included.xml -->
<project name="included" basedir="." default="...">
  <property file="included.properties"/>
</project>
```

This snippet however will resolve `included.properties` against the basedir of `including.xml`, because the basedir of `included.xml` is ignored by Ant. The right way to use `included.properties` is:

```
<!-- included.xml -->
<project name="included" basedir="." default="...">
  <dirname property="included.basedir" file="${ant.file.included}"/>
  <property file="${included.basedir}/included.properties"/>
</project>
```

As explained above `${ant.file.included}` stores the path of the build script, that defines the project called `included`, (in short it stores the path to `included.xml`) and `<dirname>` takes its directory. This technique also allows `included.xml` to be used as a standalone file (without being included in other project).

The above description only works for included files that actually are included from files and not from URLs. For files included from URLs using resources relative to the included file requires you to use tasks that can work on non-file resources in the first place. To create a relative resource you'd use something like:

```
  <loadproperties>
    <url baseUrl="${ant.file.included}"
         relativePath="included.properties"/>
  </loadproperties>
```

Parameters

Attribute	Description	Required
file	The file to include. If this is a relative file name, the file name will be resolved relative to the *including* file. **Note**, this is unlike most other ant file attributes, where relative files are resolved relative to ${basedir}.	Yes or a nested resource collection
optional	If true, do not stop the build if the file does not exist, default is false.	No
as	Specifies the prefix prepended to the target names. If ommitted, the name attribute of the project tag of the included file will be used.	Yes, if the included file's project tag doesn't specify a name attribute.
prefixSeparator	Specifies the separator to be used between the prefix and the target name. Defaults to ".".	No

Parameters specified as nested elements

any resource or resource collection

The specified resources will be included.

Examples

```
  <include file="../common-targets.xml"/>
```

Includes targets from the common-targets.xml file that is in a parent directory.

```
  <include file="${deploy-platform}.xml"/>
```

Includes the project defined by the property deploy-platform

```
  <include>
    <javaresource name="common/targets.xml">
      <classpath location="common.jar"/>
    </javaresource>
  </include>
```

Includes targets from the targets.xml file that is inside the directory common inside the jar file common.jar.

How is <import> different from <include>?

The short version: Use import if you intend to override a target, otherwise use include.

When using import the imported targets are available by up to two names. Their "normal" name without any prefix and potentially with a prefixed name (the value of the as attribute or the imported project's name attribute, if any).

When using include the included targets are only available in the prefixed form.

When using import, the imported target's depends attribute remains unchanged, i.e. it uses "normal" names and allows you to override targets in the dependency list.

When using include, the included targets cannot be overridden and their depends attributes are rewritten so that prefixed names are used. This allows writers of the included file to control which target is invoked as part of the dependencies.

It is possible to include the same file more than once by using different prefixes, it is not possible to import the same file more than once.

Examples

nested.xml shall be:

```
<project>
  <target name="setUp">
    <property name="prop" value="in nested"/>
  </target>

  <target name="echo" depends="setUp">
    <echo>prop has the value ${prop}</echo>
  </target>
</project>
```

When using import like in

```
<project default="test">
  <target name="setUp">
    <property name="prop" value="in importing"/>
  </target>

  <import file="nested.xml" as="nested"/>

  <target name="test" depends="nested.echo"/>
```

```
</project>
```

Running the build file will emit:

```
setUp:

nested.echo:
     [echo] prop has the value in importing

test:
```

When using include like in

```
<project default="test">
  <target name="setUp">
    <property name="prop" value="in importing"/>
  </target>

  <include file="nested.xml" as="nested"/>

  <target name="test" depends="nested.echo"/>
</project>
```

Running the target build file will emit:

```
nested.setUp:

nested.echo:
     [echo] prop has the value in nested

test:
```

and there won't be any target named "echo" on the including build file.

JavaCC

Description

Invokes the JavaCC compiler compiler on a grammar file.

To use the javacc task, set the *target* attribute to the name of the grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise javacc writes the generated files to the directory containing the grammar file.

This task only invokes JavaCC if the grammar file is newer than the generated Java files. javacc assumes that the Java class name of the generated parser is the same as the name of the grammar file, ignoring the .jj. If this is not the case, the javacc task will still work, but it will always generate the output files.

Parameters

Attribute	Description	Required
target	The grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
buildparser	Sets the BUILD_PARSER grammar option. This is a boolean option.	No
buildtokenmanager	Sets the BUILD_TOKEN_MANAGER grammar option. This is a boolean option.	No
cachetokens	Sets the CACHE_TOKENS grammar option. This is a boolean option.	No
choiceambiguitycheck	Sets the CHOICE_AMBIGUITY_CHECK grammar option. This is an integer option.	No
commontokenaction	Sets the COMMON_TOKEN_ACTION grammar option. This is a boolean option.	No
debuglookahead	Sets the DEBUG_LOOKAHEAD grammar option. This is a boolean option.	No
debugparser	Sets the DEBUG_PARSER grammar option. This is a boolean option.	No
debugtokenmanager	Sets the DEBUG_TOKEN_MANAGER grammar option. This is a boolean option.	No
errorreporting	Sets the ERROR_REPORTING grammar option. This is a boolean option.	No
forcelacheck	Sets the FORCE_LA_CHECK grammar option. This is a boolean option.	No
ignorecase	Sets the IGNORE_CASE grammar option. This is a boolean option.	No
javaunicodeescape	Sets the JAVA_UNICODE_ESCAPE grammar option. This is a boolean option.	No
jdkversion	Sets the JDK_VERSION option. This is a string option.	No
keeplinecolumn	Sets the KEEP_LINE_COLUMN grammar option. This is a boolean option.	No
lookahead	Sets the LOOKAHEAD grammar option. This is an integer option.	No
optimizetokenmanager	Sets the OPTIMIZE_TOKEN_MANAGER grammar option. This is a boolean option.	No
otherambiguitycheck	Sets the OTHER_AMBIGUITY_CHECK grammar option. This is an integer option.	No
sanitycheck	Sets the SANITY_CHECK grammar option. This is a boolean option.	No

static	Sets the STATIC grammar option. This is a boolean option.	No
unicodeinput	Sets the UNICODE_INPUT grammar option. This is a boolean option.	No
usercharstream	Sets the USER_CHAR_STREAM grammar option. This is a boolean option.	No
usertokenmanager	Sets the USER_TOKEN_MANAGER grammar option. This is a boolean option.	No

Example

```
<javacc
    target="src/Parser.jj"
    outputdirectory="build/src"
    javacchome="c:/program files/JavaCC"
    static="true"
/>
```

This invokes JavaCC on grammar file src/Parser.jj, writing the generated files to build/src. The grammar option STATIC is set to true when invoking JavaCC.

JJDoc

Since Ant 1.6

Description

Invokes the JJDoc preprocessor for the JavaCC compiler compiler. It takes a JavaCC parser specification and produces documentation for the BNF grammar. It can operate in three modes, determined by command line options.

To use the jjdoc task, set the *target* attribute to the name of the JavaCC grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputfile* to write the generated BNF documentation file to a specific (directory and) file. Otherwise jjdoc writes the generated BNF documentation file as the JavaCC grammar file with a suffix .txt or .html.

This task only invokes JJDoc if the grammar file is newer than the generated BNF documentation file.

Parameters

Attribute	Description	Required
target	The javacc grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputfile	The file to write the generated BNF documentation file to. If not set, the file is written with the same name as the JavaCC grammar file but with a the suffix .html or .txt.	No
text	Sets the TEXT BNF documentation option. This is a boolean option.	No
onetable	Sets the ONE_TABLE BNF documentation option. This is a boolean option.	No

Example

```
<jjdoc
    target="src/Parser.jj"
    outputfile="doc/ParserBNF.html"
    javacchome="c:/program files/JavaCC"
/>
```

This invokes JJDoc on grammar file src/Parser.jj, writing the generated BNF documentation file, ParserBNF.html, file to doc.

JJTree

Description

Invokes the JJTree preprocessor for the JavaCC compiler compiler. It inserts parse tree building actions at various places in the JavaCC source that it generates. The output of JJTree is run through JavaCC to create the parser.

To use the jjtree task, set the *target* attribute to the name of the JJTree grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputdirectory* to write the generated JavaCC grammar and node files to a specific directory. Otherwise jjtree writes the generated JavaCC grammar and node files to the directory containing the JJTree grammar file. As an extra option, you can also set the *outputfile* to write the generated JavaCC grammar file to a specific (directory and) file. Otherwise jjtree writes the generated JavaCC grammar file as the JJTree grammar file with a suffix .jj.

This task only invokes JJTree if the grammar file is newer than the generated JavaCC file.

Parameters

Attribute	Description	Required
target	The jjtree grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputdirectory	The directory to write the generated JavaCC grammar and node files to. If not set, the files are written to the directory containing the grammar file.	No
outputfile	The file to write the generated JavaCC grammar file to. If not set, the file is written with the same name as the JJTree grammar file but with a the suffix `.jj`. This is a filename relative to *outputdirectory* if specified, the project's basedir.	No
buildnodefiles	Sets the BUILD_NODE_FILES grammar option. This is a boolean option.	No
multi	Sets the MULTI grammar option. This is a boolean option.	No
nodedefaultvoid	Sets the NODE_DEFAULT_VOID grammar option. This is a boolean option.	No
nodefactory	Sets the NODE_FACTORY grammar option. This is boolean option.	No
nodescopehook	Sets the NODE_SCOPE_HOOK grammar option. This is a boolean option.	No
nodeusesparser	Sets the NODE_USES_PARSER grammar option. This is a boolean option.	No
static	Sets the STATIC grammar option. This is a boolean option.	No
visitor	Sets the VISITOR grammar option. This is a boolean option.	No
nodepackage	Sets the NODE_PACKAGE grammar option. This is a string option.	No
visitorexception	Sets the VISITOR_EXCEPTION grammar option. This is a string option.	No
nodeprefix	Sets the NODE_PREFIX grammar option. This is a string option.	No

Example

```
<jjtree
    target="src/Parser.jjt"
    outputdirectory="build/src"
    javacchome="c:/program files/JavaCC"
    nodeusesparser="true"
/>
```

This invokes JJTree on grammar file src/Parser.jjt, writing the generated grammar file, Parser.jj, file to build/src. The grammar option NODE_USES_PARSER is set to true when invoking JJTree.

Comparison output locations between command line JJTree and different Ant taskdef versions

Command Line JJTree options *and Generated Files* (working directory: `/tmp`)	**Ant 1.5.3** versus command line	**Ant 1.6** versus command line
jjtree grammar.jjt /tmp/grammar.jj /tmp/<generated>.java	Same	Same
jjtree relative/grammar.jjt /tmp/grammar.jj /tmp/<generated>.java	/tmp/relative/grammar.jj /tmp/relative/<generated>.java	Same
jjtree /tmp/absolute/grammar.jjt /tmp/grammar.jj /tmp/<generated>.java	/tmp/absolute/grammar.jj /tmp/absolute/<generated>.java	Same
jjtree -OUTPUT_DIRECTORY:relative grammar.jjt /tmp/relative/grammar.jj /tmp/relative/<generated>.java	Same	Same
jjtree -OUTPUT_DIRECTORY:relative relative/grammar.jjt /tmp/relative/grammar.jj /tmp/relative/<generated>.java	Same	Same
jjtree -OUTPUT_DIRECTORY:relative /tmp/absolute/grammar.jjt /tmp/relative/grammar.jj /tmp/relative/<generated>.java	Same	Same
jjtree -OUTPUT_DIRECTORY:/tmp/absolute/ grammar.jjt /tmp/absolute/grammar.jj /tmp/absolute/<generated>.java	Same	Same
jjtree -OUTPUT_DIRECTORY:/tmp/absolute/ relative/grammar.jjt /tmp/absolute/grammar.jj /tmp/absolute/<generated>.java	Same	Same
jjtree -OUTPUT_DIRECTORY:/tmp/absolute/ /tmp/absolute/grammar.jjt /tmp/absolute/grammar.jj /tmp/absolute/<generated>.java	Same	Same
jjtree -OUTPUT_FILE:output.jj grammar.jjt /tmp/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj relative/grammar.jjt /tmp/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj /tmp/absolute/grammar.jjt /tmp/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj -OUTPUT_DIRECTORY:relative grammar.jjt /tmp/relative/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj -OUTPUT_DIRECTORY:relative relative/grammar.jjt /tmp/relative/output.jj /tmp/relative/<generated>.java	Not Supported	Same

jjtree -OUTPUT_FILE:output.jj -OUTPUT_DIRECTORY:relative /tmp/absolute/grammar.jjt /tmp/relative/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ grammar.jjt /tmp/absolute/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ relative/grammar.jjt /tmp/absolute/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ /tmp/absolute/grammar.jjt /tmp/absolute/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj relative/grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj /tmp/absolute/grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj -OUTPUT_DIRECTORY:relative grammar.jjt /tmp/relative/subdir/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj -OUTPUT_DIRECTORY:relative relative/grammar.jjt /tmp/relative/subdir/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj -OUTPUT_DIRECTORY:relative /tmp/absolute/grammar.jjt /tmp/relative/subdir/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ grammar.jjt /tmp/absolute/subdir/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ relative/grammar.jjt /tmp/absolute/subdir/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:subdir/output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ /tmp/absolute/grammar.jjt /tmp/absolute/subdir/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj relative/grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj /tmp/absolute/grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Same

jjtree -OUTPUT_FILE:D:/tmp/subdir/output.jj grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Not Supported *)
jjtree -OUTPUT_FILE:D:/tmp/subdir/output.jj relative/grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Not Supported *)
jjtree -OUTPUT_FILE:D:/tmp/subdir/output.jj /tmp/absolute/grammar.jjt /tmp/subdir/output.jj /tmp/<generated>.java	Not Supported	Not Supported *)
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj -OUTPUT_DIRECTORY:relative grammar.jjt /tmp/relative/tmp/subdir/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj -OUTPUT_DIRECTORY:relative relative/grammar.jjt /tmp/relative/tmp/subdir/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj -OUTPUT_DIRECTORY:relative /tmp/absolute/grammar.jjt /tmp/relative/tmp/subdir/output.jj /tmp/relative/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ grammar.jjt /tmp/absolute/tmp/subdir/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ relative/grammar.jjt /tmp/absolute/tmp/subdir/output.jj /tmp/absolute/<generated>.java	Not Supported	Same
jjtree -OUTPUT_FILE:/tmp/subdir/output.jj -OUTPUT_DIRECTORY:/tmp/absolute/ /tmp/absolute/grammar.jjt /tmp/absolute/tmp/subdir/output.jj /tmp/absolute/<generated>.java	Not Supported	Same

*) Footnote: When running JJTree with the Ant taskdef *jjtree* the option `-OUTPUT_DIRECTORY` must always be set, because the project's basedir and the ant working directory might differ. So even if you don't specify the jjtree taskdef *outputdirectory* JJTree will be called with the `-OUTPUT_DIRECTORY` set to the project's basedirectory. But when the `-OUTPUT_DIRECTORY` is set, the `-OUTPUT_FILE` setting is handled as if relative to this `-OUTPUT_DIRECTORY`. Thus when the `-OUTPUT_FILE` is absolute or contains a drive letter we have a problem. Therefore absolute *outputfile*s (when the *outputdirectory* isn't specified) are made relative to the default directory. And for this reason *outputfile*s that contain a drive letter can't be supported.

By the way: specifying a drive letter in the `-OUTPUT_FILE` when the `-OUTPUT_DIRECTORY` is set, also results in strange behavior when running JJTree from the command line.

Native2Ascii

Description:

Converts files from native encodings to ASCII with escaped Unicode. A common usage is to convert source files maintained in a native operating system encoding, to ASCII prior to compilation.

Files in the directory *src* are converted from a native encoding to ASCII. By default, all files in the directory are converted. However, conversion may be limited to selected files using *includes* and *excludes* attributes. For more information on file matching patterns, see the section on directory based tasks. If no *encoding* is specified, the default encoding for the JVM is used. If *ext* is specified, then output files are renamed to use it as a new extension. More sophisticated file name translations can be achieved using a nested `<mapper>` element. By default an identity mapper will be used. If *dest* and *src* point to the same directory, the *ext* attribute or a nested `<mapper>` is required.

This task forms an implicit File Set, and supports most attributes of `<fileset>` (`dir` becomes `src`) as well as nested `<include>`, `<exclude>`, and `<patternset>` elements.

It is possible to use different converters. This can be selected with the `implementation` attribute or a nested element. Here are the choices of the attribute:

- default - the default converter (kaffe or sun) for the platform.
- sun (the standard converter of the JDK)
- kaffe (the standard converter of Kaffe)

Attribute	Description	Required
reverse	Reverse the sense of the conversion, i.e. convert from ASCII to native **only supported by the sun converter**	No
encoding	The native encoding the files are in (default is the default encoding for the JVM)	No
src	The directory to find files in (default is *basedir*)	No
dest	The directory to output file to	Yes
ext	File extension to use in renaming output files	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
implementation	The converter implementation to use. If this attribute is not set, the default converter for the current VM will be used. (See the above list of valid converters.)	No

Parameters specified as nested elements

arg

You can specify additional command line arguments for the converter with nested `<arg>` elements. These elements are

specified like Command-line Arguments but have an additional attribute that can be used to enable arguments only if a given converter implementation will be used.

Attribute	Description	Required
value	See Command-line Arguments.	Exactly one of these.
line		
file		
path		
implementation	Only pass the specified argument if the chosen converter implementation matches the value of this attribute. Legal values are the same as those in the above list of valid compilers.)	No

implementationclasspath *since Ant 1.8.0*

A PATH like structure holding the classpath to use when loading the converter implementation if a custom class has been specified. Doesn't have any effect when using one of the built-in converters.

Any nested element of a type that implements Native2AsciiAdapter *since Ant 1.8.0*

If a defined type implements the `Native2AsciiAdapter` interface a nested element of that type can be used as an alternative to the `implementation` attribute.

Examples

```
<native2ascii encoding="EUCJIS" src="srcdir" dest="srcdir"
   includes="**/*.eucjis" ext=".java"/>
```

Converts all files in the directory *srcdir* ending in `.eucjis` from the EUCJIS encoding to ASCII and renames them to end in `.java`.

```
<native2ascii encoding="EUCJIS" src="native/japanese" dest="src"
   includes="**/*.java"/>
```

Converts all the files ending in `.java` in the directory *native/japanese* to ASCII, placing the results in the directory *src*. The names of the files remain the same.

If you want to use a custom Native2AsciiAdapter `org.example.MyAdapter` you can either use the implementation attribute:

```
<native2ascii encoding="EUCJIS" src="srcdir" dest="srcdir"
       includes="**/*.eucjis" ext=".java"
       implementation="org.example.MyAdapter"/>
```

or a define a type and nest this into the task like in:

```
<componentdef classname="org.example.MyAdapter"
              name="myadapter"/>
<native2ascii encoding="EUCJIS" src="srcdir" dest="srcdir"
       includes="**/*.eucjis" ext=".java">
  <myadapter/>
</native2ascii>
```

in which case your native2ascii adapter can support attributes and nested elements of its own.

PreSetDef

Description

The preset definition generates a new definition based on a current definition with some attributes or elements preset.

since Ant 1.6

The resolution of properties in any of the attributes or nested text takes place with the definition is used and *not* when the preset definition is defined.

Parameters

Attribute	Description	Required
name	the name of the new definition	Yes
uri	The uri that this definition should live in.	No

Parameters specified as nested elements

another type with attributes or elements set

The `<presetdef>` task takes one nested element as a parameter. This nested element can be any other type or task. The attributes and elements that need to be preset are placed here.

Examples

The following fragment defines a javac task with the debug, deprecation srcdir and destdir attributes set. It also has a src element to source files from a generated directory.

```
<presetdef name="my.javac">
   <javac debug="${debug}" deprecation="${deprecation}"
          srcdir="${src.dir}" destdir="${classes.dir}">
      <src path="${gen.dir}"/>
   </javac>
</presetdef>
```

This can be used as a normal javac task - example:

```
<my.javac/>
```

The attributes specified in the preset task may be overridden - i.e. they may be seen as optional attributes - example:

```
<my.javac srcdir="${test.src}" deprecation="no"/>
```

One may put a presetdef definition in an antlib. For example suppose the jar file antgoodies.jar has the antlib.xml as follows:

```
<antlib>
   <taskdef resource="com/acme/antgoodies/tasks.properties"/>
   <!-- Implement the common use of the javac command -->
   <presetdef name="javac">
      <javac deprecation="${deprecation}" debug="${debug}"
             srcdir="src" destdir="classes"/>
   </presetdef>
```

```
</antlib>
```

One may then use this in a build file as follows:

```
<project default="example" xmlns:antgoodies="antlib:com.acme.antgoodies">
   <target name="example">
      <!-- Compile source -->
      <antgoodies:javac srcdir="src/main"/>
      <!-- Compile test code -->
      <antgoodies:javac srcdir="src/test"/>
   </target>
</project>
```

The following is an example of evaluation of properties when the definition is used:

```
<target name="defineandcall">
   <presetdef name="showmessage">
      <echo>message is '${message}'</echo>
   </presetdef>
   <showmessage/>
   <property name="message" value="Message 1"/>
   <showmessage/>
   <antcall target="called">
      <param name="message" value="Message 2"/>
   </antcall>
</target>
<target name="called">
   <showmessage/>
</target>
```

The command ant defineandcall results in the output:

```
defineandcall:
[showmessage] message is '${message}'
[showmessage] message is 'Message 1'

called:
[showmessage] message is 'Message 2'
```

It is possible to use a trick to evaluate properties when the definition is *made* rather than used. This can be useful if you do not expect some properties to be available in child builds run with `<ant ... inheritall="false">`:

```
<macrodef name="showmessage-presetdef">
  <attribute name="messageval"/>
  <presetdef name="showmessage">
    <echo>message is '@{messageval}'</echo>
  </presetdef>
</macrodef>
<showmessage-presetdef messageval="${message}"/>
```

XSLT

The name `style` is a deprecated name for the same task.

Description

Process a set of documents via XSLT.

This is useful for building views of XML based documentation, or for generating code.

Note: If you are using JDK 1.4 or higher, this task does not require external libraries not supplied in the Ant distribution. However, often the built in XSL engine is not as up to date as a fresh download, so an update is still highly recommended in particular since the built-in XSLT processors of Java 5 (and to a certain extent Java 6) are known to have serious issues. See Library Dependencies for more information.

It is possible to refine the set of files that are being processed. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Note: Unlike other similar tasks, this task treats directories that have been matched by the include/exclude patterns of the implicit fileset in a special way. It will apply the stylesheets to all files contain in them as well. Since the default include pattern is `**` this means it will apply the stylesheet to all files. If you specify an excludes pattern, it may still work on the files matched by those patterns because the parent directory has been matched. If this behavior is not what you want, set the scanincludedirectories attribute to false.

Starting with Ant 1.7 this task supports nested resource collections in addition to (or instead of, depending on the useImplicitFileset attribute) the implicit fileset formed by this task.

This task supports the use of a nested `<param>` element which is used to pass values to an `<xsl:param>` declaration.

This task supports the use of a nested xmlcatalog element which is used to perform Entity and URI resolution.

Parameters

Attribute	Description	Required
basedir	where to find the source XML file, default is the project's basedir.	No
destdir	directory in which to store the results.	Yes, unless in and out have been specified.
extension	desired file extension to be used for the targets. If not specified, the default is ".html". Will be ignored if a nested `<mapper>` has been specified.	No
style	name of the stylesheet to use - given either relative to the project's basedir or as an absolute path.	No, if the location of the stylesheet is

	Alternatively, a nested element which ant can interpret as a resource can be used to indicate where to find the stylesheet *deprecated variation :* If the stylesheet cannot be found, and if you have specified the attribute basedir for the task, ant will assume that the style attribute is relative to the basedir of the task.	specified using a nested <style> element
classpath	the classpath to use when looking up the XSLT processor.	No
classpathref	the classpath to use, given as reference to a path defined elsewhere.	No
force	Recreate target files, even if they are newer than their corresponding source files or the stylesheet.	No; default is false
processor	name of the XSLT processor to use. Permissible value is : • "trax" for a TraX compliant processor (ie JAXP interface implementation such as Xalan 2 or Saxon) Defaults to trax. Support for xalan1 has been removed in ant 1.7.	No
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
in	specifies a single XML document to be styled. Should be used with the out attribute.	No
out	specifies the output name for the styled result from the in attribute.	No
scanincludeddirectories	If any directories are matched by the includes/excludes patterns, try to transform all files in these directories. Default is `true`	No
reloadstylesheet	Control whether the stylesheet transformer is created anew for every transform opertaion. If you set this to true, performance may suffer, but you may work around a bug in certain Xalan-J versions. Default is `false`. *Since Ant 1.5.2*.	No
useImplicitFileset	Whether the implicit fileset formed by this task shall be used. If you set this to false you must use nested resource collections - or the in attribute, in which case this attribute has no impact anyway. Default is `true`. *Since Ant 1.7*.	No
filenameparameter	Specifies a xsl parameter for accessing the name of the current processed file. If not set, the file name is not passed to the transformation. *Since Ant 1.7*.	No
filedirparameter	Specifies a xsl parameter for accessing the directory of the current processed file. For files in the current directory a value of '.' will be passed to the transformation. If not set, the directory is not	No

	passed to the transformation. *Since Ant 1.7.*	
supressWarnings	Whether processor warnings shall be suppressed. This option requires support by the processor, it is supported by the trax processor bundled with Ant. *Since Ant 1.8.0.*	No, default is false.
failOnError	Whether the build should fail if any error occurs. Note that transformation errors can still be surpressed by setting failOnTransformationError to false even if this attribute is true. *Since Ant 1.8.0.*	No, default is true.
failOnTransformationError	Whether the build should fail if an error occurs while transforming the document. Note that this attribute has no effect of `failOnError` is false. *Since Ant 1.8.0.*	No, default is true.
failOnNoResources	Whether the build should fail if the nested resource collection is empty. Note that this attribute has no effect of `failOnError` is false. *Since Ant 1.8.0.*	No, default is true.

Parameters specified as nested elements

any resource collection

since Ant 1.7

Use resource collections to specify resources that the stylesheet should be applied to. Use a nested mapper and the task's destdir attribute to specify the output files.

classpath

The classpath to load the processor from can be specified via a nested `<classpath>`, as well - that is, a path-like structure.

xmlcatalog

The xmlcatalog element is used to perform Entity and URI resolution.

param

Param is used to pass a parameter to the XSL stylesheet.

Parameters

Attribute	Description	Required
name	Name of the XSL parameter	Yes
expression	Text value to be placed into the param. Was originally intended to be an XSL expression.	Yes
if	The param will only be passed if this property is set.	No
unless	The param will not be passed if this property is set.	No

outputproperty ('trax' processors only)

Used to specify how you wish the result tree to be output as specified in the XSLT specifications.

Parameters

Attribute	Description	Required
name	Name of the property	Yes
value	value of the property.	Yes

factory ('trax' processors only)

Used to specify factory settings.

Parameters

Attribute	Description	Required
name	fully qualified classname of the transformer factory to use. For example `org.apache.xalan.processor.TransformerFactoryImpl` or `org.apache.xalan.xsltc.trax.TransformerFactoryImpl` or `net.sf.saxon.TransformerFactoryImpl`...	No. Defaults to the JAXP lookup mechanism.

Parameters specified as nested elements

attribute

Used to specify settings of the processor factory. The attribute names and values are entirely processor specific so you must be aware of the implementation to figure them out. Read the documentation of your processor. For example, in Xalan 2.x:

- http://xml.apache.org/xalan/features/optimize (boolean)
- http://xml.apache.org/xalan/features/incremental (boolean)
- ...

And in Saxon 7.x:

- http://saxon.sf.net/feature/allow-external-functions (boolean)
- http://saxon.sf.net/feature/timing (boolean)
- http://saxon.sf.net/feature/traceListener (string)
- http://saxon.sf.net/feature/treeModel (integer)
- http://saxon.sf.net/feature/linenumbering (integer)
- ...

Parameters

Attribute	Description	Required
name	Name of the attribute	Yes

value	value of the attribute.	Yes

mapper

since Ant 1.6.2

You can define filename transformations by using a nested mapper element. The default mapper used by `<xslt>` removes the file extension from the source file and adds the extension specified via the extension attribute.

style

Since Ant 1.7

The nested style element can be used to specify your stylesheet in terms of Ant's resource types. With this element, the stylesheet should be specified as a nested resource or single-element collection. Alternatively, use the `refid` to specify the resource or collection as a reference.

sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the factory or transformation. These properties will be made available to the VM during the execution of the class. The attributes for this element are the same as for environment variables.

since Ant 1.8.0.

syspropertyset

You can specify a set of properties to be used as system properties with syspropertysets.

since Ant 1.8.0.

Examples

```
<xslt basedir="doc" destdir="build/doc"
       extension=".html" style="style/apache.xsl"/>
```

Using an xmlcatalog

```
<xslt basedir="doc" destdir="build/doc"
      extension=".html" style="style/apache.xsl">
  <xmlcatalog refid="mycatalog"/>
</xslt>

<xslt basedir="doc" destdir="build/doc"
   extension=".html" style="style/apache.xsl">
   <xmlcatalog>
       <dtd
         publicId="-//ArielPartners//DTD XML Article V1.0//EN"
         location="com/arielpartners/knowledgebase/dtd/article.dtd"/>
   </xmlcatalog>
</xslt>
```

Using XSL parameters

```
<xslt basedir="doc" destdir="build/doc"
      extension=".html" style="style/apache.xsl">
  <param name="date" expression="07-01-2000"/>
</xslt>
```

Then if you declare a global parameter "date" with the top-level element <xsl:param name="date"/>, the variable `$date` will subsequently have the value 07-01-2000.

Using output properties

```
<xslt in="doc.xml" out="build/doc/output.xml"
      style="style/apache.xsl">
  <outputproperty name="method" value="xml"/>
  <outputproperty name="standalone" value="yes"/>
  <outputproperty name="encoding" value="iso8859_1"/>
  <outputproperty name="indent" value="yes"/>
</xslt>
```

Using factory settings

```
<xslt in="doc.xml" out="build/doc/output.xml"
      style="style/apache.xsl">
  <factory name="org.apache.xalan.processor.TransformerFactoryImpl">
    <attribute name="http://xml.apache.org/xalan/features/optimize" value="true"/>
  </factory>
</xslt>
```

Using a mapper

```
<xslt basedir="in" destdir="out"
      style="style/apache.xsl">
  <mapper type="glob" from="*.xml.en" to="*.html.en"/>
</xslt>
```

Using a nested resource to define the stylesheet

```
<xslt in="data.xml" out="${out.dir}/out.xml">
    <style>
        <url url="${printParams.xsl.url}"/>
    </style>
    <param name="set" expression="value"/>
</xslt>
```

Print the current processed file name

```
<project>
  <xslt style="printFilename.xsl" destdir="out" basedir="in" extension=".txt"
        filenameparameter="filename"
        filedirparameter="filedir"
  />
</project>

<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:param name="filename"></xsl:param>
    <xsl:param name="filedir">.</xsl:param>

<xsl:template match="/">
  Current file is <xsl:value-of select="$filename"/> in directory <xsl:value-of
select="$filedir"/>.
</xsl:template>

</xsl:stylesheet>
```

Use an XInclude-aware version of Xerces while transforming

```
<xslt ...>
    <sysproperty key="org.apache.xerces.xni.parser.XMLParserConfiguration"
                 value="org.apache.xerces.parsers.XIncludeParserConfiguration"
```

```
    />
<xslt>
```

Available

Description

Sets a property if a resource is available at runtime. This resource can be a file, a directory, a class in the classpath, or a JVM system resource.

If the resource is present, the property value is set to true by default; otherwise, the property is not set. You can set the value to something other than the default by specifying the `value` attribute.

Normally, this task is used to set properties that are useful to avoid target execution depending on system parameters.

Parameters

Attribute	Description	Required
property	The name of the property to set.	Yes
value	The value to set the property to. Defaults to "true".	No
classname	The class to look for in the classpath.	Yes
file	The file to look for.	
resource	The resource to look for in the JVM.	
classpath	The classpath to use when looking up `classname` or `resource`.	No
filepath	The path to use when looking up `file`.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No
type	The type of `file` to look for, either a directory (`type="dir"`) or a file (`type="file"`). If not set, the property will be set if the name specified in the `file` attribute exists as either a file or a directory.	No
ignoresystemclasses	Ignore Ant's runtime classes, using only the specified classpath. Only affects the "classname" attribute. Defaults to "false"	No
searchparents	This contains the behaviour of the "file" type. If true, the available task will, when searching for a file, search not only the directories specified but will also search the parent directories of those specified. If false, only the directories specified will be searched. Defaults to "false". *Since Ant 1.7*	No

Parameters specified as nested elements

classpath

`Available`'s `classpath` attribute is a path-like structure and can also be set via a nested `<classpath>` element.

filepath

`Available`'s `filepath` attribute is a path-like structure and can also be set via a nested `<filepath>` element.

Examples

```
<available classname="org.whatever.Myclass" property="Myclass.present"/>
```

sets the `Myclass.present` property to the value "true" if the class `org.whatever.Myclass` is found in Ant's classpath.

```
<property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
<available file="${jaxp.jar}" property="jaxp.jar.present"/>
```

sets the `jaxp.jar.present` property to the value "true" if the file `./lib/jaxp11/jaxp.jar` is found.

```
<available file="/usr/local/lib" type="dir"
           property="local.lib.present"/>
```

sets the `local.lib.present` property to the value "true" if the directory `/usr/local/lib` is found.

```
...in project ...
<property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
<path id="jaxp" location="${jaxp.jar}"/>
...in target ...
<available classname="javax.xml.transform.Transformer"
           classpathref="jaxp" property="jaxp11.present"/>
```

sets the `jaxp11.present` property to the value "true" if the class `javax.xml.transform.Transformer` is found in the classpath referenced by `jaxp` (in this case, `./lib/jaxp11/jaxp.jar`).

```
<available property="have.extras" resource="extratasks.properties">
  <classpath>
    <pathelement location="/usr/local/ant/extra.jar" />
  </classpath>
</available>
```

sets the `have.extras` property to the value "true" if the resource-file `extratasks.properties` is found.

Basename

Description

Task to determine the basename of a specified file, optionally minus a specified suffix.

When this task executes, it will set the specified property to the value of the last path element of the specified file. If `file` is a directory, the basename will be the last directory element. If `file` is a full-path, relative-path, or simple filename, the basename will be the simple file name, without any directory elements.

Parameters

Attribute	Description	Required
file	The path to take the basename of.	Yes
property	The name of the property to set.	Yes
suffix	The suffix to remove from the resulting basename (specified either with or without the ".").	No

Examples

```
<basename property="jar.filename" file="${lib.jarfile}"/>
```

will set `jar.filename` to `myjar.jar`, if `lib.jarfile` is defined as either a full-path filename (eg., `/usr/local/lib/myjar.jar`), a relative-path filename (eg., `lib/myjar.jar`), or a simple filename (eg., `myjar.jar`).

```
<basename property="cmdname" file="D:/usr/local/foo.exe"
          suffix=".exe"/>
```

will set `cmdname` to `foo`.

```
<property environment="env"/>
<basename property="temp.dirname" file="${env.TEMP}"/>
```

will set `temp.dirname` to the last directory element of the path defined for the `TEMP` environment variable.

BuildNumber

Description

This is a basic task that can be used to track build numbers.

It will first attempt to read a build number from a file (by default, `build.number` in the current directory), then set the property `build.number` to the value that was read in (or to `0`, if no such value). It will then increment the number by one and write it back out to the file. (See the PropertyFile task if you need finer control over things such as the property name or the number format.)

Parameters

Attribute	Description	Required
file	The file to read and write the build number from/to.	No; defaults to "build.number"

Examples

```
<buildnumber/>
```

Read, increment, and write a build number to the default file, `build.number`.

```
<buildnumber file="mybuild.number"/>
```

Read, increment, and write a build number to the file `mybuild.number`.

Condition

Description

Sets a property if a certain condition holds true - this is a generalization of Available and Uptodate.

If the condition holds true, the property value is set to true by default; otherwise, the property is not set. You can set the value to something other than the default by specifying the `value` attribute.

Conditions are specified as nested elements, you must specify exactly one condition.

Parameters

Attribute	Description	Required
property	The name of the property to set.	Yes
value	The value to set the property to. Defaults to "true".	No
else	The value to set the property to if the condition evaluates to *false*. By default the property will remain unset. *Since Ant 1.6.3*	No

Parameters specified as nested elements

All conditions to test are specified as nested elements, for a complete list see here.

Examples

```
<condition property="javamail.complete">
  <and>
    <available classname="javax.activation.DataHandler"/>
    <available classname="javax.mail.Transport"/>
  </and>
</condition>
```

sets the property `javamail.complete` if both the JavaBeans Activation Framework and JavaMail are available in the classpath.

```
<condition property="isMacOsButNotMacOsX">
  <and>
    <os family="mac"/>

    <not>
      <os family="unix"/>

    </not>
  </and>
</condition>
```

sets the property `isMacOsButNotMacOsX` if the current operating system is MacOS, but not MacOS X - which Ant considers to be in the Unix family as well.

```
<condition property="isSunOSonSparc">
  <os name="SunOS" arch="sparc"/>

</condition>
```

sets the property `isSunOSonSparc` if the current operating system is SunOS and if it is running on a sparc architecture.

Uptodate

Description

Sets a property if a target file or set of target files is more up-to-date than a source file or set of source files. A single source file is specified using the `srcfile` attribute. A set of source files is specified using the nested `<srcfiles>` elements. These are FileSets, whereas multiple target files are specified using a nested `<mapper>` element.

By default, the value of the property is set to `true` if the timestamp of the source file(s) is not more recent than the timestamp of the corresponding target file(s). You can set the value to something other than the default by specifying the `value` attribute.

If a `<srcfiles>` element is used, without also specifying a `<mapper>` element, the default behavior is to use a merge mapper, with the `to` attribute set to the value of the `targetfile` attribute.

Normally, this task is used to set properties that are useful to avoid target execution depending on the relative age of the specified files.

Parameters

Attribute	Description	Required
property	The name of the property to set.	Yes
value	The value to set the property to.	No; defaults to `true`.
srcfile	The file to check against the target file(s).	Yes, unless a nested `<srcfiles>` or `<srcresources>` element is present.
targetfile	The file for which we want to determine the status.	Yes, unless a nested `<mapper>` element is present.

Parameters specified as nested elements

srcfiles

The nested `<srcfiles>` element is a fileset and allows you to specify a set of files to check against the target file(s).

Note: You can specify either the `srcfile` attribute or nested `<srcfiles>` elements, but not both.

Note that the task will completely ignore any directories that seem to be matched by the srcfiles fileset, it will only consider normal files. If you need logic that applies to directories as well, use a nested srcresource and a dirset (for example).

srcresources

The nested `<srcresources>` element is a union and allows you to specify a collection of resources to check against the target file(s). *Since Ant 1.7*

mapper

The nested `<mapper>` element allows you to specify a set of target files to check for being up-to-date with respect to a

set of source files.

The mapper "to" attribute is relative to the target file, or to the "dir" attribute of the nested srcfiles element.

Since Ant 1.6.3, one can use a filenamemapper type in place of the mapper element.

Examples

```
  <uptodate property="xmlBuild.notRequired" targetfile="${deploy}\xmlClasses.jar" >
    <srcfiles dir= "${src}/xml" includes="**/*.dtd"/>
  </uptodate>
```

sets the property `xmlBuild.notRequired` to `true` if the `${deploy}/xmlClasses.jar` file is more up-to-date than any of the DTD files in the `${src}/xml` directory.

This can be written as:

```
  <uptodate property="xmlBuild.notRequired">
    <srcfiles dir= "${src}/xml" includes="**/*.dtd"/>
    <mapper type="merge" to="${deploy}\xmlClasses.jar"/>
  </uptodate>
```

as well. The `xmlBuild.notRequired` property can then be used in a `<target>` tag's `unless` attribute to conditionally run that target. For example, running the following target:

```
<target name="xmlBuild" depends="chkXmlBuild" unless="xmlBuild.notRequired">
  ...
</target>
```

will first run the `chkXmlBuild` target, which contains the `<uptodate>` task that determines whether `xmlBuild.notRequired` gets set. The property named in the `unless` attribute is then checked for being set/not set. If it did get set (ie., the jar file is up-to-date), then the `xmlBuild` target won't be run.

The following example shows a single source file being checked against a single target file:

```
  <uptodate property="isUpToDate"
            srcfile="/usr/local/bin/testit"
            targetfile="${build}/.flagfile"/>
```

sets the property `isUpToDate` to `true` if `/usr/local/bin/testit` is not newer than `${build}/.flagfile`.

The following shows usage of a relative mapper.

```
    <uptodate property="checkUptodate.uptodate">
      <srcfiles dir="src" includes="*"/>
      <mapper type="merge" to="../dest/output.done"/>
    </uptodate>
    <echo message="checkUptodate result: ${checkUptodate.uptodate}"/>
```

The previous example can be a bit confusing, so it may be better to use absolute paths:

```
    <property name="dest.dir" location="dest"/>
    <uptodate property="checkUptodate.uptodate">
      <srcfiles dir="src" includes="*"/>
      <mapper type="merge" to="${dest.dir}/output.done"/>
    </uptodate>
```

Dirname

Description

Task to determine the directory path of a specified file.

When this task executes, it will set the specified property to the value of the specified file (or directory) up to, but not including, the last path element. If the specified file is a path that ends in a filename, the filename will be dropped. If the specified file is just a filename, the directory will be the current directory.

Note: This is not the same as the UNIX dirname command, which is defined as "strip non-directory suffix from filename". <dirname> determines the full directory path of the specified file.

Parameters

Attribute	Description	Required
file	The path to take the dirname of.	Yes
property	The name of the property to set.	Yes

Examples

```
  <dirname property="antfile.dir" file="${ant.file}"/>
```

will set `antfile.dir` to the directory path for `${ant.file}`.

```
  <dirname property="foo.dirname" file="foo.txt"/>
```

will set `foo.dirname` to the project's basedir.

echoproperties

Description

Displays all the current properties (or a subset of them specified by a nested `<propertyset>`) in the project. The output can be sent to a file if desired. This task can be used as a somewhat contrived means of returning data from an `<ant>` invocation, but is really for debugging build files.

Parameters

Attribute	Description	Required
destfile	If specified, the value indicates the name of the file to send the output of the statement to. The generated output file is compatible for loading by any Java application as a property file. If not specified, then the output will go to the Ant log.	No
prefix	a prefix which is used to filter the properties only those properties starting with this prefix will be echoed.	No
regex	a regular expression which is used to filter the properties only those properties whose names match it will be echoed.	No
failonerror	By default, the "failonerror" attribute is enabled. If an error occurs while writing the properties to a file, and this attribute is enabled, then a BuildException will be thrown, causing the build to fail. If disabled, then IO errors will be reported as a log statement, and the build will continue without failure from this task.	No
format	One of `text` or `xml`. Determines the output format. Defaults to `text`.	No

Parameters specified as nested elements

propertyset

You can specify subsets of properties to be echoed with propertysets. Using `propertysets` gives more control on which properties will be picked up. The attributes `prefix` and `regex` are just shorcuts that use `propertysets` internally.

since Ant 1.6.

Examples

```
<echoproperties/>
```

Report the current properties to the log.

```
<echoproperties destfile="my.properties"/>
```

Report the current properties to the file "my.properties", and will fail the build if the file could not be created or written to.

```
<echoproperties destfile="my.properties" failonerror="false"/>
```

Report the current properties to the file "my.properties", and will log a message if the file could not be created or

written to, but will still allow the build to continue.

```
<echoproperties prefix="java."/>
```

List all properties beginning with "java."

```
<echoproperties>
  <propertyset>
    <propertyref prefix="java."/>
  </propertyset>
</echoproperties>
```

This again lists all properties beginning with "java." using a nested `</propertyset>` which is an equivalent but longer way.

```
<echoproperties regex=".*ant.*"/>
```

Lists all properties that contain "ant" in their names. The equivalent snippet with `</propertyset>` is:

```
<echoproperties>
  <propertyset>
    <propertyref regex=".*ant.*"/>
  </propertyset>
</echoproperties>
```

LoadFile

Description

Specialization of loadresource that works on files exclusively and provides a srcFile attribute for convenience. Unless an encoding is specified, the encoding of the current locale is used.

If the resource content is empty (maybe after processing a filterchain) the property is not set.

Parameters

Attribute	Description	Required
srcFile	source file	Yes
property	property to save to	Yes
encoding	encoding to use when loading the file	No
failonerror	Whether to halt the build on failure	No, default "true"
quiet	Do not display a diagnostic message (unless Ant has been invoked with the `-verbose` or `-debug` switches) or modify the exit status to reflect an error. Setting this to "true" implies setting failonerror to "false". *Since Ant 1.7.0.*	No, default "false"

The LoadFile task supports nested FilterChains.

Examples

```
    <loadfile property="message"
      srcFile="message.txt"/>
```

Load file message.txt into property "message"; an `<echo>` can print this. This is identical to

```
    <loadresource property="message">
       <file file="message.txt"/>
    </loadresource>
```

```
    <loadfile property="encoded-file"
      srcFile="loadfile.xml"
      encoding="ISO-8859-1"/>
```

Load a file using the latin-1 encoding

```
    <loadfile
      property="optional.value"
      srcFile="optional.txt"
      failonerror="false"/>
```

Load a file, don't fail if it is missing (a message is printed, though)

```
    <loadfile
      property="mail.recipients"
      srcFile="recipientlist.txt">
      <filterchain>
        <striplinebreaks/>
      </filterchain>
    </loadfile>
```

Load a property which can be used as a parameter for another task (in this case mail), merging lines to ensure this happens.

```
    <loadfile
      property="system.configuration.xml"
      srcFile="configuration.xml">
        <filterchain>
          <expandproperties/>
        </filterchain>
    </loadfile>
```

Load an XML file into a property, expanding all properties declared in the file in the process.

LoadProperties

Description

Load a file's contents as Ant properties. This is equivalent to `<property file|resource="..."/>` except that it supports nested `<filterchain>` elements. Also if the file is missing, the build is halted with an error, rather than a warning being printed.

If you want to simulate property's prefix attribute, please use prefixlines filter.

Parameters

Attribute	Description	Required
srcFile	source file	One of these or a nested resource
resource	the resource name of the property file	
encoding	encoding to use when loading the file	No
classpath	the classpath to use when looking up a resource.	No
classpathref	the classpath to use when looking up a resource, given as reference to a `<path>` defined elsewhere..	No
prefix	Prefix to apply to loaded properties; a "." is appended to the prefix if not specified. *Since Ant 1.8.1*	No

Parameters specified as nested elements

any resource or single element resource collection

The specified resource will be used as src. *Since Ant 1.7*

FilterChain

classpath

for use with the *resource* attribute.

Examples

```
    <loadproperties srcFile="file.properties"/>
```

or

```
    <loadproperties>
      <file file="file.properties"/>
    </loadproperties>
```

Load contents of file.properties as Ant properties.

```
    <loadproperties srcFile="file.properties">
      <filterchain>
        <linecontains>
          <contains value="import."/>
```

```
        </linecontains>
      </filterchain>
    </loadproperties>
```

Read the lines that contain the string "import." from the file "file.properties" and load them as Ant properties.

```
    <loadproperties>
      <gzipresource>
        <url url="http://example.org/url.properties.gz"/>
      </gzipresource>
    </loadproperties>
```

Load contents of http://example.org/url.properties.gz, uncompress it on the fly and load the contents as Ant properties.

Makeurl Task

Description

This task takes one or more filenames and turns them into URLs, which it then assigns to a property. Useful when setting up RMI or JNLP codebases, for example. Nested filesets are supported; if present, these are turned into the URLs with the supplied separator between them (default: space).

Examples:

```
<makeurl file="${user.home}/.m2/repository" property="m2.repository.url"/>
```

Sets the property `m2.repository.url` to the file: URL of the local Maven2 repository.

```
<makeurl property="codebase"><fileset dir="lib includes="*.jar"/></makeurl>
```

Set the property `codebase` to the three URLs of the files provided as nested elements.

Parameters

Attribute	Description	Type	Requirement
file	name of a file to be converted into a URL	File	optional, if a nested fileset or path is supplied
property	name of a property to set to the URL	String	required
separator	separator for the multi-URL option	String	optional
validate	validate that every named file exists	boolean	optional; default: true

Parameters as nested elements

fileset (org.apache.tools.ant.types.FileSet)

A fileset of JAR files to include in the URL list, each separated by the separator.

path (org.apache.tools.ant.types.Path)

Add a path to the URL. All elements in the path will be converted to individual URL entries.

Pathconvert

Description

Converts nested ResourceCollections, or a reference to just one, into a path form for a particular platform, optionally storing the result into a given property. It can also be used when you need to convert a Resource Collection into a list, separated by a given character, such as a comma or space, or, conversely, e.g. to convert a list of files in a FileList into a path.

Nested `<map>` elements can be specified to map Windows drive letters to Unix paths, and vice-versa.

More complex transformations can be achieved using a nested `<mapper>` (since Ant 1.6.2).

Parameters

Attribute	Description	Required
targetos	The target architecture. Must be one of 'unix', 'windows', 'netware', 'tandem' or 'os/2'. This is a shorthand mechanism for specifying both `pathsep` and `dirsep` according to the specified target architecture.	No
dirsep	The character(s) to use as the directory separator in the generated paths.	No, defaults to current JVM `File.separator`
pathsep	The character(s) to use as the path-element separator in the generated paths.	No, defaults to current JVM `File.pathSeparator`
property	The name of the property in which to place the converted path.	No, result will be logged if unset
refid	What to convert, given as a reference to a `<path>`, `<fileset>`, `<dirset>`, or `<filelist>` defined elsewhere	No; if omitted, a nested `<path>` element must be supplied.
setonempty	Should the property be set, even if the result is the empty string?	No; default is "true".
preserveduplicates	Whether to preserve duplicate resources. **Since Ant 1.8**	No; default "false".

Parameters specified as nested elements

map

Specifies the mapping of path prefixes between Unix and Windows.

Attribute	Description	Required
from	The prefix to match. Note that this value is case-insensitive when the build is running on a Windows platform and case-sensitive when running on a Unix platform. *Since Ant 1.7.0*, on Windows this value is also insensitive to the slash style used for directories, one can use '/' or '\'.	Yes
to	The replacement text to use when `from` is matched.	Yes

Each map element specifies a single replacement map to be applied to the elements of the path being processed. If no

map entries are specified, then no path prefix mapping is performed.

Note: The map elements are applied in the order specified, and only the first matching map element is applied. So, the ordering of your map elements can be important, if any `from` values are prefixes of other `from` values.

Resource Collections

If the `refid` attribute is not specified, then one or more nested Resource Collections must be supplied.

mapper

A single nested `<mapper>` element can be specified to perform any of various filename transformations (since Ant 1.6.2).

Examples

In the examples below, assume that the `${wl.home}` property has the value `d:\weblogic`, and `${wl.home.unix}` has the value `/weblogic`.

Example 1

```
    <path id="wl.path">
      <pathelement location="${wl.home}/lib/weblogicaux.jar"/>
      <pathelement location="${wl.home}/classes"/>
      <pathelement location="${wl.home}/mssqlserver4/classes"/>
      <pathelement location="c:\winnt\System32"/>
    </path>

    <pathconvert targetos="unix" property="wl.path.unix" refid="wl.path">
      <map from="${wl.home}" to="${wl.home.unix}"/>
      <map from="c:" to=""/>
    </pathconvert>
```

will generate the path shown below and store it in the property named `wl.path.unix`.

```
/weblogic/lib/weblogicaux.jar:/weblogic/classes:/weblogic/mssqlserver4/classes:/WINNT/SYSTEM32
```

Example 2

Given a FileList defined as:

```
  <filelist id="custom_tasks.jars"
        dir="${env.HOME}/ant/lib"
        files="njavac.jar,xproperty.jar"/>
```

then:

```
    <pathconvert targetos="unix" property="custom_tasks.jars" refid="custom_tasks.jars">
      <map from="${env.HOME}" to="/usr/local"/>
    </pathconvert>
```

will convert the list of files to the following Unix path:

```
/usr/local/ant/lib/njavac.jar:/usr/local/ant/lib/xproperty.jar
```

Example 3

```
    <fileset dir="${src.dir}" id="src.files">
      <include name="**/*.java"/>
    </fileset>
```

```
    <pathconvert pathsep="," property="javafiles" refid="src.files"/>
```

This example takes the set of files determined by the fileset (all files ending in `.java`), joins them together separated by commas, and places the resulting list into the property `javafiles`. The directory separator is not specified, so it defaults to the appropriate character for the current platform. Such a list could then be used in another task, like `javadoc`, that requires a comma separated list of files.

Example 4

```
    <pathconvert property="prop" dirsep="|">
      <map from="${basedir}/abc/" to=''/>
      <path location="abc/def/ghi"/>
    </pathconvert>
```

This example sets the property "prop" to "def|ghi" on Windows and on Unix.

PropertyFile

Introduction

Ant provides an optional task for editing property files. This is very useful when wanting to make unattended modifications to configuration files for application servers and applications. Currently, the task maintains a working property file with the ability to add properties or make changes to existing ones. Since Ant 1.8.0 comments and layout of the original properties file are preserved.

PropertyFile Task

Parameters

Attribute	Description	Required
file	Location of the property file to be edited	Yes
comment	Header for the file itself	no
jdkproperties	Use java.lang.Properties, which will loose comments and layout of file (default is 'false'). *since Ant 1.8.0*	no

The boolean attribute 'jdkproperties' is provided to recover the previous behaviour of the task, in which the layout and any comments in the properties file were lost by the task.

Parameters specified as nested elements

Entry

Use nested `<entry>` elements to specify actual modifications to the property file itself.

Attribute	Description	Required
key	Name of the property name/value pair	Yes
value	Value to set (=), to add (+) or subtract (-)	At least one must be specified, if *operation* is not *delete*
default	Initial value to set for a property if it is not already defined in the property file. For type date, an additional keyword is allowed: "now"	
type	Regard the value as : int, date or string (default)	No
operation	One of the following operations: **for all datatypes:** • "del" : deletes an entry • "+" : adds a value to the existing value	No

	"=" : sets a value instead of the existing value (default) **for date and int only:** • "-" : subtracts a value from the existing value	
pattern	For int and date type only. If present, Values will be parsed and formatted accordingly.	No
unit	The unit of the value to be applied to date +/- operations. Valid Values are: • millisecond • second • minute • hour • day (default) • week • month • year This only applies to date types using a +/- operation.	No

The rules used when setting a property value are shown below. The operation occurs **after** these rules are considered.

- If only value is specified, the property is set to it regardless of its previous value.
- If only default is specified and the property previously existed in the property file, it is unchanged.
- If only default is specified and the property did not exist in the property file, the property is set to default.
- If value and default are both specified and the property previously existed in the property file, the property is set to value.
- If value and default are both specified and the property did not exist in the property file, the property is set to default.

Examples

The following changes the my.properties file. Assume my.properties look like:

```
# A string value
akey=original value

# The following is a counter, which will be incremented by 1 for
# each time the build is run.
anint=1
```

After running, the file would now look like

```
#My properties
#Wed Aug 31 13:47:19 BST 2005
# A string value
akey=avalue

# The following is a counter, which will be incremented by 1 for
# each time the build is run.
anint=2

adate=2005/08/31 13\:47
```

```
formated.int=0014

formated.date=243 13\:47
```

The slashes conform to the expectations of the Properties class. The file will be stored in a manner so that each character is examined and escaped if necessary.

The layout and comment of the original file is preserved. New properties are added at the end of the file. Existing properties are overwritten in place.

```
<propertyfile
    file="my.properties"
    comment="My properties">
  <entry  key="akey" value="avalue"/>
  <entry  key="adate" type="date" value="now"/>
  <entry  key="anint" type="int" default="0" operation="+"/>
  <entry  key="formated.int" type="int" default="0013" operation="+" pattern="0000"/>
  <entry  key="formated.date" type="date" value="now" pattern="DDD HH:mm"/>
</propertyfile>
```

To produce dates relative from today :

```
<propertyfile
    file="my.properties"
    comment="My properties">
  <entry  key="formated.date-1"
      type="date" default="now" pattern="DDD"
      operation="-" value="1"/>
  <entry  key="formated.tomorrow"
      type="date" default="now" pattern="DDD"
      operation="+" value="1"/>
</propertyfile>
```

Concatenation of strings :

```
<propertyfile
    file="my.properties"
    comment="My properties">
  <entry  key="progress" default="" operation="+" value="."/>
</propertyfile>
```

Each time called, a "." will be appended to "progress"

Whichresource

Description

Find a class or resource on the supplied classpath, or the system classpath if none is supplied. The named property is set if the item can be found. For example:

```
<whichresource resource="/log4j.properties" property="log4j.url" >
```

Parameters

Attribute	Description	Required
property	The property to fill with the URL of the resource of class.	Yes
class	The name of the class to look for.	Exactly one of these.
resource	The name of the resource to look for.	
classpath	The classpath to use when looking up `class` or `resource`.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere. *Since Ant 1.7.1.*	No

Parameters specified as nested elements

classpath

`Whichresource`'s `classpath` attribute is a path-like structure and can also be set via a nested `<classpath>` element.

Examples

The following shows using a classpath reference.

```
<path id="bsf.classpath">
  <fileset dir="${user.home}/lang/bsf" includes="*.jar"/>
</path>
<whichresource property="bsf.class.location"
               class="org.apache.bsf.BSFManager"
               classpathref="bsf.classpath"/>
<echo>${bsf.class.location}</echo>
```

The following shows using a nested classpath.

```
<whichresource
  property="ant-contrib.antlib.location"
  resource="net/sf/antcontrib/antlib.xml">
  <classpath>
    <path path="f:/testing/ant-contrib/target/ant-contrib.jar"/>
  </classpath>
</whichresource>
<echo>${ant-contrib.antlib.location}</echo>
```

XmlProperty

Description

Loads property values from a well-formed xml file. There are no other restrictions than "well-formed". You can choose the layout you want. For example this XML property file:

```
<root>
  <properties>
    <foo>bar</foo>
  </properties>
</root>
```

is roughly equivalent to this Java property file:

```
root.properties.foo = bar
```

By default, this load does *no* processing of the input. In particular, unlike the Property task, property references (i.e., `${foo}`) are not resolved.

Semantic Attributes

Input processing can be enabled by using the **semanticAttributes** attribute. If this attribute is set to *true* (its default is *false*), the following processing occurs as the input XML file is loaded:

- Property references are resolved.
- The following attributes are treated differently:
  - **id**: The property is associated with the given id value.
  - **location**: The property is treated as a file location
  - **refid**: The property is set to the value of the referenced property.
  - **value**: The property is set to the value indicated.
- Path-like Structures can be defined by use of the following attributes:
  - **pathid**: The given id is used to identify a path. The nested XML tag name is ignored. Child elements can be used (XML tag names are ignored) to identify elements of the path.

For example, with semantic attribute processing enabled, this XML property file:

```
<root>
  <properties>
    <foo location="bar"/>
    <quux>${root.properties.foo}</quux>
  </properties>
</root>
```

is roughly equivalent to the following fragments in a build.xml file:

```
<property name="root.properties.foo" location="bar"/>
<property name="root.properties.quux" value="${root.properties.foo}"/>
```

Parameters

Attribute	Description	Required
file	The XML file to parse.	Yes, or a nested resource collection.

prefix	The prefix to prepend to each property	No
keepRoot	Keep the xml root tag as the first value in the property name.	No, default is *true*.
validate	Validate the input file (e.g. by a DTD). Otherwise the XML must only be well-formed.	No, default is *false*.
collapseAttributes	Treat attributes as nested elements.	No, default is *false*.
semanticAttributes	Enable special handling of certain attribute names. See the Semantic Attributes section for more information.	No, default is *false*.
includeSemanticAttribute	Include the semantic attribute name as part of the property name. Ignored if *semanticAttributes* is not set to *true*. See the Semantic Attributes section for more information.	No, default is *false*.
rootDirectory	The directory to use for resolving file references. Ignored if *semanticAttributes* is not set to *true*.	No, default is *${basedir}*.
delimiter	Delimiter for splitting multiple values. *since Ant 1.7.1*	No, defaults to comma

Nested Elements

xmlcatalog

The `<xmlcatalog>` element is used to perform entity resolution.

any resource or single element resource collection

The specified resource will be used as input.

Examples

Non-semantic Attributes

Here is an example xml file that does not have any semantic attributes.

```
<root-tag myattr="true">
 <inner-tag someattr="val">Text</inner-tag>
 <a2><a3><a4>false</a4></a3></a2>
</root-tag>
```

default loading

This entry in a build file:

```
<xmlproperty file="somefile.xml"/>
```

is equivalent to the following properties:

```
root-tag(myattr)=true
root-tag.inner-tag=Text
root-tag.inner-tag(someattr)=val
root-tag.a2.a3.a4=false
```

collapseAttributes=false

This entry in a build file:

```
   <xmlproperty file="somefile.xml" collapseAttributes="true"/>
```

is equivalent to the following properties:

```
   root-tag.myattr=true
   root-tag.inner-tag=Text
   root-tag.inner-tag.someatt=val
   root-tag.a2.a3.a4=false
```

keepRoot=false

This entry in a build file:

```
   <xmlproperty file="somefile.xml" keepRoot="false"/>
```

is equivalent to the following properties:

```
   inner-tag=Text
   inner-tag(someattr)=val
   a2.a3.a4=false
```

Semantic Attributes

Here is an example xml file that has semantic attributes.

```
  <root-tag>
    <version value="0.0.1"/>
    <build folder="build">
      <classes id="build.classes" location="${build.folder}/classes"/>
      <reference refid="build.classes"/>
    </build>
    <compile>
      <classpath pathid="compile.classpath">
        <pathelement location="${build.classes}"/>
      </classpath>
    </compile>
    <run-time>
      <jars>*.jar</jars>
      <classpath pathid="run-time.classpath">
        <path refid="compile.classpath"/>
        <pathelement path="${run-time.jars}"/>
      </classpath>
    </run-time>
  </root-tag>
```

default loading (semanticAttributes=true)

This entry in a build file:

```
   <xmlproperty file="somefile.xml" keepRoot="false"
                semanticAttributes="true"/>
```

is equivalent to the following entries in a build file:

```
  <property name="version" value="0.0.1"/>
  <property name="build.folder" value="build"/>
  <property name="build.classes" location="${build.folder}/classes" id="build.classes"/>
  <property name="build.reference" refid="build.classes"/>

  <property name="run-time.jars" value="*.jar"/>

  <path id="compile.classpath">
    <pathelement location="${build.classes}"/>
  </path>

  <path id="run-time.classpath">
    <path refid="compile.classpath"/>
```

```
    <pathelement path="${run-time.jars}"/>
  </path>
```

includeSemanticAttribute="true"

This entry in a build file:

```
   <xmlproperty file="somefile.xml"
                semanticAttributes="true" keepRoot="false"
                includeSemanticAttribute="true"/>
```

is equivalent to the following entries in a build file:

```
  <property name="version.value" value="0.0.1"/>
  <property name="build.folder" value="build"/>
  <property name="build.classes.location" location="${build.folder}/classes"/>
  <property name="build.reference.refid" refid="build.classes"/>

  <property name="run-time.jars" value="*.jar"/>

  <path id="compile.classpath">
    <pathelement location="${build.classes}"/>
  </path>

  <path id="run-time.classpath">
    <path refid="compile.classpath"/>
    <pathelement path="${run-time.jars}"/>
  </path>
```

FTP

Description

The ftp task implements a basic FTP client that can send, receive, list, delete files, and create directories. See below for descriptions and examples of how to perform each task.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information. *Get the latest version of this library, for the best support in Ant*

The ftp task attempts to determine what file system is in place on the FTP server. Supported server types are Unix, NT, OS2, VMS, and OS400. In addition, NT and OS400 servers which have been configured to display the directory in Unix style are also supported correctly. Otherwise, the system will default to Unix standards. *remotedir* must be specified in the exact syntax required by the ftp server. If the usual Unix conventions are not supported by the server, *separator* can be used to set the file separator that should be used instead.

See the section on directory based tasks, on how the inclusion/exclusion of files works, and how to write patterns.

This task does not currently use the proxy information set by the `<setproxy>` task, and cannot go through a firewall via socks.

Warning: there have been problems reported concerning the ftp get with the `newer` attribute. Problems might be due to format of ls -l differing from what is expected by commons-net, for instance due to specificities of language used by the ftp server in the directory listing. If you encounter such a problem, please send an email including a sample directory listing coming from your ftp server (ls -l on the ftp prompt).

If you can connect but not upload or download, try setting the `passive` attribute to true to use the existing (open) channel, instead of having the server try to set up a new connection.

Parameters

Attribute	Description	Required
server	the address of the remote ftp server.	Yes
port	the port number of the remote ftp server. Defaults to port 21.	No
userid	the login id to use on the ftp server.	Yes
password	the login password to use on the ftp server.	Yes
account	the account to use on the ftp server. *since Ant 1.7.*	No
remotedir	remote directory on the ftp server see table below for detailed usage	No
action	the ftp action to perform, defaulting to "send". Currently supports "put", "get", "del", "list", "chmod", "mkdir", "rmdir", and "site".	No
binary	selects binary-mode ("yes") or text-mode ("no") transfers. Defaults to "yes"	No
passive	selects passive-mode ("yes") transfers, for better through-firewall connectivity, at the price of performance. Defaults to "no"	No
verbose	displays information on each file transferred if set to "yes". Defaults to "no".	No

depends	transfers only new or changed files if set to "yes". Defaults to "no".	No
newer	a synonym for *depends*. see timediffauto and timediffmillis	No
timediffauto	set to `"true"` to make ant calculate the time difference between client and server. *requires write access in the remote directory* Since ant 1.6	No
timestampGranularity	Specify either `MINUTE`, `NONE`, (or you may specify `""` which is equivalent to not specifying a value, useful for property-file driven scripts). Allows override of the typical situation in PUT and GET where local filesystem timestamps are `HH:mm:ss` and the typical FTP server's timestamps are `HH:mm`. This can throw off uptodate calculations. However, the default values should suffice for most applications. Since ant 1.7	No. Only applies in "puts" and "gets" where the default values are `MINUTE` for PUT and `NONE` for GET. (It is not as necessary in GET because we have the **preservelastmodified** option.)
timediffmillis	**Deprecated**. Number of milliseconds to add to the time on the remote machine to get the time on the local machine. The **timestampGranularity** attribute (for which the default values should suffice in most situations), and the **serverTimeZoneConfig** option, should make this unnecessary. **serverTimeZoneConfig** does the math for you and also knows about Daylight Savings Time. Since ant 1.6	No
separator	sets the file separator used on the ftp server. Defaults to "/".	No
umask	sets the default file permissions for new files, unix only.	No
chmod	sets or changes file permissions for new or existing files, unix only. If used with a put action, chmod will be issued for each file.	No
listing	the file to write results of the "list" action. Required for the "list" action, ignored otherwise.	No
ignoreNoncriticalErrors	flag which permits the task to ignore some non-fatal error codes sent by some servers during directory creation: wu-ftp in particular. Default: false	No
skipFailedTransfers	flag which enables unsuccessful file put, delete and get operations to be skipped with a warning and the remainder of the files still transferred. Default: false	No
preservelastmodified	Give the copied files the same last modified time as the original source files (applies to getting files only). (*Note*: Ignored on Java 1.1)	No; defaults to false.
retriesAllowed	Set the number of retries allowed on an file-transfer operation. If a number > 0 specified, each file transfer can fail up to that many times before the operation is failed. If -1 or "forever" specified, the operation will keep trying until it succeeds.	No; defaults to 0
siteCommand	Set the server-specific SITE command to execute if the `action` attribute has been specified as `"site"`.	No
initialSiteCommand	Set a server-specific SITE command to execute immediately after login.	No

enableRemoteVerification	Whether data connection sshould be verified to connect to the same host as the control connection. This is a security measure that is enabled by default, but it may be useful to disable it in certain firewall scenarios. *since Ant 1.8.0*	No, default is true

The following attributes require jakarta-commons-net-1.4.0 or greater.

Use these options when the standard options don't work, because

- the server is in a different timezone and you need timestamp dependency checking
- the default timestamp formatting doesn't match the server display and list parsing therefore fails

If none of these is specified, the default mechanism of letting the system auto-detect the server OS type based on the FTP SYST command and assuming standard formatting for that OS type will be used.

To aid in property-file-based development where a build script is configured with property files, for any of these attributes, a value of `""` is equivalent to not specifying it.

Please understand that these options are incompatible with the autodetection scheme. If any of these options is specified, (other than with a value of `""`) a system type must be chosen and if systemTypeKey is not specified, UNIX will be assumed. The philosophy behind this is that these options are for setting non-standard formats, and a build-script author who knows what system he is dealing with will know what options to need to be set. Otherwise, these options should be left alone and the default autodetection scheme can be used and will work in the majority of cases.

systemTypeKey	Specifies the type of system in use on the server. Supported values are `"UNIX", "VMS", "WINDOWS", "OS/2", "OS/400", "MVS"`. If not specified, (or specified as `""`) and if no other xxxConfig attributes are specified, the autodectection mechanism based on the FTP SYST command will be used. Since ant 1.7	No, but if any of the following xxxConfig attributes is specified, UNIX will be assumed, even if `""` is specified here.
serverTimeZoneConfig	Specify as a Java TimeZone identifier, (e.g. `GMT`, `America/Chicago` or `Asia/Jakarta`) the timezone used by the server for timestamps. This enables timestamp dependency checking even when the server is in a different time zone from the client. Time Zones know, also, about daylight savings time, and do not require you to calculate milliseconds of difference. If not specified, (or specified as `""`), the time zone of the client is assumed. Since ant 1.7	No
defaultDateFormatConfig	Specify in Java SimpleDateFormat notation, (e.g. `yyyy-MM-dd`), the date format generally used by the FTP server to parse dates. In some cases this will be the only date format used. In others, (unix for example) this will be used for dates older than a year old. (See **recentDateFormatConfig**). If not specified, (or specified as `""`), the default date format for the system type indicated by the **systemTypeKey** attribute will be used. Since ant 1.7	No.
recentDateFormatConfig	Specify in Java SimpleDateFormat notation, (e.g. `MMM dd hh:mm`) the date format used by the FTP server to parse dates less than a year old. If not specified (or specified as `""`), and if the system type indicated by the system key uses a recent date format, its standard format will be used. Since ant 1.7	No

serverLanguageCodeConfig	a two-letter ISO-639 language code used to specify the language used by the server to format month names. This only needs to be specified when the server uses non-numeric abbreviations for months in its date listings in a language other than English. This appears to be becoming rarer and rarer, as commonly distributed ftp servers seem increasingly to use English or all-numeric formats. Languages supported are: • en - English • fr - French • de - German • it - Italian • es - Spanish • pt - Portuguese • da - Danish • sv - Swedish • no - Norwegian • nl - Dutch • ro - Romanian • sq - Albanian • sh - Serbo-croatian • sk - Slovak • sl - Slovenian If you require a language other than the above, see also the **shortMonthNamesConfig** attribute. Since ant 1.7	No
shortMonthNamesConfig	specify the month abbreviations used on the server in file timestamp dates as a pipe-delimited string for each month. For example, a set of month names used by a hypothetical Icelandic FTP server might conceivably be specified as `"jan|feb|mar|apr|maí|jún|júl|ágú|sep|okt|nóv|des"`. This attribute exists primarily to support languages not supported by the **serverLanguageCode** attribute. Since ant 1.7	No

Note about remotedir attribute

Action	meaning of `remotedir`	use of nested `fileset` (s)
send/put	base directory to which the files are sent	they are used normally and evaluated on the local machine
recv/get	base directory from which the files are retrieved	the remote files located under the `remotedir` matching the include/exclude patterns of the `fileset`
del/delete	base directory from which files get deleted	the remote files located under the `remotedir` matching the include/exclude patterns of the `fileset`
list	base directory from which files are listed	the remote files located under the `remotedir` matching the include/exclude patterns of the `fileset`
mkdir	directory to create	not used
		`remotedir`

chmod	base directory from which the mode of files get changed	the remote files located under the matching the include/exclude patterns of the `fileset`
rmdir	base directory from which directories get removed	the remote directories located under the `remotedir` matching the include/exclude patterns of the `fileset`

Parameters specified as nested elements

fileset

The ftp task supports any number of nested `<fileset>` elements to specify the files to be retrieved, or deleted, or listed, or whose mode you want to change.

The attribute `followsymlinks` of `fileset` is supported on local (put) as well as remote (get, chmod, delete) filesets. *Before ant 1.6 there was no support of symbolic links in remote filesets. In order to exclude symbolic links (preserve the behavior of ant 1.5.x and older), you need to explicitly set `followsymlinks` to `false`.* On remote filesets hidden files are not checked for being symbolic links. Hidden files are currently assumed to not be symbolic links.

Sending Files

The easiest way to describe how to send files is with a couple of examples:

```
  <ftp server="ftp.apache.org"
       userid="anonymous"
       password="me@myorg.com">
    <fileset dir="htdocs/manual"/>
  </ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and uploads all files in the `htdocs/manual` directory to the default directory for that user.

```
  <ftp server="ftp.apache.org"
       remotedir="incoming"
       userid="anonymous"
       password="me@myorg.com"
       depends="yes">
    <fileset dir="htdocs/manual"/>
  </ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and uploads all new or changed files in the `htdocs/manual` directory to the `incoming` directory relative to the default directory for `anonymous`.

```
  <ftp server="ftp.apache.org"
       port="2121"
       remotedir="/pub/incoming"
       userid="coder"
       password="java1"
       passive="yes"
       depends="yes"
       binary="no">
    <fileset dir="htdocs/manual">
      <include name="**/*.html"/>
    </fileset>
  </ftp>
```

Logs in to `ftp.apache.org` at port `2121` as `coder` with password `java1` and uploads all new or changed HTML files in the `htdocs/manual` directory to the `/pub/incoming` directory. The files are transferred in text mode. Passive mode has been switched on to send files from behind a firewall.

```
  <ftp server="ftp.hypothetical.india.org"
```

```
      port="2121"
      remotedir="/pub/incoming"
      userid="coder"
      password="java1"
      depends="yes"
      binary="no"
      systemTypeKey="Windows"
      serverTimeZoneConfig="India/Calcutta">
    <fileset dir="htdocs/manual">
      <include name="**/*.html"/>
    </fileset>
  </ftp>
```

Logs in to a Windows server at `ftp.hypothetical.india.org` at port `2121` as `coder` with password `java1` and uploads all new or changed (accounting for timezone differences) HTML files in the `htdocs/manual` directory to the `/pub/incoming` directory. The files are transferred in text mode.

```
  <ftp server="ftp.nt.org"
      remotedir="c:\uploads"
      userid="coder"
      password="java1"
      separator="\"
      verbose="yes">
    <fileset dir="htdocs/manual">
      <include name="**/*.html"/>
    </fileset>
  </ftp>
```

Logs in to the Windows-based `ftp.nt.org` as `coder` with password `java1` and uploads all HTML files in the `htdocs/manual` directory to the `c:\uploads` directory. Progress messages are displayed as each file is uploaded.

Getting Files

Getting files from an FTP server works pretty much the same way as sending them does. The only difference is that the nested filesets use the remotedir attribute as the base directory for the files on the FTP server, and the dir attribute as the local directory to put the files into. The file structure from the FTP site is preserved on the local machine.

```
  <ftp action="get"
      server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com">
    <fileset dir="htdocs/manual">
      <include name="**/*.html"/>
    </fileset>
  </ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and recursively downloads all .html files from default directory for that user into the `htdocs/manual` directory on the local machine.

```
  <ftp action="get"
      server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com"
      systemTypeKey="UNIX"
      defaultDateFormatConfig="yyyy-MM-dd HH:mm">
    <fileset dir="htdocs/manual">
      <include name="**/*.html"/>
    </fileset>
  </ftp>
```

If apache.org ever switches to a unix FTP server that uses the new all-numeric format for timestamps, this version would become necessary. It would accomplish the same functionality as the previous example but would successfully handle the numeric timestamps. The `systemTypeKey` is not necessary here but helps clarify what is going on.

```
  <ftp action="get"
      server="ftp.hypthetical.fr"
      userid="anonymous"
      password="me@myorg.com"
      defaultDateFormatConfig="d MMM yyyy"
```

```
      recentDateFormatConfig="d MMM HH:mm"
      serverLanguageCodeConfig="fr">
    <fileset dir="htdocs/manual">
      <include name="**/*.html"/>
  </fileset>
</ftp>
```

Logs into a UNIX FTP server at `ftp.hypothetical.fr` which displays dates with French names in Standard European format, as `anonymous`, and recursively downloads all .html files from default directory for that user into the `htdocs/manual` directory on the local machine.

Deleting Files

As you've probably guessed by now, you use nested fileset elements to select the files to delete from the remote FTP server. Again, the filesets are relative to the remote directory, not a local directory. In fact, the dir attribute of the fileset is ignored completely.

```
<ftp action="del"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com">
  <fileset>
    <include name="**/*.tmp"/>
  </fileset>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and tries to delete all *.tmp files from the default directory for that user. If you don't have permission to delete a file, a BuildException is thrown.

Listing Files

```
<ftp action="list"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com"
     listing="data/ftp.listing">
  <fileset>
    <include name="**"/>
  </fileset>
</ftp>
```

This provides a file listing in `data/ftp.listing` of all the files on the FTP server relative to the default directory of the `anonymous` user. The listing is in whatever format the FTP server normally lists files.

Creating Directories

Note that with the mkdir action, the directory to create is specified using the remotedir attribute.

```
<ftp action="mkdir"
     server="ftp.apache.org"
     userid="anonymous"
     password="me@myorg.com"
     remotedir="some/remote/dir"/>
```

This creates the directory `some/remote/dir` beneath the default root directory. As with all other actions, the directory separator character must be correct according to the desires of the FTP server.

Removing Directories

This action uses nested fileset elements to select the directories to remove from the remote FTP server. The filesets are relative to the remote directory, not a local directory. The dir attribute of the fileset is ignored completely. The

directories to be removed must be empty, or contain only other directories that have been also selected to be removed by the filesets patterns, otherwise a BuildException will be thrown. Also, if you don't have permission to remove a directory, a BuildException is thrown.

```
  <ftp action="rmdir"
       server="ftp.apache.org"
       userid="anonymous"
       password="me@myorg.com"
       remotedir="/somedir" >
    <fileset>
      <include name="dira"/>
      <include name="dirb/**"/>
    </fileset>
  </ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and tries to remove `/somedir/dira` directory and all the directories tree starting at, and including, `/somedir/dirb`. When removing the `/somedir/dirb` tree, it will start at the leaves moving up to the root, so that when it tries to remove a directory it is sure all the directories under it are already removed. Obviously all the files in the tree must have been already deleted.

As an example suppose you want to delete everything contained into `/somedir`, so invoke first the `<ftp>` task with `action="delete"`, then with `action="rmdir"` specifying in both cases `remotedir="/somedir"` and

```
    <fileset>
        <include name="**"/>
    </fileset>
```

The directory specified in the `remotedir` parameter is never selected for remove, so if you need to remove it, specify its parent in `remotedir` parameter and include it in the `<fileset>` pattern, like `"somedir/**"`.

RExec

Description

Task to automate a remote rexec session. Just like the Telnet task, it uses nested `<read>` to indicate strings to wait for, and `<write>` tags to specify text to send to the remote process.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information.

You can specify the commands you want to execute as nested elements or via the command attribute, we recommend you use the command attribute. If you use the command attribute, you must use the username and password attributes as well.

Parameters

Attribute	Values	Required
userid	the login id to use on the remote server.	No
password	the login password to use on the remote server.	No
server	the address of the remote rexec server.	Yes
command	the command to execute on the remote server.	No
port	the port number of the remote rexec server. Defaults to port 512 in BSD Unix systems.	No
timeout	set a default timeout to wait for a response. Specified in seconds. Default is no timeout.	No

Nested Elements

The input to send to the server, and responses to wait for, are described as nested elements.

read

declare (as a text child of this element) a string to wait for. The element supports the timeout attribute, which overrides any timeout specified for the task as a whole. It also has a `string` attribute, which is an alternative to specifying the string as a text element.

It is not necessary to declare a closing `<read>` element like for the Telnet task. The connection is not broken until the command has completed and the input stream (output of the command) is terminated.

write

describes the text to send to the server. The `echo` boolean attribute controls whether the string is echoed to the local log; this is "true" by default

Example

A simple example of connecting to a server and running a command.

```
<rexec userid="bob" password="badpass" server="localhost" command="ls"/>
```

The task can be used with other ports as well:

```
<rexec port="80" userid="bob" password="badpass" server="localhost" command="ls"/>
```

SCP

Description

since Ant 1.6

Copies a file or FileSet to or from a (remote) machine running an SSH daemon. FileSet *only* works for copying files from the local machine to a remote machine.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information. This task has been tested with jsch-0.1.2 and later.

See also the sshexec task

Parameters

Attribute	Description	Required
file	The file to copy. This can be a local path or a remote path of the form *user[:password]@host:/directory/path*. *:password* can be omitted if you use key based authentication or specify the password attribute. The way remote path is recognized is whether it contains @ character or not. This will not work if your localPath contains @ character.	Yes, unless a nested `<fileset>` element is used.
localFile	This is an alternative to the file attribute. But this must always point to a local file. The reason this was added was that when you give file attribute it is treated as remote if it contains @ character. This character can exist also in local paths. *since Ant 1.6.2*	Alternative to file attribute.
remoteFile	This is an alternative to the file attribute. But this must always point to a remote file. *since Ant 1.6.2*	Alternative to file attribute.
todir	The directory to copy to. This can be a local path or a remote path of the form *user[:password]@host:/directory/path*. *:password* can be omitted if you use key based authentication or specify the password attribute. The way remote path is recognized is whether it contains @ character or not. This will not work if your localPath contains @ character.	Yes
localTodir	This is an alternative to the todir attribute. But this must always point to a local directory. The reason this was added was that when you give todir attribute it is treated as remote if it contains @ character. This character can exist also in local paths. *since Ant 1.6.2*	Alternative to todir attribute.
localTofile	Changes the file name to the given name while receiving it, only useful if receiving a single file. *since Ant 1.6.2*	Alternative to todir attribute.
remoteTodir	This is an alternative to the todir attribute. But this must always point to a remote directory. *since Ant 1.6.2*	Alternative to todir attribute.
remoteTofile	Changes the file name to the given name while sending it, only useful if sending a single file. *since Ant 1.6.2*	Alternative to todir attribute.

port	The port to connect to on the remote host.	No, defaults to 22.
trust	This trusts all unknown hosts if set to yes/true. **Note** If you set this to false (the default), the host you connect to must be listed in your knownhosts file, this also implies that the file exists.	No, defaults to No.
knownhosts	This sets the known hosts file to use to validate the identity of the remote host. This must be a SSH2 format file. SSH1 format is not supported.	No, defaults to ${user.home}/.ssh/known_hosts.
failonerror	Whether to halt the build if the transfer fails.	No; defaults to true.
password	The password.	Not if you are using key based authentication or the password has been given in the file or todir attribute.
keyfile	Location of the file holding the private key.	Yes, if you are using key based authentication.
passphrase	Passphrase for your private key.	No, defaults to an empty string.
verbose	Determines whether SCP outputs verbosely to the user. Currently this means outputting dots/stars showing the progress of a file transfer. *since Ant 1.6.2*	No; defaults to false.
sftp	Determines whether SCP uses the sftp protocol. The sftp protocol is the file transfer protocol of SSH2. It is recommended that this be set to true if you are copying to/from a server that doesn't support scp1. *since Ant 1.7*	No; defaults to false.
preserveLastModified	Determines whether the last modification timestamp of downloaded files is preserved. It only works when transferring from a remote to a local system and probably doesn't work with a server that doesn't support SSH2. *since Ant 1.8.0*	No; defaults to false.

Parameters specified as nested elements

fileset

FileSets are used to select sets of files to copy. To use a fileset, the `todir` attribute must be set.

Examples

Copy a single local file to a remote machine

```
<scp file="myfile.txt" todir="user:password@somehost:/home/chuck"/>
```

Copy a single local file to a remote machine with separate password attribute

```
<scp file="myfile.txt" todir="user@somehost:/home/chuck" password="password"/>
```

Copy a single local file to a remote machine using key base authentication.

```
<scp file="myfile.txt"
     todir="user@somehost:/home/chuck"
     keyfile="${user.home}/.ssh/id_dsa"
     passphrase="my extremely secret passphrase"
/>
```

Copy a single remote file to a local directory

```
<scp file="user:password@somehost:/home/chuck/myfile.txt" todir="../some/other/dir"/>
```

Copy a remote directory to a local directory

```
<scp file="user:password@somehost:/home/chuck/*" todir="/home/sara"/>
```

Copy a local directory to a remote directory

```
<scp todir="user:password@somehost:/home/chuck/">
  <fileset dir="src_dir"/>
</scp>
```

Copy a set of files to a directory

```
<scp todir="user:password@somehost:/home/chuck">
  <fileset dir="src_dir">
    <include name="**/*.java"/>
  </fileset>
</scp>

<scp todir="user:password@somehost:/home/chuck">
  <fileset dir="src_dir" excludes="**/*.java"/>
</scp>
```

Security Note: Hard coding passwords and/or usernames in scp task can be a serious security hole. Consider using variable substitution and include the password on the command line. For example:

```
   <scp todir="${username}:${password}@host:/dir" ...>
```

Invoking ant with the following command line:

```
   ant -Dusername=me -Dpassword=mypassword target1 target2
```

Is slightly better, but the username/password is exposed to all users on an Unix system (via the ps command). The best approach is to use the `<input>` task and/or retrieve the password from a (secured) .properties file.

Unix Note: File permissions are not retained when files are copied; they end up with the default `UMASK` permissions instead. This is caused by the lack of any means to query or set file permissions in the current Java runtimes. If you need a permission- preserving copy function, use `<exec executable="scp" ... >` instead.

SSHEXEC

Description

since Ant 1.6

Runs a command on a remote machine running SSH daemon.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information. This task has been tested with jsch-0.1.29 and above and won't work with versions of jsch earlier than 0.1.28.

See also the scp task

Parameters

Attribute	Description	Required
host	The hostname or IP address of the remote host to which you wish to connect.	Yes
username	The username on the remote host to which you are connecting.	Yes
command	The command to run on the remote host.	Either this or commandResource must be set
commandResource	The resource (file) that contains the commands to run on the remote host. Since Ant 1.7.1	Either this or command must be set
port	The port to connect to on the remote host.	No, defaults to 22.
trust	This trusts all unknown hosts if set to yes/true. **Note** If you set this to false (the default), the host you connect to must be listed in your knownhosts file, this also implies that the file exists.	No, defaults to No.
knownhosts	This sets the known hosts file to use to validate the identity of the remote host. This must be a SSH2 format file. SSH1 format is not supported.	No, defaults to ${user.home}/.ssh/known_hosts.
failonerror	Whether to halt the build if the command does not complete successfully.	No; defaults to true.
password	The password.	Not if you are using key based authentication or the password has been given in the file or todir attribute.
keyfile	Location of the file holding the private key.	Yes, if you are using key based authentication.
passphrase	Passphrase for your private key.	No, defaults to an empty string.
output	Name of a file to which to write the output.	No
append	Whether output file should be appended to or overwritten. Defaults to false, meaning overwrite any existing file.	No
outputproperty	The name of a property in which the output of the command	No

	should be stored. If you use the commandResource attribute, each command's output will be prefixed by the command itself.	
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds **unlike telnet, which expects a timeout in seconds**). Defaults to 0 which means "wait forever".	No
input	A file from which the executed command's standard input is taken. This attribute is mutually exclusive with the inputstring attribute. When executing more than one command via commandResource, input will be read for each command. *since Ant 1.8.0*	No
verbose	Determines whether sshexec outputs verbosely to the user. Similar output is generated as the ssh commandline tool wit the -v option. *since Ant 1.8.0*	No, defaults to false
inputstring	A string which serves as the input stream for the executed command. This attribute is mutually exclusive with the input attribute. When executing more than one command via commandResource, input will be read for each command. *since Ant 1.8.0*	No

Examples

Run a command on a remote machine using password authentication

```
  <sshexec host="somehost"
        username="dude"
        password="yo"
        command="touch somefile"/>
```

Run a command on a remote machine using key authentication

```
  <sshexec host="somehost"
        username="dude"
        keyfile="${user.home}/.ssh/id_dsa"
        passphrase="yo its a secret"
        command="touch somefile"/>
```

Run a command on a remote machine using key authentication with no passphrase

```
  <sshexec host="somehost"
        username="dude"
        keyfile="${user.home}/.ssh/id_dsa"
        command="touch somefile"/>
```

Run a set of commands from a command resource (file) on a remote machine using key authentication with no passphrase

```
  <sshexec host="somehost"
        username="dude"
        keyfile="${user.home}/.ssh/id_dsa"
        commandResource="to_run"/>
```

Security Note: Hard coding passwords and/or usernames in sshexec task can be a serious security hole. Consider using variable substitution and include the password on the command line. For example:

```
  <sshexec host="somehost"
        username="${username}"
        password="${password}"
        command="touch somefile"/>
```

Invoking ant with the following command line:

```
    ant -Dusername=me -Dpassword=mypassword target1 target2
```

Is slightly better, but the username/password is exposed to all users on an Unix system (via the ps command). The best approach is to use the `<input>` task and/or retrieve the password from a (secured) .properties file.

Telnet

Description

Task to automate a remote telnet session. The task uses nested `<read>` to indicate strings to wait for, and `<write>` tags to specify text to send.

If you do specify a userid and password, the system will assume a common unix prompt to wait on. This behavior can be easily over-ridden.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information.

Parameters

Attribute	Values	Required
userid	the login id to use on the telnet server.	Only if password is specified
password	the login password to use on the telnet server.	Only if userid is specified
server	the address of the remote telnet server.	Yes
port	the port number of the remote telnet server. Defaults to port 23.	No
initialCR	send a cr after connecting ("yes"). Defaults to "no".	No
timeout	set a default timeout to wait for a response. Specified in seconds. Default is no timeout.	No

Nested Elements

The commands to send to the server, and responses to wait for, are described as nested elements.

read

declare (as a text child of this element) a string to wait for. The element supports the timeout attribute, which overrides any timeout specified for the task as a whole. It also has a `string` attribute, which is an alternative to specifying the string as a text element.

Always declare an opening and closing `<read>` element to ensure that statements are not sent before the connection is ready, and that the connection is not broken before the final command has completed.

write

describes the text to send to the server. The `echo` boolean attribute controls whether the string is echoed to the local log; this is "true" by default

Examples

A simple example of connecting to a server and running a command. This assumes a prompt of "ogin:" for the userid, and a prompt of "assword:" for the password.

```
<telnet userid="bob" password="badpass" server="localhost">
   <read>/home/bob</read>
   <write>ls</write>
   <read string="/home/bob"/>
</telnet>
```

This task can be rewritten as:

```
<telnet server="localhost">
   <read>ogin:</read>
   <write>bob</write>
   <read>assword:</read>
   <write>badpass</write>
   <read>/home/bob</read>
   <write>ls</write>
   <read>/home/bob</read>
</telnet>
```

A timeout can be specified at the `<telnet>` level or at the `<read>` level. This will connect, issue a sleep command that is suppressed from displaying and wait 10 seconds before quitting.

```
<telnet userid="bob" password="badpass" server="localhost" timeout="20">
   <read>/home/bob</read>
   <write echo="false">sleep 15</write>
   <read timeout="10">/home/bob</read>
</telnet>
```

The task can be used with other ports as well:

```
<telnet port="80" server="localhost" timeout="20">
   <read/>
   <write>GET / http/0.9</write>
   <write/>
   <read timeout="10">&lt;/HTML&gt;</read>
</telnet>
```

To use this task against the WinNT telnet service, you need to configure the service to use classic authentication rather than NTLM negotiated authentication. This can be done in the Telnet Server Admin app: select "display/change registry settings", then "NTLM", then set the value of NTLM to 1.

CvsChangeLog

Description

Generates an XML-formatted report file of the change logs recorded in a CVS repository.

Important: This task needs "cvs" on the path. If it isn't, you will get an error (such as error 2 on windows). If <cvs> doesn't work, try to execute cvs.exe from the command line in the target directory in which you are working. Also note that this task assumes that the cvs exceutable is compatible with the Unix version from cvshome.org, this is not completely true for certain other cvs clients - like CVSNT for example - and some operation may fail when using such an incompatible client.

Parameters

included in the report. *Since Ant 1.8.0*

Attribute	Description	Required
Attributes from parent Cvs task which are meaningful here Since ant 1.6.1		
cvsRoot	the CVSROOT variable.	No
cvsRsh	the CVS_RSH variable.	No
package	the package/module to check out. **Note:** multiple attributes can be split using spaces. Use a nested <module> element if you want to specify a module with spaces in its name.	No
port	Port used by CVS to communicate with the server.	No, default port 2401.
passfile	Password file to read passwords from.	No, default file ~/.cvspass.
failonerror	Stop the build process if the command exits with a return code other than 0. Defaults to false	No
tag	query the changelog for a specific branch.	No
Specific attributes		
dir	The directory from which to run the CVS *log* command.	No; defaults to ${basedir}.
destfile	The file in which to write the change log report.	Yes
usersfile	Property file that contains name-value pairs mapping user IDs and names that should be used in the report in place of the user ID.	No
daysinpast	Sets the number of days into the past for which the change log information should be retrieved.	No
start	The earliest date from which change logs are to be included in the report.	No
end	The latest date to which change logs are to be included in the report.	No
remote	If set to true, works against the repository (using rlog) without a working copy. Default is false. *Since Ant 1.8.0*	No
startTag	The start of a tag range. If endTag is also specified, they must both be on the same branch. If endTag is not specified, the end of the range will be the latest on the same branch on	No

	which startTag lives. *Since Ant 1.8.0*	
endTag	The end of a tag range. If startTag is also specified, they must both be on the same branch. If startTag is not specified, the start of the range will be the top of the branch on which endTag lives.	No

Parameters specified as nested elements

user

The nested `<user>` element allows you to specify a mapping between a user ID as it appears on the CVS server and a name to include in the formatted report. Anytime the specified user ID has made a change in the repository, the `<author>` tag in the report file will include the name specified in `displayname` rather than the user ID.

Attribute	Description	Required
displayname	The name to be used in the CVS change log report.	Yes
userid	The userid of the person as it exists on the CVS server.	Yes

module

Specifies a package/module to work on, unlike the package attribute modules specified using this attribute can contain spaces in their name.

Attribute	Description	Required
name	The module's/package's name.	Yes.

Examples

```
<cvschangelog dir="dve/network"
              destfile="changelog.xml"
/>
```

Generates a change log report for all the changes that have been made under the `dve/network` directory. It writes these changes into the file `changelog.xml`.

```
<cvschangelog dir="dve/network"
              destfile="changelog.xml"
              daysinpast="10"
/>
```

Generates a change log report for any changes that were made under the `dve/network` directory in the past 10 days. It writes these changes into the file `changelog.xml`.

```
<cvschangelog dir="dve/network"
              destfile="changelog.xml"
              start="20 Feb 2002"
              end="20 Mar 2002"
/>
```

Generates a change log report for any changes that were made between February 20, 2002 and March 20, 2002 under the `dve/network` directory. It writes these changes into the file `changelog.xml`.

```
<cvschangelog dir="dve/network"
              destfile="changelog.xml"
              start="20 Feb 2002"
/>
```

Generates a change log report for any changes that were made after February 20, 2002 under the `dve/network` directory. It writes these changes into the file `changelog.xml`.

```
  <cvschangelog dir="dve/network"
                destfile="changelog.xml">
       <user displayname="Peter Donald" userid="donaldp"/>
  </cvschangelog>
```

Generates a change log report for all the changes that were made under the `dve/network` directory, substituting the name "Peter Donald" in the `<author>` tags anytime it encounters a change made by the user ID "donaldp". It writes these changes into the file `changelog.xml`.

Generates a change log report on the `ANT_16_BRANCH`.

```
 <cvschangelog dir="c:/dev/asf/ant.head" passfile="c:/home/myself/.cvspass"
                destfile="changelogant.xml" tag="ANT_16_BRANCH"/>
```

Generate Report

Ant includes a basic XSLT stylesheet that you can use to generate a HTML report based on the xml output. The following example illustrates how to generate a HTML report from the XML report.

```
        <style in="changelog.xml"
               out="changelog.html"
               style="${ant.home}/etc/changelog.xsl">
          <param name="title" expression="Ant ChangeLog"/>
          <param name="module" expression="ant"/>
          <param name="cvsweb" expression="http://cvs.apache.org/viewcvs/"/>
        </style>
```

Sample Output

```
<changelog>
  <entry>
    <date>2002-03-06</date>
    <time>12:00</time>
    <author>Peter Donald</author>
    <file>
      <name>org/apache/myrmidon/build/AntlibDescriptorTask.java</name>
      <revision>1.3</revision>
      <prevrevision>1.2</prevrevision>
    </file>
    <msg><![CDATA[Use URLs directly rather than go via a File.

This allows templates to be stored inside jar]]></msg>
  </entry>
</changelog>
```

cvspass

Description

Adds entries to a .cvspass file. Adding entries to this file has the same affect as a cvs login command.

CVSNT Note: CVSNT prefers users to store the passwords inside the registry. If the task doesn't seem to work for you, the most likely reason is that CVSNT ignores your .cvspass file completely. See bug zilla report 21657 for recommended workarounds.

Parameters

Attribute	Description	Required
cvsroot	the CVS repository to add an entry for.	Yes
password	Password to be added to the password file.	Yes
passfile	Password file to add the entry to.	No, default is `~/.cvspass`.

Examples

```
<cvspass cvsroot=":pserver:anoncvs@cvs.apache.org:/home/cvspublic"
     password="anoncvs"
/>
```

Adds an entry into the ~/.cvspass password file.

CvsTagDiff

Description

Generates an XML-formatted report file of the changes between two tags or dates recorded in a CVS repository.

Important: This task needs "cvs" on the path. If it isn't, you will get an error (such as error 2 on windows). If <cvs> doesn't work, try to execute cvs.exe from the command line in the target directory in which you are working. Also note that this task assumes that the cvs exceutable is compatible with the Unix version from cvshome.org, this is not completely true for certain other cvs clients - like CVSNT for example - and some operation may fail when using such an incompatible client.

Parameters

Attribute	Description	Required
startTag	The earliest tag from which diffs are to be included in the report.	exactly one of the two.
startDate	The earliest date from which diffs are to be included in the report. accepts all formats accepted by the cvs command for -D date_spec arguments	
endTag	The latest tag from which diffs are to be included in the report.	exactly one of the two.
endDate	The latest date from which diffs are to be included in the report. accepts all formats accepted by the cvs command for -D date_spec arguments	
destfile	The file in which to write the diff report.	Yes
ignoreRemoved	When set to true, the report will not include any removed files. *Since Ant 1.8.0*	No, defaults to false.

Parameters inherited from the cvs task

Attribute	Description	Required
compression	true, false, or the number 1-9 (corresponding to possible values for CVS -z# argument). Any other value is treated as false	No. Defaults to no compression. if passed true, level 3 compression is assumed.
cvsRoot	the CVSROOT variable.	No
cvsRsh	the CVS_RSH variable.	No
package	the package/module to analyze. Since ant 1.6 multiple packages separated by spaces are possible. aliases corresponding to different modules are also possible Use a nested <module> element if you want to specify a module with spaces in its name.	No
Yes		
quiet	suppress informational messages.	No, default "false"
port	Port used by CVS to communicate with the server.	No, default port 2401.
passfile	Password file to read passwords from.	No, default file ~/.cvspass.

failonerror	Stop the buildprocess if the command exits with a returncode other than 0. Defaults to false	No

Parameters specified as nested elements

module

Specifies a package/module to work on, unlike the package attribute modules specified using this attribute can contain spaces in their name.

Attribute	Description	Required
name	The module's/package's name.	Yes.

Examples

```
<cvstagdiff cvsRoot=":pserver:anoncvs@cvs.apache.org:/home/cvspublic"
              destfile="tagdiff.xml"
              package="ant"
              startTag="ANT_14"
              endTag="ANT_141"
/>
```

Generates a tagdiff report for all the changes that have been made in the `ant` module between the tags `ANT_14` and `ANT_141`. It writes these changes into the file `tagdiff.xml`.

```
<cvstagdiff
              destfile="tagdiff.xml"
              package="ant"
              startDate="2002-01-01"
              endDate="2002-31-01"
/>
```

Generates a tagdiff report for all the changes that have been made in the `ant` module in january 2002. In this example `cvsRoot` has not been set. The current `cvsRoot` will be used (assuming the build is started from a folder stored in `cvs`. It writes these changes into the file `tagdiff.xml`.

```
<cvstagdiff
              destfile="tagdiff.xml"
              package="ant jakarta-gump"
              startDate="2003-01-01"
              endDate="2003-31-01"
/>
```

Generates a tagdiff report for all the changes that have been made in the `ant` and `jakarta-gump` modules in january 2003. In this example `cvsRoot` has not been set. The current `cvsRoot` will be used (assuming the build is started from a folder stored in `cvs`. It writes these changes into the file `tagdiff.xml`.

Generate Report

Ant includes a basic XSLT stylesheet that you can use to generate a HTML report based on the xml output. The following example illustrates how to generate a HTML report from the XML report.

```
      <style in="tagdiff.xml"
             out="tagdiff.html"
             style="${ant.home}/etc/tagdiff.xsl">
        <param name="title" expression="Ant Diff"/>
        <param name="module" expression="ant"/>
        <param name="cvsweb" expression="http://cvs.apache.org/viewcvs/"/>
      </style>
```

Output

The cvsroot and package attributes of the tagdiff element are new in ant 1.6.
Notes on entry attributes :

Attribute	Comment
name	when reporting on one package, the package name is removed from the output
revision	supplied for files which exist at the end of the reporting period
prevrevision	supplied for files which exist at the beginning of the reporting period. Old CVS servers do not supply it for deleted files. CVS 1.12.2 supplies it.

```
<?xml version="1.0" encoding="UTF-8"?>
<tagdiff startTag="ANT_14" endTag="ANT_141"
cvsroot=":pserver:anoncvs@cvs.apache.org:/home/cvspublic" package="ant">
  <entry>
    <file>
      <name>src/main/org/apache/tools/ant/DirectoryScanner.java</name>
      <revision>1.15.2.1</revision>
      <prevrevision>1.15</prevrevision>
    </file>
  </entry>
</tagdiff>
```

Ant ClearCase Tasks

by:
Curtis White (cwhite at aracnet dot com),
Sean P. Kane (spkane at genomatica dot com),
Rob Anderson (Anderson.Rob at vectorscm dot com), and
Sean Egan (sean at cm-logic dot com)

Version 1.6 - 02/25/2003

ClearCase Support

Table of Contents

Introduction

Ant provides several optional tasks for working with ClearCase. These tasks correspond to various ClearCase commands using the Cleartool program. The current tasks available for Ant correspond to only a few of the significant ClearCase commands.

More tasks can be easily added by deriving from the ClearCase class and then adding functionality that is specific to that ClearCase command.

Important: these tasks all require `cleartool` on the command line. If a task fails with an IOException, especially error code 2 on Windows, this is your problem.

CCCheckin

Description

Task to perform a "cleartool checkin" command to ClearCase.

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
comment	Specify a comment. Only one of comment or commentfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or commentfile may be used.	No
nowarn	Suppress warning messages	No
preservetime	Preserve the modification time	No
keepcopy	Keeps a copy of the file with a .keep extension	No
identical	Allows the file to be checked in even if it is identical to the original	No
failonerr	Throw an exception if the command fails. Default is true	No

Examples

```
<cccheckin viewpath="c:/views/viewdir/afile"
        commentfile="acomment.txt"
        nowarn="true"
        identical="true"/>
```

Does a ClearCase *checkin* on the file *c:/views/viewdir/afile*. Comment text from the file *acomment.txt* is added to ClearCase as a comment. All warning messages are suppressed. The file is checked in even if it is *identical* to the original.

CCCheckout

Description

Task to perform a "cleartool checkout" command to ClearCase.

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
reserved	Specifies whether to check out the file as reserved or not	Yes
out	Creates a writable file under a different filename	No
nodata	Checks out the file but does not create an editable file containing its data	No
branch	Specify a branch to check out the file to	No
version	Allows checkout of a version other than main latest	No
nowarn	Suppress warning messages	No

comment	Specify a comment. Only one of comment or commentfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or commentfile may be used.	No
notco	Fail if it's already checked out to the current view. Set to false to ignore it. Since ant 1.6.1	No
failonerr	Throw an exception if the command fails. Default is true. Since ant 1.6.1	No

Examples

```
<cccheckout viewpath="c:/views/viewdir/afile"
        reserved="true"
        branch="abranch"
        nowarn="true"
        comment="Some comment text"/>
```

Does a ClearCase *checkout* on the file *c:/views/viewdir/afile*. It is checked out as *reserved* on branch called *abranch*. All warning messages are suppressed. A *Some comment text* is added to ClearCase as a comment.

CCUnCheckout

Description

Task to perform a UnCheckout command to ClearCase.

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
keepcopy	Specifies whether to keep a copy of the file with a .keep extension or not	No
failonerr	Throw an exception if the command fails. Default is true Since ant 1.6.1	No

Examples

```
<ccuncheckout viewpath="c:/views/viewdir/afile"
        keepcopy="true"/>
```

Does a ClearCase *uncheckout* on the file *c:/views/viewdir/afile*. A copy of the file called *c:/views/viewdir/afile.keep* is kept.

CCUpdate

Description

Task to perform an "cleartool update" command to ClearCase.

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase snapshot view file or directory that the command will operate on	No
graphical	Displays a graphical dialog during the update	No
log	Specifies a log file for ClearCase to write to	No
overwrite	Specifies whether to overwrite hijacked files or not	No
rename	Specifies that hijacked files should be renamed with a .keep extension	No
currenttime	Specifies that modification time should be written as the current time. Either currenttime or preservetime can be specified.	No
preservetime	Specifies that modification time should preserved from the VOB time. Either currenttime or preservetime can be specified.	No
failonerr	Throw an exception if the command fails. Default is true. Since ant 1.6.1	No

Examples

```
<ccupdate viewpath="c:/views/viewdir"
        graphical="false"
        log="log.log"
        overwrite="true"
        currenttime="true"
        rename="false"/>
```

Does a ClearCase *update* on the snapshot view directory *c:/views/viewdir*. A graphical dialog will be displayed. The output will be logged to *log.log* and it will overwrite any hijacked files. The modified time will be set to the current time.

CCMklbtype

Description

Task to perform a "mklbtype" command to ClearCase.

Parameters

Attribute	Values	Required
typename	Name of the label type to create	Yes
vob	Name of the VOB	No
replace	Replace an existing label definition of the same type	No
global	Either global or ordinary can be specified, not both. Creates a label type that is global to the VOB or to VOBs that use this VOB	No
ordinary	Either global or ordinary can be specified, not both. Creates a label type that can be used only in the current VOB. **Default**	No

pbranch	Allows the label type to be used once per branch in a given element's version tree	No
shared	Sets the way mastership is checked by ClearCase. See ClearCase documentation for details	No
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
failonerr	Throw an exception if the command fails. Default is true Since ant 1.6.1	No

Examples

```
<ccmklbtype typename="VERSION_1"
        ordinary="true"
        comment="Development version 1"/>
```

Does a ClearCase *mklbtype* to create a label type named *VERSION_1*. It is created as *ordinary* so it is available only to the current VOB. The text *Development version 1* is added as a comment.

CCMklabel

Description

Task to perform a "mklabel" command to ClearCase.

Parameters

Attribute	Values	Required
typename	Name of the label type	Yes
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
replace	Replace a label of the same type on the same branch	No
recurse	Process each subdirectory under viewpath	No
version	Identify a specific version to attach the label to	No
vob	Name of the VOB	No
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
failonerr	Throw an exception if the command fails. Default is true Since ant 1.6.1	No

Examples

```
<ccmklabel viewpath="c:/views/viewdir/afile"
        comment="Some comment text"
        recurse="true"
        version="\main\2"
        typename="VERSION_1"/>
```

Does a ClearCase *mklabel* on the file *c:/views/viewdir/afile* under the main branch for version 2 (*\main\2*). Text *Some comment text* is added as a comment. It will *recurse* all subdirectories.

CCRmtype

Description

Task to perform a "rmtype" command to ClearCase.

Parameters

Attribute	Values	Required
typekind	The kind of type to create. Valid types are: **attype** - attribute type **brtype** - branch type **eltype** - element type **hltype** - hyperlink type **lbtype** - label type **trtype** - trigger type	Yes
typename	The name of the type to remove	Yes
ignore	Used with trigger types only. Forces removal of trigger type even if a pre-operation trigger would prevent its removal	No
rmall	Removes all instances of a type and the type object itself	No
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
failonerr	Throw an exception if the command fails. Default is true Since ant 1.6.1	No

Examples

```
<ccrmtype typekind="lbtype"
        typename="VERSION_1"
        commentfile="acomment.txt"
        rmall="true"/>
```

Does a ClearCase *rmtype* to remove a label type (*lbtype*) named *VERSION_1*. Comment text from the file *acomment.txt* is added as a comment. All instances of the type are removed, including the type object itself.

CCLock

Description

Task to perform a "cleartool lock" command to ClearCase.

Parameters

Attribute	Values	Required

replace	Specifies replacing an existing lock	No
nusers	Specifies user(s) who can still modify the object	No
obsolete	Specifies that the object should be marked obsolete	No
comment	Specifies how to populate comments fields	No
pname	Specifies the object pathname to be locked.	No
objselect	This variable is obsolete. Should use *objsel* instead.	No
objsel	Specifies the object(s) to be locked. Since ant 1.6.1	No
failonerr	Throw an exception if the command fails. Default is true. Since ant 1.6.1	No

Examples

```
<cclock
    objsel="stream:Application_Integration@\MyProject_PVOB"
    />
```

Does a ClearCase *lock* on the object *stream:Application_Integration@\MyProject_PVOB*.

CCUnlock

Description

Task to perform a "cleartool unlock" command to ClearCase.

Parameters

Attribute	Values	Required
comment	Specifies how to populate comments fields	No
pname	Specifies the object pathname to be unlocked.	No
objselect	This variable is obsolete. Should use *objsel* instead.	No
objsel	Specifies the object(s) to be unlocked. Since ant 1.6.1	No
failonerr	Throw an exception if the command fails. Default is true. Since ant 1.6.1	No

Examples

```
<ccunlock
    objsel="stream:Application_Integration@\MyProject_PVOB"
    />
```

Does a ClearCase *unlock* on the object *stream:Application_Integration@\MyProject_PVOB*.

CCMkbl

Description

Task to perform a "cleartool mkbl" command to ClearCase.

Parameters

Attribute	Values	Required
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
baselinerootname	Specify the name to be associated with the baseline.	Yes
nowarn	Suppress warning messages	No
identical	Allows the baseline to be created even if it is identical to the previous baseline.	No
full	Creates a full baseline.	No
nlabel	Allows the baseline to be created without a label.	No
failonerr	Throw an exception if the command fails. Default is true. Since ant 1.6.1	No

Examples

```
<ccmkbl
    baselinerootname="Application_Baseline_AUTO"
    identical="yes"
    full="no"
    viewpath="v:\ApplicationCC"
    />
```

Does a ClearCase *mkbl* on the Integration view at *v:\ApplicationCC* even if it is *identical* to a previous baseline. The new baseline with be incremental and named "Application_Baseline_AUTO".

CCMkattr

Description

Task to perform a "cleartool mkattr" command to ClearCase.
Since ant 1.6.1

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	Yes
replace	Replace the value of the attribute if it already exists	No
recurse	Process each subdirectory under viewpath	No

version	Identify a specific version to attach the attribute to	No
typename	Name of the attribute type	Yes
typevalue	Value to attach to the attribute type	Yes
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
failonerr	Throw an exception if the command fails. Default is true	No

Examples

```
<ccmkattr viewpath="c:/views/viewdir/afile"
    typename="BugFix"
    typevalue="34445"
    />
```

Does a ClearCase *mkattr* on the file *c:/views/viewdir/afile* and attaches the attribute *BugFix* with a value of *34445* to it.

CCMkdir

Description

Task to perform a "cleartool mkdir" command to ClearCase.
Since ant 1.6.1

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view directory that the command will operate on	Yes
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
nocheckout	Do not checkout after element creation	No
failonerr	Throw an exception if the command fails. Default is true	No

Examples

```
<ccmkdir viewpath="c:/views/viewdir/adir"
        nochcekout="true"
        comment="Some comment text"/>
```

Does a ClearCase *mkdir* on the dir *c:/views/viewdir/adir* and does not automatically check it out.

CCMkelem

Description

Task to perform a "cleartool mkelem" command to ClearCase.
Since ant 1.6.1

Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	Yes
comment	Specify a comment. Only one of comment or cfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or cfile may be used.	No
nowarn	Suppress warning messages	No
nocheckout	Do not checkout after element creation	No
checkin	Checkin element after creation	No
preservetime	Preserve the modification time (for checkin)	No
master	Assign mastership of the main branch to the current site	No
eltype	Element type to use during element creation	No
failonerr	Throw an exception if the command fails. Default is true	No

Examples

```
<ccmkelem viewpath="c:/views/viewdir/afile"
        eltype="text_file"
        checkin="true"
        comment="Some comment text"/>
```

Does a ClearCase *mkelem* on the file *c:/views/viewdir/afile* with element type *text_file*. It also checks in the file after creation.

Continuus Support

These ant tasks are wrappers around Continuus Source Manager. They have been tested against versions 5.1/6.2 on Windows 2000, but should work on other platforms with ccm installed.

author: Benoit Mousaud (benoit.moussaud@criltelecom.com)

CCMCheckin

Description

Task to checkin a file

Parameters

Attribute	Values	Required
file	Path to the file that the command will operate on	Yes
comment	Specify a comment. Default is "Checkin" plus the date	No
task	Specify the task number used to check in the file (may use 'default')	No
ccmdir	path to the ccm executable file, required if it is not on the PATH	No

Examples

```
<ccmcheckin file="c:/wa/com/foo/MyFile.java"
        comment="mycomment"/>
```

Checks in the file *c:/wa/com/foo/MyFile.java*. Comment attribute *mycomment* is added as a task comment. The task used is the one set as the default.

CCMCheckout

Description

Task to perform a Checkout command to Continuus

Parameters

Attribute	Values	Required
file	Path to the file that the command will operate on	Yes (file|fileset)
fileset	fileset containing the file to be checked out	
comment	Specify a comment.	No
task	Specify the task number used to checkin the file (may use 'default')	No
ccmdir	path to the ccm executable file, required if it is not on the PATH	No

Examples

```
<ccmcheckout file="c:/wa/com/foo/MyFile.java"
        comment="mycomment"/>
```

Check out the file *c:/wa/com/foo/MyFile.java*. Comment attribute *mycomment* is added as a task comment The used task is the one set as the default.

```
<ccmcheckout  comment="mycomment">
  <fileset dir="lib" >
    <include name="**/*.jar"/>
  </fileset>
</ccmcheckout >
```

Check out all the files in the *lib* directory having the *.jar* extension. Comment attribute *mycomment* is added as a task comment The used task is the one set as the default.

CCMCheckinTask

Description

Task to perform a check in default task command to Continuus

Parameters

Attribute	Values	Required
comment	Specify a comment.	No
task	Specify the task number used to check in the file (may use 'default')	No
ccmdir	path to the ccm executable file, required if it is not on the PATH	No

Examples

```
<ccmcheckintask comment="blahblah/>
```

Does a Checkin default task on all the checked out files in the current task.

CCMReconfigure

Description

Task to perform an reconfigure command to Continuus.

Parameters

Attribute	Values	Required
recurse	recurse on subproject (default false)	No
verbose	do a verbose reconfigure operation (default false)	No
ccmproject	Specifies the ccm project on which the operation is applied.	Yes
ccmdir	path to the ccm executable file, required if it is not on the PATH	No

Examples

```
<ccmreconfigure ccmproject="ANTCCM_TEST#BMO_1"
          verbose="true"/>
```

Does a Continuus *reconfigure* on the project *ANTCCM_TEST#BMO_1*.

CCMCreateTask

Description

Create a Continuus task.

Parameters

Attribute	Values	Required
comment	Specify a comment.	No
platform	Specify the target platform	No
ccmdir	path to the ccm executable file, required if it is not on the PATH	No
resolver	Specify the resolver	No
release	Specify the CCM release	No
subsystem	Specify the subsystem	No
task	Specify the task number used to checkin the file (may use 'default')	No

Examples

```
<ccmcreatetask resolver="${user.name}"
             release="ANTCCM_TEST" comment="blahblah"/>
```

Creates a task for the release *ANTCCM_TEST* with the current user as the resolver for this task.

Microsoft Visual SourceSafe Tasks User Manual

by

- Craig Cottingham
- Andrew Everitt
- Balazs Fejes 2
- Glenn_Twiggs@bmc.com
- Martin Poeschl (mpoeschl@marmot.at)
- Phillip Wells
- Jon Skeet (jon.skeet@peramon.com)
- Nigel Magnay (nigel.magnay@parsec.co.uk)
- Gary S. Weaver
- Jesse Stockall

Contents

Introduction

These tasks provide an interface to the Microsoft Visual SourceSafe SCM. The `org.apache.tools.ant.taskdefs.optional.vss` package consists of a simple framework to support vss functionality as well as some Ant tasks encapsulating frequently used vss commands. Although it is possible to use these commands on the desktop, they were primarily intended to be used by automated build systems.

If you get a CreateProcesss IOError=2 when running these, it means that ss.exe was not found. Check to see if you can run it from the command line -you may need to alter your path, or set the `ssdir` property.

The Tasks

vssget	Retrieves a copy of the specified VSS file(s).
vsslabel	Assigns a label to the specified version or current version of a file or project.
vsshistory	Shows the history of a file or project in VSS.
vsscheckin	Updates VSS with changes made to a checked out file, and unlocks the VSS master copy.
vsscheckout	Copies a file from the current project to the current folder, for the purpose of editing.
vssadd	Adds a new file into the VSS Archive
vsscp	Change the current project being used in VSS
vsscreate	Creates a project in VSS.

Task Descriptions

VssGet

Description

Task to perform GET commands to Microsoft Visual SourceSafe.

If you specify two or more attributes from version, date and label only one will be used in the order version, date, label.

Parameters

Attribute	Values	Required
vsspath	SourceSafe path which specifies the project/file(s) you wish to perform the action on.	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the password out and VSS does not accept login without a password.	No
localpath	Override the working directory and get to the specified path	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
writable	true or false; default false	No
recursive	true or false; default false. Note however that in the SourceSafe UI , there is a setting accessed via Tools/Options/GeneralTab called "Act on projects recursively". If this setting is checked, then the recursive attribute is effectively ignored, and the get will always be done recursively	No
version	a version number to get	No, only one of these allowed
date	a date stamp to get at	
label	a label to get for	
quiet	suppress output (off by default)	No
autoresponse	What to respond with (sets the -I option). By default, -I- is used; values of Y or N will be appended to this.	No
writablefiles	Behavior when local files are writable. Valid options are: `replace`, `skip` and `fail`; Defaults to `fail` `skip` implies `failonerror=false`	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No
filetimestamp	Set the behavior for timestamps of local files. Valid options are `current`, `modified`, or `updated`. Defaults to `current`.	No

Note that only one of version, date or label should be specified

Examples

```
<vssget localPath="C:\mysrc\myproject"
        recursive="true"
        label="Release1"
        login="me,mypassword"
        vsspath="$/source/aProject"
        writable="true"/>
```

Does a get on the VSS-Project *$/source/myproject* using the username *me* and the password *mypassword*. It will recursively get the files which are labeled *Release1* and write them to the local directory *C:\mysrc\myproject*. The local files will be writable.

VssLabel

Description

Task to perform LABEL commands to Microsoft Visual SourceSafe.

Assigns a label to the specified version or current version of a file or project.

Parameters

Attribute	Values	Required
vsspath	SourceSafe path which specifies the project/file(s) you wish to perform the action on.	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the password out and VSS does not accept login without a password.	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
label	A label to apply to the hierarchy	Yes
version	An existing file or project version to label. By default the current version is labeled.	No
comment	The comment to use for this label. Empty or '-' for no comment.	No
autoresponse	What to respond with (sets the -I option). By default, -I- is used; values of Y or N will be appended to this.	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No

Examples

```
<vsslabel vsspath="$/source/aProject"
          login="me,mypassword"
          label="Release1"/>
```

Labels the current version of the VSS project *$/source/aProject* with the label *Release1* using the username *me* and the password *mypassword*.

```
<vsslabel vsspath="$/source/aProject/myfile.txt"
          version="4"
          label="1.03.004"/>
```

Labels version 4 of the VSS file *$/source/aProject/myfile.txt* with the label *1.03.004*. If this version already has a label, the operation (and the build) will fail.

VssHistory

Description

Task to perform HISTORY commands to Microsoft Visual SourceSafe.

Parameters

Attribute	Values	Required
vsspath	SourceSafe path which specifies the project/file(s) you wish to perform the action on.	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the password out and VSS does not accept login without a password.	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
fromDate	Start date for comparison	See below
toDate	End date for comparison	See below
dateFormat	Format of dates in fromDate and toDate. Used when calculating dates with the numdays attribute. This string uses the formatting rules of SimpleDateFormat. Defaults to DateFormat.SHORT.	No
fromLabel	Start label for comparison	No
toLabel	Start label for comparison	No
numdays	The number of days for comparison.	See below
output	File to write the diff.	No
recursive	true or false	No
style	brief, codediff, default or nofile. The default is default.	No
user	Name the user whose changes we would like to see	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No

Specifying the time-frame

There are different ways to specify what time-frame you wish to evaluate:

- Changes between two dates: Specify both `fromDate` and `toDate`
- Changes before a date: Specify `toDate`
- Changes after a date: Specify `fromDate`
- Changes X Days before a date: Specify `toDate` and (negative!) `numDays`
- Changes X Days after a date: Specify `fromDate` and `numDays`

Examples

```
<vsshistory vsspath="$/myProject" recursive="true"
            fromLabel="Release1"
            toLabel="Release2"/>
```

Shows all changes between "Release1" and "Release2".

```
<vsshistory vsspath="$/myProject" recursive="true"
            fromDate="01.01.2001"
            toDate="31.03.2001"/>
```

Shows all changes between January 1st 2001 and March 31st 2001 (in Germany, date must be specified according to your locale).

```
<tstamp>
  <format property="to.tstamp" pattern="M-d-yy;h:mma"/>
</tstamp>

<vsshistory vsspath="$/myProject" recursive="true"
            numDays="-14"
            dateFormat="M-d-yy;h:mma"
            toDate="${to.tstamp}"/>
```

Shows all changes in the 14 days before today.

VssCheckin

Description

Task to perform CHECKIN commands to Microsoft Visual SourceSafe.

Parameters

Attribute	Values	Required
vsspath	SourceSafe path which specifies the project/file(s) you wish to perform the action on.	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the password out and VSS does not accept login without a password.	No
localpath	Override the working directory and get to the specified path	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
writable	true or false	No
recursive	true or false	No
comment	Comment to use for the files that where checked in.	No
autoresponse	'Y', 'N' or empty. Specify how to reply to questions from VSS.	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No

Examples

```
<vsscheckin vsspath="$/test/test*"
            localpath="D:\build\"
```

```
            comment="Modified by automatic build"/>
```

Checks in the file(s) named *test** in the project *$/test* using the local directory *D:\build*.

VssCheckout

Description

Task to perform CHECKOUT commands to Microsoft Visual SourceSafe.

If you specify two or more attributes from version, date and label only one will be used in the order version, date, label.

Parameters

Attribute	Values	Required
vsspath	SourceSafe path which specifies the project/file(s) you wish to perform the action on.	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the password out and VSS does not accept login without a password.	No
localpath	Override the working directory and get to the specified path	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
writable	true or false	No
recursive	true or false	No
version	a version number to get	No, only one of these allowed
date	a date stamp to get at	
label	a label to get for	
writablefiles	Behavior when local files are writable. Valid options are: `replace`, `skip` and `fail`; Defaults to `fail` `skip` implies `failonerror=false`	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No
filetimestamp	Set the behavior for timestamps of local files. Valid options are `current`, `modified`, or `updated`. Defaults to `current`.	No
getlocalcopy	Set the behavior to retrieve local copies of the files. Defaults to true.	No

Examples

```
<vsscheckout vsspath="$/test"
             localpath="D:\build"
             recursive="true"
             login="me,mypass"/>
```

Does a recursive checkout of the project *$/test* to the directory D:\build.

VssAdd

Description

Task to perform ADD commands to Microsoft Visual SourceSafe.

Parameters

Attribute	Values	Required
localpath	Specify the local file(s) to add to VSS	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the password out and VSS does not accept login without a password.	No
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
writable	true or false	No
recursive	true or false	No
comment	Comment to use for the files that where checked in.	No
autoresponse	'Y', 'N' or empty. Specify how to reply to questions from VSS.	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No

Examples

```
<vssadd localpath="D:\build\build.00012.zip"
            comment="Added by automatic build"/>
```

Add the file named build.00012.zip into the project current working directory (see vsscp).

VssCp

Description

Task to perform CP (Change Project) commands to Microsoft Visual SourceSafe.

This task is typically used before a VssAdd in order to set the target project

Parameters

Attribute	Values	Required
vsspath	SourceSafe path which specifies the project you wish to make the current project.	Yes
login	username[,password] - The username and password needed to get access to VSS. Note that you may need to specify both (if you have a password) - Ant/VSS will hang if you leave the	No

	password out and VSS does not accept login without a password.	
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
serverPath	directory where `srcsafe.ini` resides.	No
failonerror	Stop the buildprocess if ss.exe exits with a returncode of 100. Defaults to true	No

Examples

```
<vsscp vsspath="$/Projects/ant"/>
```

Sets the current VSS project to *$/Projects/ant*.

VssCreate

Description

Task to perform CREATE commands to Microsoft Visual Source Safe.

Creates a new project in VSS.

Parameters

Attribute	Values	Required
login	username,password	No
vsspath	SourceSafe path of project to be created	Yes
ssdir	directory where `ss.exe` resides. By default the task expects it to be in the PATH.	No
quiet	suppress output (off by default)	No
failOnError	fail if there is an error creating the project (true by default)	No
autoresponse	What to respond with (sets the -I option). By default, -I- is used; values of Y or N will be appended to this.	No
comment	The comment to use for this label. Empty or '-' for no comment.	No

Examples

```
<vscreate vsspath="$/existingProject/newProject"/>
```

Creates the VSS-Project *$/existingProject/newProject*.

Ant Pvcs Task User Manual

Note: Before using this task, the user running ant must have access to the commands of PVCS (get and pcli) and must have access to the repository. Note that the way to specify the repository is platform dependent so use property to specify location of repository.

by

- Thomas Christensen (tchristensen@nordija.com)
- Don Jeffery (donj@apogeenet.com)
- Jon Dickinson (dickinson.j@ucles.org.uk)

Version 1.1 - 2001/06/27

Problems with UNC pathnames and the use of () in paths are fixed and an updateonly argument introduced.

Version 1.0 - 2001/01/31

Initial release.

Table of Contents

Introduction

The pvcs task allows the user of ant to extract the latest edition of the source code from a PVCS repository. PVCS is a version control system developed by Merant.
This version has been tested against PVCS version 6.5 and 6.6 under Windows and Solaris.

Pvcs Task

Description

The pvcs task is set to point at a PVCS repository and optionally a project within that repository, and can from that specification get the latest version of the files contained by the repository.

Parameters

Attribute	Description	Required
repository	The location of the repository (see your PVCS manuals)	Yes

pvcsproject	The project within the PVCS repository to extract files from ("/" is root project and that is default if this attribute isn't specified)	No
label	Only files marked with this label are extracted.	No
promotiongroup	Only files within this promotion group are extracted. Using both the *label* and the *promotiongroup* tag will cause the files in the promotion group and with that label to be extracted.	No
config	path of a non default .cfg file. Can be given absolute or relative to ant's base directory.	No
force	If set to *yes* all files that exists and are writable are overwritten. Default *no* causes the files that are writable to be ignored. This stops the PVCS command *get* to stop asking questions!	No
workspace	By specifying a workspace, the files are extracted to that location. A PVCS workspace is a name for a location of the workfiles and isn't as such the location itself. You define the location for a workspace using the PVCS GUI clients. If this isn't specified the default workspace for the current user is used.	No
pvcsbin	On some systems the PVCS executables *pcli* and *get* are not found in the PATH. In such cases this attribute should be set to the bin directory of the PVCS installation containing the executables mentioned before. If this attribute isn't specified the tag expects the executables to be found using the PATH environment variable.	No
ignorereturncode	If set to *true* the return value from executing the pvcs commands are ignored.	No
updateonly	If set to *true* files are gotten only if newer than existing local files.	No
filenameformat	The format of your folder names in a format suitable for `java.text.MessageFormat`. Defaults to `{0}-arc({1})`. Repositories where the archive extension is not `-arc` should set this.	No
linestart	Used to parse the output of the pcli command. It defaults to `"P:`. The parser already knows about / and \\, this property is useful in cases where the repository is accessed on a Windows platform via a drive letter mapping.	No
revision	Retrieve the specified revision.	No
userid	Use the specified userid.	No

Nested Elements

pvcsproject element

`pvcs` supports a nested `<pvcsproject>` element, that represents a project within the PVCS repository to extract files from. By nesting multiple `<pvcsproject>` elements under the `<pvcs>` task, multiple projects can be specified.

Parameters

Attribute	Description	Required
name	The name of the pvcs project	Yes

Examples

The following set-up extracts the latest version of the files in the pvcs repository.

```
  <!-- ================================================================= -->
  <!-- Get the latest version                                            -->
  <!-- ================================================================= -->
```

```
  <target name="getlatest">
    <pvcs repository="/mnt/pvcs" pvcsproject="/myprj"/>
  </target>
```

Now run:

```
ant getlatest
```

This will cause the following output to appear:

```
  getlatest:
  [pvcs] PVCS Version Manager (VMGUI) v6.6.10 (Build 870) for Windows NT/80x86
  [pvcs] Copyright 1985-2000 MERANT. All rights reserved.
  [pvcs] PVCS Version Manager (get) v6.6.10 (Build 870) for Windows NT/80x86
  [pvcs] Copyright 1985-2000 MERANT. All rights reserved.
  [pvcs] c:\myws\myprj\main.java <- C:\mypvcs\archives\myprj\main.java-arc
  [pvcs] rev 1.1
  [pvcs] c:\myws\myprj\apache\tool.java <- C:\mypvcs\archives\myprj\apache\tools.java-arc
  [pvcs] rev 1.5

  BUILD SUCCESSFUL

  Total time: 19 seconds
```

This next example extracts the latest version of the files in the pvcs repository from two projects using nested `<pvcsproject>` elements.

```
  <!-- ===================================================================-->
  <!-- Get latest from myprj and myprj2                                   -->
  <!-- ===================================================================-->
  <target name="getlatest2">
    <pvcs repository="/mnt/pvcs">
      <pvcsproject name="/myprj"/>
      <pvcsproject name="/myprj2"/>
    </pvcs>
  </target>
```

Now run:

```
ant getlatest2
```

This will cause the following output to appear:

```
  getlatest2:
  [pvcs] PVCS Version Manager (VMGUI) v6.6.10 (Build 870) for Windows NT/80x86
  [pvcs] Copyright 1985-2000 MERANT.  All rights reserved.
  [pvcs] PVCS Version Manager (get) v6.6.10 (Build 870) for Windows NT/80x86
  [pvcs] Copyright 1985-2000 MERANT.  All rights reserved.
  [pvcs] c:\myws\myprj\main.java <- C:\mypvcs\archives\myprj\main.java-arc
  [pvcs] rev 1.1
  [pvcs] c:\myws\myprj\apache\tool.java <- C:\mypvcs\archives\myprj\apache\tool.java-arc
  [pvcs] rev 1.5
  [pvcs] c:\myws\myprj2\apache\tool2.java <- C:\mypvcs\archives\myprj2\apache\tool2.java-arc
  [pvcs] rev 1.2

  BUILD SUCCESSFUL

  Total time: 22 seconds
```

SourceOffSite Tasks User Manual

by

- Jesse Stockall

Version 1.1 2002/01/23

Contents

Introduction

These tasks provide an interface to the Microsoft Visual SourceSafe SCM via SourceGear's SourceOffSite product. SourceOffSite is an add-on to Microsoft's VSS, that allows remote development teams and tele-commuters that need fast and secure read/write access to a centralized SourceSafe database via any TCP/IP connection. SOS provides Linux ,Solaris & Windows clients. The `org.apache.tools.ant.taskdefs.optional.sos` package consists of a simple framework to support SOS functionality as well as some Ant tasks encapsulating frequently used SOS commands. Although it is possible to use these commands on the desktop, they were primarily intended to be used by automated build systems. These tasks have been tested with SourceOffSite version 3.5.1 connecting to VisualSourceSafe 6.0. The tasks have been tested with Linux, Solaris & Windows2000.

The Tasks

sosget	Retrieves a read-only copy of the specified project or file.
soslabel	Assigns a label to the specified project.
soscheckin	Updates VSS with changes made to a checked out file or project, and unlocks the VSS master copy.
soscheckout	Retrieves a read-write copy of the specified project or file, locking the VSS master copy

---

Task Descriptions

SOSGet

Description

Task to perform GET commands with SOS

Parameters

Attribute	Values	Required
soscmd	Directory which contains soscmd(.exe) soscmd(.exe) must be in the path if this is not specified	No
vssserverpath	path to the srcsafe.ini - eg. \\server\vss\srcsafe.ini	Yes
sosserverpath	address & port of the SOS server - eg. 192.168.0.1:8888	Yes
projectpath	SourceSafe project path - eg. $/SourceRoot/Project1	Yes
file	Filename to act upon If no file is specified then act upon the project	No
username	SourceSafe username	Yes
password	SourceSafe password	No
localpath	Override the working directory and get to the specified path	No
soshome	The path to the SourceOffSite home directory	No
nocompress	true or false - disable compression	No
recursive	true or false - Only works with the GetProject command	No
version	a version number to get - Only works with the GetFile command	No
label	a label version to get - Only works with the GetProject command	No
nocache	true or false - Only needed if SOSHOME is set as an environment variable	No
verbose	true or false - Status messages are displayed	No

Example

```
<sosget verbose="true"
        recursive="true"
        username="build"
        password="build"
        localpath="tmp"
        projectpath="$/SourceRoot/project1"
        sosserverpath="192.168.10.6:8888"
        vssserverpath="d:\vss\srcsafe.ini"/>
```

Connects to a SourceOffsite server on 192.168.10.6:8888 with build,build as the username & password. The SourceSafe database resides on the same box as the SOS server & the VSS database is at "d:\vss\srcsafe.ini" Does a recursive GetProject on $/SourceRoot/project1, using tmp as the working directory.

SOSLabel

Description

Task to perform Label commands with SOS

Parameters

Attribute	Values	Required
soscmd	Directory which contains soscmd(.exe) soscmd(.exe) must be in the path if this is not specified	No
vssserverpath	path to the srcsafe.ini - eg. \\server\vss\srcsafe.ini	Yes
sosserverpath	address and port of the SOS server - eg. 192.168.0.1:8888	Yes
projectpath	SourceSafe project path - eg. $/SourceRoot/Project1	Yes
username	SourceSafe username	Yes
password	SourceSafe password	No
label	The label to apply to a project	Yes
comment	A comment to be applied to all files being labeled	No
verbose	true or false - Status messages are displayed	No

Example

```
<soslabel username="build"
          password="build"
          label="test label"
          projectpath="$/SourceRoot/project1"
          sosserverpath="192.168.10.6:8888"
          vssserverpath="d:\vss\srcsafe.ini"/>
```

Connects to a SourceOffsite server on 192.168.10.6:8888 with build,build as the username & password. The SourceSafe database resides on the same box as the SOS server & the VSS database is at "d:\vss\srcsafe.ini". Labels the $/SourceRoot/project1 project with "test label".

SOSCheckIn

Description

Task to perform CheckIn commands with SOS

Parameters

Attribute	Values	Required
soscmd	Directory which contains soscmd(.exe) soscmd(.exe) must be in the path if this is not specified	No
vssserverpath	path to the srcsafe.ini - eg. \\server\vss\srcsafe.ini	Yes
sosserverpath	address and port of the SOS server - eg. 192.168.0.1:8888	Yes
projectpath	SourceSafe project path - eg. $/SourceRoot/Project1	Yes
file	Filename to act upon If no file is specified then act upon the project	No
username	SourceSafe username	Yes
password	SourceSafe password	No

localpath	Override the working directory and get to the specified path	No
soshome	The path to the SourceOffSite home directory	No
nocompress	true or false - disable compression	No
recursive	true or false - Only works with the CheckOutProject command	No
nocache	true or false - Only needed if SOSHOME is set as an environment variable	No
verbose	true or false - Status messages are displayed	No
comment	A comment to be applied to all files being checked in	No

Example

```
<soscheckin username="build"
            password="build"
            file="foobar.txt"
            verbose="true"
            comment="comment abc"
            projectpath="$/SourceRoot/project1"
            sosserverpath="server1:8888"
            vssserverpath="\\server2\vss\srcsafe.ini"/>
```

Connects to a SourceOffsite server on server1:8888 with build,build as the username & password. The SourceSafe database resides on a different box (server2) & the VSS database is on a share called "vss". Checks-in only the "foobar.txt" file adding a comment of "comment abc". Extra status messages will be displayed on screen.

SOSCheckOut

Description

Task to perform CheckOut commands with SOS

Parameters

Attribute	Values	Required
soscmd	Directory which contains soscmd(.exe) soscmd(.exe) must be in the path if this is not specified	No
vssserverpath	path to the srcsafe.ini - eg. \\server\vss\srcsafe.ini	Yes
sosserverpath	address and port of the SOS server - eg. 192.168.0.1:8888	Yes
projectpath	SourceSafe project path - eg. $/SourceRoot/Project1	Yes
file	Filename to act upon If no file is specified then act upon the project	No
username	SourceSafe username	Yes
password	SourceSafe password	No
localpath	Override the working directory and get to the specified path	No
soshome	The path to the SourceOffSite home directory	No
nocompress	true or false - disable compression	No
recursive	true or false - Only works with the CheckOutProject command	No

nocache	true or false - Only needed if SOSHOME is set as an environment variable	No
verbose	true or false - Status messages are displayed	No

Example

```
<soscheckout soscmd="/usr/local/bin"
             verbose="true"
             username="build"
             password="build"
             projectpath="$/SourceRoot/project1"
             sosserverpath="192.168.10.6:8888"
             vssserverpath="\\server2\vss\srcsafe.ini"/>
```

Connects to a SourceOffsite server on server1:8888 with build,build as the username & password. The SourceSafe database resides on a different box (server2) & the VSS database is on a share called "vss". Checks-out "project1", Only the "project1" directory will be locked as the recursive option was not set. Extra status messages will be displayed on screen. The soscmd(.exe) file to be used resides in /usr/local/bin.

StarTeam Support

- STCheckout
- STCheckin
- STLabel
- STList
- *StarTeam (deprecated)*

The StarTeam revision control system was recently acquired by Borland. These tasks make use of functions from the StarTeam API to work with that system. As a result they are only available to licensed users of StarTeam. You must have `starteam-sdk.jar` in your classpath to run these tasks. For more information about the StarTeam API and how to license it, see the Borland web site.

All the StarTeam task names are in lower case.

Important Note on Installation and Licensing:

On Windows machines, the mere presence of `starteam-sdk.jar` on the classpath is not sufficient for getting these tasks to work properly. These tasks also require a fully-installed and fully-licensed version of the StarGate Runtime. This is part of a StarTeam client installation or may be installed separately. The full client install is not required. In particular, the Windows path must include the directory where the StarGate Runtime `.dll` files are installed.

Earlier versions of Ant (prior to 1.5.2) did not have this restriction because they were not as dependent on the StarTeam runtime - which the newer versions use to access StarTeam file status information. The older versions lacked this important capability.

Common Parameters for All Starteam Tasks

The following parameters, having to do with making the connection to a StarTeam project, are common to all the following tasks except the deprecated *StarTeam* task.

Attribute	Description	Required
username	The username of the account used to log in to the StarTeam server.	yes
password	The password of the account used to log in to the StarTeam server.	yes
URL	A string of the form `servername:portnum/project/view` which enables user to set all of these elements in one string.	Either this ...
servername	The name of the StarTeam server.	... or **all four** of these must be defined.
serverport	The port number of the StarTeam server.	
projectname	The name of the StarTeam project on which to operate.	
viewname	The name of the view in the StarTeam project on which to operate.	

STCheckout

Description

Checks out files from a StarTeam project.

The *includes* and *excludes* attributes function differently from other tasks in Ant. Inclusion/exclusion by folder is NOT supported.

Parameters

See also the required common StarTeam parameters.

Attribute	Description	Required
rootstarteamfolder	The root of the subtree in the StarTeam repository from which to check out files. Defaults to the root folder of the view ('/'). ***If supplied, this should always be an "absolute" path, that is, it should begin with a '/'. Relative paths have little meaning in this context and confuse StarTeam.***	no
rootlocalfolder	The local folder which will be the root of the tree to which files are checked out. If this is not supplied, then the StarTeam "default folder" associated with *rootstarteamfolder* is used.	no
createworkingdirs	creates local folders even when the corresponding StarTeam folder is empty. Defaults to "true".	no
deleteuncontrolled	if true, any files NOT in StarTeam will be deleted. Defaults to "true".	no
includes	Only check out files that match at least one of the patterns in this list. Patterns must be separated by *commas*. Patterns in *excludes* take precedence over patterns in *includes*.	no
excludes	Do not check out files that match at least one of the patterns in this list. Patterns must be separated by *commas*. Patterns in *excludes* take precedence over patterns in *includes*.	no
label	Check out files as of this label. The label must exist in starteam or an exception will be thrown.	Either or neither, but not both, may be specified. Neither `locked` or `unlocked` may be true if either `label` or `asofdate` is specified.
asofdate	Check out files as of this date. The date must be formatted in ISO8601 datetime (`yyyy-MM-dd'T'HH:mm:ss`), ISO8601 date(`yyyy-MM-dd`) or a user-defined SimpleDateFormat defined in the `asofDateFormat` attribute. If the date is not parsable by the default or selected format, an exception will be thrown. *Since Ant 1.6.*	
asofdateformat	java.util.SimpleDateFormat compatible string used to parse the `asofdate` attribute. *Since Ant 1.6.*	no
recursive	Indicates if subfolders should be searched for files to check out. Defaults to "true".	no
forced	If true, checkouts will occur regardless of the status that StarTeam is maintaining for the file. If false, status will be used to determine which files to check out. Defaults to "false".	no
locked	If true, file will be locked against changes by other users. If false (default) has no effect.	Either or neither, but not both, may be true. Neither may be true if a `label` or
unlocked	If true, file will be unlocked so that other users may change it.	

	This is a way to reverse changes that have not yet been checked in. If false (default) has no effect.	an `asofdate` is specified.
userepositorytimestamp	true means checked out files will get the repository timestamp. false(default) means the checked out files will be timestamped at the time of checkout.	no
preloadfileinformation	The StarTeam server has the ability to preload file metadata for an entire tree prior to beginning action on that tree. Doing so can in some instances lead to substantially faster actions, particularly over large trees. Setting this to "yes" (default) engages this functionality, setting it to "no" turns it off.	no
convertEOL	If true, (default) all ascii files will have their end-of-line characters adjusted to that of the local machine on checkout. This is normally what you'd want but if for some reason you don't want that to happen, set it to false and the files will be checked out with whatever end-of-line characters are used on the server.	no

Examples

```
<stcheckout servername="STARTEAM"
            serverport="49201"
            projectname="AProject"
            viewname="AView"
            username="auser"
            password="secret"
            rootlocalfolder="C:\dev\buildtest\co"
            forced="true"
/>
```

The minimum necessary to check out files out from a StarTeam server. This will check out all files in the *AView* view of the *AProject* project to `C:\dev\buildtest\co`. Empty folders in StarTeam will have local folders created for them and any non-StarTeam files found in the tree will be deleted.

```
<stcheckout URL="STARTEAM:49201/Aproject/AView"
            username="auser"
            password="secret"
            rootlocalfolder="C:\dev\buildtest\co"
            forced="true"
/>
```

And this is a simpler way of accomplishing the same thing as the previous example, using the URL attribute.

```
<stcheckout URL="STARTEAM:49201/Aproject/AView"
            username="auser"
            password="secret"
            rootlocalfolder="C:\dev\buildtest\co"
            rootstarteamfolder="\Dev"
            excludes="*.bak *.old"
            label="v2.6.001"
            forced="true"
/>
```

This will check out all files from the *Dev* folder and below that do not end in *.bak* or *.old* with the label *v2.6.001*.

```
<stcheckout URL="STARTEAM:49201/Aproject/AView"
            username="auser"
            password="secret"
            rootlocalfolder="C:\dev\buildtest\co"
            includes="*.htm,*.html"
            excludes="index.*"
            forced="true"
/>
```

This is an example of overlapping *includes* and *excludes* attributes. Because *excludes* takes precedence over *includes*,

files named `index.html` will not be checked out by this command.

```
  <stcheckout URL="STARTEAM:49201/Aproject/AView"
              username="auser"
              password="secret"
              rootlocalfolder="C:\dev\buildtest\co"
              includes="*.htm,*.html"
              excludes="index.*"
              forced="true"
              recursive="false"
  />
```

This example is like the previous one, but will only check out files in C:\dev\buildtest\co, because of the turning off of the recursive attribute.

```
  <stcheckout URL="STARTEAM:49201/Aproject/AView"
              username="auser"
              password="secret"
              rootstarteamfolder="src/java"
              rootlocalfolder="C:\dev\buildtest\co"
              forced="true"
  />
```

```
  <stcheckout URL="STARTEAM:49201/Aproject/AView"
              username="auser"
              password="secret"
              rootstarteamfolder="src/java"
  />
```

```
 <stcheckout URL="STARTEAM:49201/Aproject/AView"
             username="auser"
             password="secret"
             rootstarteamfolder="src/java"
             rootlocalfolder="C:\dev\buildtest\co\src\java"
             forced="true"
  />
```

In the preceding three examples, assuming that the AProject project has a default folder of "C:\work\AProject", the first example will check out the tree of files rooted in the src/java folder of the AView view of the AProject in the StarTeam repository to a local tree rooted at C:\dev\buildtest\co, the second to a tree rooted at C:\work\AProject\src\java (since no *rootlocalfolder* is specified) and the third to a tree rooted at C:\dev\buildtest\co\src\java. Note also, that since the second example does not set "forced" true, only those files which the repository considers out-of-date will be checked out.

STCheckin

Description

Checks files into a StarTeam project. Optionally adds files and in the local tree that are not managed by the repository to its control.

The *includes* and *excludes* attributes function differently from other tasks in Ant. Inclusion/exclusion by folder is NOT supported.

Parameters

See also the required common StarTeam parameters.

Attribute	Description	Required
rootstarteamfolder	The root of the subtree in the StarTeam repository into which to files will be checked. Defaults to the root folder of the view ('/'). ***If supplied, this should always be an "absolute" path, that is, it should begin with a '/'. Relative paths have little meaning in this context and confuse StarTeam.***	no
rootlocalfolder	The local folder which will be the root of the tree to which files are checked out. If this is not supplied, then the StarTeam "default folder" associated with *rootstarteamfolder* is used.	no
comment	Checkin comment to be saved with the file.	no
adduncontrolled	if true, any files or folders NOT in StarTeam will be added to the repository. Defaults to "false".	no
includes	Only check in files that match at least one of the patterns in this list. Patterns must be separated by *commas*. Patterns in *excludes* take precedence over patterns in *includes*.	no
excludes	Do not check in files that match at least one of the patterns in this list. Patterns must be separated by *commas*. Patterns in *excludes* take precedence over patterns in *includes*.	no
recursive	Indicates if subfolders should be searched for files to check in. Defaults to "false".	no
forced	If true, checkins will occur regardless of the status that StarTeam is maintaining for the file. If false, checkins will use this status to decide which files to update. Defaults to "false".	no
unlocked	If true, file will be unlocked so that other users may change it. If false (default) lock status will not change.	no
preloadfileinformation	The StarTeam server has the ability to preload file metadata for an entire tree prior to beginning action on that tree. Doing so can in some instances lead to substantially faster actions, particularly over large trees. Setting this to "yes" (default) engages this functionality, setting it to "no" turns it off.	no

Examples

```
<stcheckin servername="STARTEAM"
           serverport="49201"
           projectname="AProject"
           viewname="AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           forced="true"
/>
```

The minimum necessary to check files into a StarTeam server. This will check all files on the local tree rooted at `C:\dev\buildtest\co` into the *AView* view of the *AProject* project in the repository. For files and folders in the local tree but not in starteam, nothing will be done. Since the *forced* attribute is set, the files which are checked in will be checked in without regard to what the StarTeam repository considers their status to be. This is a reasonable choice of attributes since StarTeam's status for a file is calculated based on the local file in the StarTeam default directory, not on the directory we are actually working with.

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           forced="true"
/>
```

And this is a simpler way of giving the same commands as the command above using the URL shortcut.

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           rootstarteamfolder="\Dev"
           excludes="*.bak *.old"
           forced="true"
/>
```

This will check all files in to the *Dev* folder and below that do not end in *.bak* or *.old* from the tree rooted at"C:\dev\buildtest\co" .

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           includes="*.htm,*.html"
           excludes="index.*"
           forced="true"
/>
```

This is an example of overlapping *includes* and *excludes* attributes. Because *excludes* takes precedence over *includes*, files named `index.html` will not be checked in by this command.

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           rootstarteamfolder="src/java"
           includes="*.htm,*.html"
           excludes="index.*"
           forced="true"
           recursive="false"
/>
```

This example is like the previous one, but will only check in files from C:\dev\buildtest\co, because of the turning off of the recursive attribute.

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           rootstarteamfolder="src/java"
           includes="version.txt"
           forced="true"
           recursive="false"
/>
```

This example is like the previous one, but will only check only in one file, C:\dev\buildtest\co\version.txt to the StarTeam folder src/java.

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
           username="auser"
           password="secret"
           rootlocalfolder="C:\dev\buildtest\co"
           rootstarteamfolder="src/java"
           includes="version.java"
           forced="true"
           recursive="false"
           addUncontrolled="true"
           comment="Fix Bug #667"
/>
```

This example is like the previous one, but will only check only in one file, C:\dev\buildtest\co\version.java to the StarTeam folder src/java. Because the *addUncontrolled* attribute has been set, if StarTeam does not already control this file in this location, it will be added to the repository. Also, it will write a comment to the repository for this version of the file.

```
<stcheckin URL="STARTEAM:49201/Aproject/AView"
```

```
            username="auser"
            password="secret"
            rootstarteamfolder="src/java"
            rootlocalfolder="C:\dev\buildtest\co"
            forced="true"
  />
```

```
  <stcheckin URL="STARTEAM:49201/Aproject/AView"
            username="auser"
            password="secret"
            rootstarteamfolder="src/java"
  />
```

```
 <stcheckin URL="STARTEAM:49201/Aproject/AView"
            username="auser"
            password="secret"
            rootstarteamfolder="src/java"
            rootlocalfolder="C:\dev\buildtest\co\src\java"
            forced="true"
  />
```

In the preceding three examples, assuming that the AProject project has a default folder of C:\work\buildtest\co\AProject, the first example will check in files from a tree rooted at C:\dev\buildtest\co, the second from a tree rooted at C:\work\buildtest\co\AProject\src\java, and the third from a tree rooted at C:\dev\buildtest\co\src\java all to a tree rooted at src/java

---

STLabel

Description

Creates a view label in StarTeam at the specified view. The label will be classified by StarTeam as a "build label". This task will fail if there already exists in *viewname* a label with the same name as the *label* parameter.

Parameters

See also the required common StarTeam parameters.

Attribute	Description	Required
label	The name to be given to the label	yes
description	A description of the label to be stored in the StarTeam project.	yes
revisionlabel	Yes means that the label attribute is to be saved as a "revision label". No (default) means that it will be saved as a "view label"	no
buildlabel	Yes means that the label attribute is to be saved as a "build label". This means that Change Requests which have an "AddressedIn" field value of "next build" will have this label assigned to that field when the label is created. No (default) means that no CRs will have this label assigned to them. This will have no effect if **revisionlabel** is also true.	no
lastbuild	The timestamp of the build that will be stored with the label. Must be formatted `yyyyMMddHHmmss`	no

Examples

This example shows the use of this tag. It will create a View label that is a build label named *Version 6.2* with *"Thorough description"* as its description.

```
<tstamp>
  <format property="nowstamp" pattern="yyyyMMddHHmmss" locale="en"/>
</tstamp>
<stlabel URL="STARTEAM:49201/Aproject/AView"
         username="auser"
         password="secret"
         label="Version 6.2"
         lastbuild="${nowstamp}"
         description="Thorough description"
/>
```

This example creates a non-build View label named *Version 6.3* with *"Thorough description"* as its description.

```
<tstamp>
  <format property="nowstamp" pattern="yyyyMMddHHmmss" locale="en"/>
</tstamp>
<stlabel URL="STARTEAM:49201/Aproject/AView"
         username="auser"
         password="secret"
         label="Version 6.3"
         lastbuild="${nowstamp}"
         description="Thorough description"
         buildlabel="false"
/>
```

This example will create a Revision label that is a build label named *Version 6.2.00.001* with *"revision label"* as its description.

```
<tstamp>
  <format property="nowstamp" pattern="yyyyMMddHHmmss" locale="en"/>
</tstamp>
<stlabel URL="STARTEAM:49201/Aproject/AView"
         username="auser"
         password="secret"
         label="Version 6.2.00.001"
         description="revision label"
         revisionlabel="true"
/>
```

STList

Description

Produces a listing of the contents of the StarTeam repository at the specified view and StarTeamFolder. The listing will contain the name of the user, if any, who has the file locked, the size of the file, its lastModifiedDate in the repository, the name of the file, and the status of the local file in the default local directory relative to the repository.

Parameters

See also the required common StarTeam parameters.

Attribute	Description	Required
rootstarteamfolder	The root of the subtree in the StarTeam repository to be listed. Defaults to the root folder of the view ('/'). ***If supplied, this should always be an "absolute" path, that is, it should begin with a '/'. Relative paths have little meaning in this context and***	no

	confuse StarTeam.	
rootlocalfolder	The local folder which will be the root of the tree to which files are compared. If this is not supplied, then the StarTeam "default folder" associated with *rootstarteamfolder* is used and a status field will appear in the listing. Otherwise, the status field will not appear.	no
includes	Only list files that match at least one of the patterns in this list. Patterns must be separated by *commas*. Patterns in *excludes* take precedence over patterns in *includes*.	no
excludes	Do not list files that match at least one of the patterns in this list. Patterns must be separated by *commas*. Patterns in *excludes* take precedence over patterns in *includes*.	no
label	List files, dates, and statuses as of this label. The label must exist in starteam or an exception will be thrown. If not specified, the most recent version of each file will be listed.	no
asofdate	List files, dates, and statuses as of this date. The date must be formatted in ISO8601 datetime (`yyyy-MM-dd'T'HH:mm:ss`), ISO8601 date(`yyyy-MM-dd`) or a user-defined SimpleDateFormat defined in the `asofDateFormat` attribute. If the date is not parsable by the default or selected format, an exception will be thrown. *Since Ant 1.6.*	no
asofdateformat	java.util.SimpleDateFormat compatible string used to parse the `asofdate` attribute. *Since Ant 1.6.*	no
recursive	Indicates if subfolders should be searched for files to list. Defaults to "true".	no
listuncontrolled	if true, any files or folders NOT in StarTeam will be included in the listing. If false, they won't. Defaults to "true".	no
preloadfileinformation	The StarTeam server has the ability to preload file metadata for an entire tree prior to beginning action on that tree. Doing so can in some instances lead to substantially faster actions, particularly over large trees. Setting this to "yes" (default) engages this functionality, setting it to "no" turns it off.	no

Examples

```
 <stlist url="WASHINGTON:49201/build"
         username="auser"
         password="secret"
/>
```

The above command might produce the following listing:

```
   [stlist] Folder: Build (Default folder: C:/work/build)
   [stlist] Folder: dev (Default folder: C:/work/build/dev)
   [stlist] Out of date  Sue Developer        1/1/02 7:25:47 PM CST     4368 build.xml
   [stlist] Missing      George Hacker        1/1/02 7:25:49 PM CST       36 Test01.properties
   [stlist] Current                           1/1/02 7:25:49 PM CST     4368 build2.xml
   [stlist] Folder: test (Default folder C:/work/build/dev/test)
   [stlist] Missing                           1/1/02 7:25:50 PM CST     4368 build2.xml
```

while adding a *rootlocalfolder* and an *excludes* param ...

```
 <stlist url="WASHINGTON:49201/build"
         username="auser"
         password="secret"
         rootlocalfolder="srcdir2"
         excludes="*.properties"
/>
```

might produce this listing. The status is missing because we are not going against the default folder.

```
   [stlist] overriding local folder to srcdir2
   [stlist] Folder: Build (Local folder: srcdir2)
   [stlist] Folder: dev (Local folder: srcdir2/dev)
   [stlist] Sue Developer        1/1/02 7:25:47 PM CST     4368 build.xml
   [stlist]                      1/1/02 7:25:49 PM CST     4368 build2.xml
   [stlist] Folder: test (Local folder: srcdir2/dev/test)
   [stlist]                      1/1/02 7:25:50 PM CST     4368 build2.xml
```

Starteam

Deprecated

This task has been deprecated. Use the STCheckout task instead.

Description

Checks out files from a StarTeam project.

The *includes* and *excludes* attributes function differently from other tasks in Ant. Multiple patterns must be separated by spaces, not commas. See the examples for more information.

Parameters

Attribute	Description	Required
username	The username of the account used to log in to the StarTeam server.	yes
password	The password of the account used to log in to the StarTeam server.	yes
servername	The name of the StarTeam server.	yes
serverport	The port number of the StarTeam server.	yes
projectname	The name of the StarTeam project.	yes
viewname	The name of the view in the StarTeam project.	yes
targetfolder	The folder to which files are checked out. What this precisely means is determined by the *targetFolderAbsolute* param.	yes
targetFolderAbsolute	Determines how *targetfolder* is interpreted, that is, whether the StarTeam "default folder" for the project is factored in (false) or whether *targetFolder* is a complete mapping to *foldername* (true). If "true", the target tree will be rooted at *targetfolder+"default folder"*. If false, the target tree will be rooted at *targetfolder*. Defaults to "false".	no
foldername	The subfolder in the project from which to check out files.	no
force	Overwrite existing folders if this is set to "true". Defaults to "false".	no
recursion	Indicates if subfolders should be searched for files to check out. Defaults to "true".	no
verbose	Provides progress information. Defaults to "false".	no
includes	Only check out files that match at least one of the patterns in this list. Patterns must be separated by spaces. Patterns in *excludes* take precedence over patterns in *includes*.	no
excludes	Do not check out files that match at least one of the patterns in this list. Patterns	no

	must be separated by spaces. Patterns in *excludes* take precedence over patterns in *includes*.	

Examples

```
  <starteam servername="STARTEAM"
            serverport="49201"
            projectname="AProject"
            viewname="AView"
            username="auser"
            password="secret"
            targetfolder="C:\dev\buildtest\co"
  />
```

The minimum necessary to check out files out from a StarTeam server. This will check out all files in the *AView* view of the *AProject* project to `C:\dev\buildtest\co`.

```
  <starteam servername="STARTEAM"
            serverport="49201"
            projectname="AProject"
            viewname="AView"
            username="auser"
            password="secret"
            targetfolder="C:\dev\buildtest\co"
            foldername="\Dev"
            excludes="*.bak *.old"
            force="true"
  />
```

This will checkout all files from the *Dev* folder and below that do not end in *.bak* or *.old*. The force flag will cause any existing files to be overwritten by the version in StarTeam.

```
  <starteam servername="STARTEAM"
            serverport="49201"
            projectname="AProject"
            viewname="AView"
            username="auser"
            password="secret"
            targetfolder="C:\dev\buildtest\co"
            includes="*.htm *.html"
            excludes="index.*"
  />
```

This is an example of overlapping *includes* and *excludes* attributes. Because *excludes* takes precedence over *includes*, files named `index.html` will not be checked out by this command.

```
 <starteam servername="STARTEAM"
            serverport="49201"
            projectname="AProject"
            foldername="src/java"
            viewname="AView"
            username="auser"
            password="secret"
            targetfolder="C:\dev\buildtest\co"
            targetfolderabsolute="true"
  />
```

```
 <starteam servername="STARTEAM"
            serverport="49201"
            projectname="AProject"
            foldername="src/java"
            viewname="AView"
            username="auser"
            password="secret"
            targetfolder="C:\dev\buildtest\co"
            targetfolderabsolute ="false"
  />
```

```
<starteam servername="STARTEAM"
           serverport="49201"
           projectname="AProject"
           foldername="src/java"
           viewname="AView"
           username="auser"
           password="secret"
           targetfolder="C:\dev\buildtest\co\src\java"
           targetfolderabsolute="true"
 />
```

In the preceding three examples, assuming that the AProject project has a default folder of "AProject", the first example will check the files located in starteam under src/java out to a tree rooted at C:\dev\buildtest\co, the second to a tree rooted at C:\dev\buildtest\co\AProject\src\java and the third to a tree rooted at C:\dev\buildtest\co\src\java.

JUnit

Description

This task runs tests from the JUnit testing framework. The latest version of the framework can be found at http://www.junit.org. This task has been tested with JUnit 3.0 up to JUnit 3.8.2; it won't work with versions prior to JUnit 3.0. It also works with JUnit 4.0, including "pure" JUnit 4 tests using only annotations and no `JUnit4TestAdapter`.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information.

Note: You must have `junit.jar` available. You can do one of:

1. Put both `junit.jar` and `ant-junit.jar` in `ANT_HOME/lib`.
2. Do not put either in `ANT_HOME/lib`, and instead include their locations in your `CLASSPATH` environment variable.
3. Add both JARs to your classpath using `-lib`.
4. Specify the locations of both JARs using a `<classpath>` element in a `<taskdef>` in the build file.
5. Leave `ant-junit.jar` in its default location in `ANT_HOME/lib` but include `junit.jar` in the `<classpath>` passed to `<junit>`. *(since Ant 1.7)*

See the FAQ for details.

Tests are defined by nested `test` or `batchtest` tags (see nested elements).

Parameters

Attribute	Description	Required
printsummary	Print one-line statistics for each testcase. Can take the values `on`, `off`, and `withOutAndErr`. `withOutAndErr` is the same as `on` but also includes the output of the test as written to `System.out` and `System.err`.	No; default is `off`.
fork	Run the tests in a separate VM.	No; default is `off`.
forkmode	Controls how many Java Virtual Machines get created if you want to fork some tests. Possible values are "perTest" (the default), "perBatch" and "once". "once" creates only a single Java VM for all tests while "perTest" creates a new VM for each TestCase class. "perBatch" creates a VM for each nested `<batchtest>` and one collecting all nested `<test>`s. Note that only tests with the same settings of `filtertrace`, `haltonerror`, `haltonfailure`, `errorproperty` and `failureproperty` can share a VM, so even if you set `forkmode` to "once", Ant may have to create more than a single Java VM. This attribute is ignored for tests that don't get forked into a new Java VM. *since Ant 1.6.2*	No; default is `perTest`.
haltonerror	Stop the build process if an error occurs during the test run.	No; default is `off`.
errorproperty	The name of a property to set in the event of an error.	No
haltonfailure	Stop the build process if a test fails (errors are considered failures as well).	No; default is

		`off`.
failureproperty	The name of a property to set in the event of a failure (errors are considered failures as well).	No.
filtertrace	Filter out Junit and Ant stack frames from error and failure stack traces.	No; default is `on`.
timeout	Cancel the individual tests if they don't finish in the given time (measured in milliseconds). Ignored if `fork` is disabled. When running multiple tests inside the same Java VM (see forkMode), timeout applies to the time that all tests use together, not to an individual test.	No
maxmemory	Maximum amount of memory to allocate to the forked VM. Ignored if `fork` is disabled. **Note**: If you get `java.lang.OutOfMemoryError: Java heap space` in some of your tests then you need to raise the size like `maxmemory="128m"`	No
jvm	The command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by `java.lang.Runtime.exec()`. Ignored if `fork` is disabled.	No; default is `java`.
dir	The directory in which to invoke the VM. Ignored if `fork` is disabled.	No
newenvironment	Do not propagate the old environment when new environment variables are specified. Ignored if `fork` is disabled.	No; default is `false`.
includeantruntime	Implicitly add the Ant classes required to run the tests and JUnit to the classpath in forked mode.	No; default is `true`.
showoutput	Send any output generated by tests to Ant's logging system as well as to the formatters. By default only the formatters receive the output.	No
outputtoformatters	*Since Ant 1.7.0.* Send any output generated by tests to the test formatters. This is "true" by default.	No
tempdir	Where Ant should place temporary files. *Since Ant 1.6.*	No; default is the project's base directory.
reloading	Whether or not a new classloader should be instantiated for each test case. Ignore if `fork` is set to true. *Since Ant 1.6.*	No; default is `true`.
clonevm	If set to true true, then all system properties and the bootclasspath of the forked Java Virtual Machine will be the same as those of the Java VM running Ant. Default is "false" (ignored if fork is disabled). *since Ant 1.7*	No
logfailedtests	When Ant executes multiple tests and doesn't stop on errors or failures it will log a "FAILED" message for each failing test to its logging system. If you set this option to false, the message will not be logged and you have to rely on the formatter output to find the failing tests. *since Ant 1.8.0*	No

By using the `errorproperty` and `failureproperty` attributes, it is possible to perform setup work (such as starting an external server), execute the test, clean up, and still fail the build in the event of a failure.

The `filtertrace` attribute condenses error and failure stack traces before reporting them. It works with both the plain and XML formatters. It filters out any lines that begin with the following string patterns:

```
    "junit.framework.TestCase"
    "junit.framework.TestResult"
    "junit.framework.TestSuite"
    "junit.framework.Assert."
    "junit.swingui.TestRunner"
    "junit.awtui.TestRunner"
    "junit.textui.TestRunner"
    "java.lang.reflect.Method.invoke("
    "sun.reflect."
    "org.apache.tools.ant."
    "org.junit."
    "junit.framework.JUnit4TestAdapter"
```

Nested Elements

The `<junit>` task supports a nested `<classpath>` element that represents a PATH like structure.

As of Ant 1.7, this classpath may be used to refer to `junit.jar` as well as your tests and the tested code.

jvmarg

If `fork` is enabled, additional parameters may be passed to the new VM via nested `<jvmarg>` elements. For example:

```
<junit fork="yes">
  <jvmarg value="-Djava.compiler=NONE"/>
  ...
</junit>
```

would run the test in a VM without JIT.

`<jvmarg>` allows all attributes described in Command-line Arguments.

sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the class. These properties will be made available to the VM during the execution of the test (either ANT's VM or the forked VM, if `fork` is enabled). The attributes for this element are the same as for environment variables.

```
<junit fork="no">
  <sysproperty key="basedir" value="${basedir}"/>
  ...
</junit>
```

would run the test in ANT's VM and make the `basedir` property available to the test.

syspropertyset

You can specify a set of properties to be used as system properties with syspropertysets.

since Ant 1.6.

env

It is possible to specify environment variables to pass to the forked VM via nested `<env>` elements. For a description of the `<env>` element's attributes, see the description in the exec task.

Settings will be ignored if `fork` is disabled.

bootclasspath

The location of bootstrap class files can be specified using this PATH like structure - will be ignored if *fork* is not `true` or the target VM doesn't support it (i.e. Java 1.1).

since Ant 1.6.

permissions

Security permissions can be revoked and granted during the execution of the class via a nested *permissions* element. For more information please see permissions

Settings will be ignored if fork is enabled.

since Ant 1.6.

assertions

You can control enablement of Java 1.4 assertions with an `<assertions>` subelement.

Assertion statements are currently ignored in non-forked mode.

since Ant 1.6.

formatter

The results of the tests can be printed in different formats. Output will always be sent to a file, unless you set the `usefile` attribute to `false`. The name of the file is determined by the name of the test and can be set by the `outfile` attribute of `<test>`.

There are four predefined formatters - one prints the test results in XML format, the other emits plain text. The formatter named `brief` will only print detailed information for testcases that failed, while `plain` gives a little statistics line for all test cases. Custom formatters that need to implement `org.apache.tools.ant.taskdefs.optional.junit.JUnitResultFormatter` can be specified.

If you use the XML formatter, it may not include the same output that your tests have written as some characters are illegal in XML documents and will be dropped.

The fourth formatter named `failure` (since Ant 1.8.0) collects all failing `testXXX()` methods and creates a new `TestCase` which delegates only these failing methods. The name and the location can be specified via Java System property or Ant property `ant.junit.failureCollector`. The value has to point to the directory and the name of the resulting class (without suffix). It defaults to *java-tmp-dir*/FailedTests.

Attribute	Description	Required
type	Use a predefined formatter (either `xml`, `plain`, `brief` or `failure`).	Exactly one of these.
classname	Name of a custom formatter class.	
extension	Extension to append to the output filename.	Yes, if `classname` has been used.
usefile	Boolean that determines whether output should be sent to a file.	No; default is `true`.

if	Only use formatter if the named property is set.	No; default is `true`.
unless	Only use formatter if the named property is **not** set.	No; default is `true`.

test

Defines a single test class.

Attribute	Description	Required
name	Name of the test class.	Yes
fork	Run the tests in a separate VM. Overrides value set in `<junit>`.	No
haltonerror	Stop the build process if an error occurs during the test run. Overrides value set in `<junit>`.	No
errorproperty	The name of a property to set in the event of an error. Overrides value set in `<junit>`.	No
haltonfailure	Stop the build process if a test fails (errors are considered failures as well). Overrides value set in `<junit>`.	No
failureproperty	The name of a property to set in the event of a failure (errors are considered failures as well). Overrides value set in `<junit>`.	No
filtertrace	Filter out Junit and Ant stack frames from error and failure stack traces. Overrides value set in `<junit>`.	No; default is `on`.
todir	Directory to write the reports to.	No; default is the current directory.
outfile	Base name of the test result. The full filename is determined by this attribute and the extension of `formatter`.	No; default is `TEST-`*name*, where *name* is the name of the test specified in the `name` attribute.
if	Only run test if the named property is set.	No
unless	Only run test if the named property is **not** set.	No

Tests can define their own formatters via nested `<formatter>` elements.

batchtest

Define a number of tests based on pattern matching.

`batchtest` collects the included resources from any number of nested Resource Collections. It then generates a test class name for each resource that ends in `.java` or `.class`.

Any type of Resource Collection is supported as a nested element, prior to Ant 1.7 only `<fileset>` has been supported.

Attribute	Description	Required
fork	Run the tests in a separate VM. Overrides value set in `<junit>`.	No
haltonerror	Stop the build process if an error occurs during the test run. Overrides value set in `<junit>`.	No
errorproperty	The name of a property to set in the event of an error. Overrides value set in `<junit>`.	No
haltonfailure	Stop the build process if a test fails (errors are considered failures as well). Overrides value set in `<junit>`.	No
failureproperty	The name of a property to set in the event of a failure (errors are considered failures as well). Overrides value set in `<junit>`	No
filtertrace	Filter out Junit and Ant stack frames from error and failure stack traces. Overrides value set in `<junit>`.	No; default is `on`.
todir	Directory to write the reports to.	No; default is the current directory.
if	Only run tests if the named property is set.	No
unless	Only run tests if the named property is **not** set.	No

Batchtests can define their own formatters via nested `<formatter>` elements.

Forked tests and `tearDown`

If a forked test runs into a timeout, Ant will terminate the Java VM process it has created, which probably means the test's `tearDown` method will never be called. The same is true if the forked VM crashes for some other reason.

Starting with Ant 1.8.0, a special formatter is distributed with Ant that tries to load the testcase that was in the forked VM and invoke that class' `tearDown` method. This formatter has the following limitations:

- It runs in the same Java VM as Ant itself, this is a different Java VM than the one that was executing the test and it may see a different classloader (and thus may be unable to load the tast class).
- It cannot determine which test was run when the timeout/crash occured if the forked VM was running multiple test. I.e. the formatter cannot work with any `forkMode` other than `perTest` and it won't do anything if the test class contains a `suite()` method.

If the formatter recognizes an incompatible `forkMode` or a `suite` method or fails to load the test class it will silently do nothing.

The formatter doesn't have any effect on tests that were not forked or didn't cause timeouts or VM crashes.

To enable the formatter, add a `formatter` like

```
<formatter classname="org.apache.tools.ant.taskdefs.optional.junit.TearDownOnVmCrash"
           usefile="false"/>
```

to your `junit` task.

Examples

```
<junit>
  <test name="my.test.TestCase"/>
</junit>
```

Runs the test defined in `my.test.TestCase` in the same VM. No output will be generated unless the test fails.

```
<junit printsummary="yes" fork="yes" haltonfailure="yes">
  <formatter type="plain"/>
  <test name="my.test.TestCase"/>
</junit>
```

Runs the test defined in `my.test.TestCase` in a separate VM. At the end of the test, a one-line summary will be printed. A detailed report of the test can be found in `TEST-my.test.TestCase.txt`. The build process will be stopped if the test fails.

```
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <pathelement location="${build.tests}"/>
    <pathelement path="${java.class.path}"/>
  </classpath>

  <formatter type="plain"/>

  <test name="my.test.TestCase" haltonfailure="no" outfile="result">
    <formatter type="xml"/>
  </test>

  <batchtest fork="yes" todir="${reports.tests}">
    <fileset dir="${src.tests}">
      <include name="**/*Test*.java"/>
      <exclude name="**/AllTests.java"/>
    </fileset>
  </batchtest>
</junit>
```

Runs `my.test.TestCase` in the same VM, ignoring the given CLASSPATH; only a warning is printed if this test fails. In addition to the plain text test results, for this test a XML result will be output to `result.xml`. Then, for each matching file in the directory defined for `${src.tests}` a test is run in a separate VM. If a test fails, the build process is aborted. Results are collected in files named `TEST-`*name*`.txt` and written to `${reports.tests}`.

```
<target name="test">
    <property name="collector.dir" value="${build.dir}/failingTests"/>
    <property name="collector.class" value="FailedTests"/>
    <!-- Delete 'old' collector classes -->
    <delete>
        <fileset dir="${collector.dir}" includes="${collector.class}*.class"/>
    </delete>
    <!-- compile the FailedTests class if present -->
    <javac srcdir="${collector.dir}" destdir="${collector.dir}"/>
    <available file="${collector.dir}/${collector.class}.class" property="hasFailingTests"/>
    <junit haltonerror="false" haltonfailure="false">
        <sysproperty key="ant.junit.failureCollector"
value="${collector.dir}/${collector.class}"/>
        <classpath>
            <pathelement location="${collector.dir}"/>
        </classpath>
        <batchtest todir="${collector.dir}" unless="hasFailingTests">
            <fileset dir="${collector.dir}" includes="**/*.java"
excludes="**/${collector.class}.*"/>
            <!-- for initial creation of the FailingTests.java -->
            <formatter type="failure"/>
            <!-- I want to see something ... -->
            <formatter type="plain" usefile="false"/>
        </batchtest>
        <test name="FailedTests" if="hasFailingTests">
            <!-- update the FailingTests.java -->
            <formatter type="failure"/>
            <!-- again, I want to see something -->
            <formatter type="plain" usefile="false"/>
        </test>
    </junit>
</target>
```

On the first run all tests are collected via the `<batchtest/>` element. It's `plain` formatter shows the output on the console. The `failure` formatter creates a java source file in `${build.dir}/failingTests/FailedTests.java` which extends `junit.framework.TestCase` and returns from a `suite()` method a test suite for the failing tests.
On a second run the collector class exists and instead of the `<batchtest/>` the single `<test/>` will run. So only the failing test cases are re-run. The two nested formatters are for displaying (for the user) and for updating the collector class.

JUnitReport

Merge the individual XML files generated by the JUnit task and eventually apply a stylesheet on the resulting merged document to provide a browsable report of the testcases results.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information.

Requirements

The task needs Apache Xalan 2.4.1+ or Xalan XSLTC (JDK 1.4 contains a version of Xalan-J 2.x while JDK 1.5 ships with a version of XSLTC). Starting from JDK 1.4.2-01 it ships with a bundled Xalan-J 2.4.1+, meaning that JDK version prior to 1.4.2-01 won't work out of the box. The table below summarize the compatibility status.

Xalan	Sun JDK Bundle	Status
2.4.1+	JDK 1.4.2-01+	OK
XSLTC	JDK 1.5.x	OK
2.x	JDK 1.4.x	DEPRECATED *Use ${ant.home}/etc/junit-frames-xalan1.xsl* *Upgrade Xalan using the JDK endorsement mechanism*

With Ant 1.6.2 we had to decide between supporting Xalan-J 1/Xalan J 2.4.1- and Xalan 2.4.1+/XSLTC, since there was no way to support both couples at the same time.

With Ant 1.7 we had to drop support Xalan-J 1, since Xalan-J 1 has not available anymore for quite some time.

Parameters

Attribute	Description	Required
tofile	The name of the XML file that will aggregate all individual XML testsuite previously generated by the JUnit task.	No. Default to TESTS-TestSuites.xml
todir	The directory where should be written the file resulting from the individual XML testsuite aggregation.	No. Default to current directory

Nested Elements

fileset

`junitreport` collects individual xml files generated by the JUnit task using the nested `<FileSet>` element.

report

Generate a browsable report based on the document created by the merge.

Parameters

Attribute	Description	Required
format	The format of the generated report. Must be "noframes" or "frames".	No, default to "frames"
styledir	The directory where the stylesheets are defined. They must be conforming to the following conventions: • frames format: the stylesheet must be named *junit-frames.xsl*. • noframes format: the stylesheet must be named *junit-noframes.xsl*.	No. Default to embedded stylesheets.
todir	The directory where the files resulting from the transformation should be written to.	No. Default to current directory

Ant assumes the following concerning the `frames` and `noframes` formats :

The `frames` format uses a stylesheet which is generating output *only* by redirecting.

The `noframes` format does not use redirecting and generates one file called `junit-noframes.html`.

Custom versions of `junit-frames.xsl` or `junit-noframes.xsl` must adhere to the above conventions.

Nested Element of the report tag

param

*Since Ant 1.7*the report tag supports nested param tags. These tags can pass XSL parameters to the stylesheet.

Parameters

Attribute	Description	Required
name	Name of the XSL parameter	Yes
expression	Text value to be placed into the param. Was originally intended to be an XSL expression.	Yes
if	The param will only be passed if this property is set.	No
unless	The param will not be passed if this property is set.	No

The built-in stylesheets support the following parameters:

XSL-Parameter	Description	Required
TITLE	Title used in <title> and <h1> tags	No. Defaults to *Unit Test Results.*

Example of report

```
<junitreport todir="./reports">
  <fileset dir="./reports">
    <include name="TEST-*.xml"/>
  </fileset>
  <report format="frames" todir="./report/html"/>
</junitreport>
```

would generate a `TESTS-TestSuites.xml` file in the directory `reports` and generate the default framed report in the directory `report/html`.

Example of report with xsl params

```
<junitreport todir="${outputdir}">
    <fileset dir="${jrdir}">
        <include name="TEST-*.xml"/>
    </fileset>
    <report todir="${outputdir}/html"
        styledir="junitreport"
        format="frames">
        <param name="key1" expression="value1"/>
        <param name="key2" expression="value2"/>
    </report>
</junitreport>
```

This example requires a file called `junitreport/junit-frames.xsl`. The XSL parameters key1 and key2 will be passed to the XSL transformation.

Antversion

Description

Stores the Ant version (when used as task) or checks for a specific Ant version (when used as condition). **Since Ant 1.7.0**

Attribute	Description	Required (Task)	Required (Condition)
atleast	The version that this at least. The format is major.minor.point.	No	One of these.
exactly	The version that this ant is exactly. The format is `major.minor.point`.	No	
property	The name of the property to set.	Yes	No (ignored)

Examples

```
<antversion property="antversion"/>
```

Stores the current Ant version in the property *antversion*.

```
<antversion property="antversion" atleast="1.6"/>
```

Stores the Ant version in the property *antversion* if the current Ant version is 1.6.0 or higher. Otherwise the property remains unset.

```
<antversion property="ant-is-exact-7" exactly="1.7.0"/>
```

Sets the property *ant-is-exact-7* if Ant 1.7.0 is running. Neither 1.6.5 nor 1.7.0 would match.

```
<condition property="Ant17isOnline">
  <and>
    <antversion exactly="1.7.0"/>
    <http url="http://ant.apache.org"/>
  </and>
</condition>
```

Sets *Ant17isOnline* if Ant 1.7.0 is running and can get a non-error-response from the Ant homepage.

Augment

Description

Modify an existing reference by adding nested elements or (re-)assigning properties mapped as XML attributes. This is an unusual task that makes use of Ant's internal processing mechanisms to reload a previously declared reference by means of the 'id' attribute, then treats the declared `augment` element as though it were the original element. **Since Ant 1.8.1**

Parameters

Attribute	Description	Required
id	The id of the reference to augment. If no such reference has been declared a `BuildException` is generated.	Yes

Additional permissible attributes are dependent on the reference to be modified.

Parameters specified as nested elements

Permissible nested elements are dependent on the reference to be modified.

Examples

Given

```
  <fileset id="input-fs" dir="${basedir}" />
```

```
  <augment id="input-fs" excludes="foo" />
```

Modifies the `excludes` attribute of `input-fs`.

```
  <augment id="input-fs">
    <filename name="bar" />
  </augment>
```

Adds a `filename` selector to `input-fs`.

Conditions

Conditions are nested elements of the `<condition>` and `<waitfor>` tasks. There are core conditions and custom conditions. Custom conditions are described in Custom Conditions. Core Conditions are described below.

Core Conditions

These are the nested elements that can be used as conditions in the `<condition>` and `<waitfor>` tasks.

not

The `<not>` element expects exactly one other condition to be nested into this element, negating the result of the condition. It doesn't have any attributes and accepts all nested elements of the condition task as nested elements as well.

and

The `<and>` element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements - all nested elements of the condition task are supported. This condition is true if all of its contained conditions are, conditions will be evaluated in the order they have been specified in the build file.

The `<and>` condition has the same shortcut semantics as the Java && operator, as soon as one of the nested conditions is false, no other condition will be evaluated.

or

The `<or>` element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements - all nested elements of the condition task are supported. This condition is true if at least one of its contained conditions is, conditions will be evaluated in the order they have been specified in the build file.

The `<or>` condition has the same shortcut semantics as the Java || operator, as soon as one of the nested conditions is true, no other condition will be evaluated.

xor

The `<xor>` element performs an exclusive or on all nested elements, similar to the ^ operator in Java. It only evaluates to true if an odd number of nested conditions are true. There is no shortcutting of evaluation, unlike the `<and>` and `<or>` tests. It doesn't have any attributes and accepts all nested elements of the condition task as nested elements as well.

available

This condition is identical to the Available task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

uptodate

This condition is identical to the Uptodate task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

os

Test whether the current operating system is of a given type. Each defined attribute is tested and the result is true only if *all* the tests succeed.

Attribute	Description	Required
family	The name of the operating system family to expect.	No
name	The name of the operating system to expect.	No
arch	The architecture of the operating system to expect.	No
version	The version of the operating system to expect.	No

Supported values for the family attribute are:

- windows (for all versions of Microsoft Windows)
- dos (for all Microsoft DOS based operating systems including Microsoft Windows and OS/2)
- mac (for all Apple Macintosh systems)
- unix (for all Unix and Unix-like operating systems)
- netware (for Novell NetWare)
- os/2 (for OS/2)
- tandem (for HP's NonStop Kernel - formerly Tandem)
- win9x for Microsoft Windows 95 and 98, ME and CE
- winnt for Microsoft Windows NT-based systems, including Windows 2000, XP and successors
- z/os for z/OS and OS/390
- os/400 for OS/400
- openvms for OpenVMS

equals

Tests whether the two given values are equal.

Attribute	Description	Required
arg1	First value to test.	Yes
arg2	Second value to test.	Yes
casesensitive	Perform a case sensitive comparision. Default is true.	No
trim	Trim whitespace from arguments before comparing them. Default is false.	No
forcestring	Force string comparison of `arg1/arg2`. Default is false. *Since Ant 1.8.1*	No

isset

Test whether a given property has been set in this project.

Attribute	Description	Required
property	The name of the property to test.	Yes

checksum

This condition is identical to the Checksum task, all attributes and nested elements of that task are supported, the property and overwrite attributes are redundant and will be ignored.

http

The `http` condition checks for a valid response from a web server of the specified url. By default, HTTP responses errors of 400 or greater are viewed as invalid.

Attribute	Description	Required
url	The full URL of the page to request. The web server must return a status code below the value of `errorsBeginAt`	Yes.
errorsBeginAt	The lowest HTTP response code that signals an error; by default '400'; server errors, not-authorized, not-found and the like are detected	No
requestMethod	The HTTP method to be used when issuing the request. Any of GET, POST, HEAD, OPTIONS, PUT, DELETEm and TRACE are valid, subject to protocol restrictions. The default if not specified is "GET". *since Ant 1.8.0*	No

socket

The `socket` condition checks for the existence of a TCP/IP listener at the specified host and port.

Attribute	Description	Required
server	The DNS name or IP address of the server.	Yes.
port	The port number to connect to.	Yes.

filesmatch

Test two files for matching. Nonexistence of one file results in "false", although if neither exists they are considered equal in terms of content. This test does a byte for byte comparision, so test time scales with byte size. NB: if the files are different sizes, one of them is missing or the filenames match the answer is so obvious the detailed test is omitted.

Attribute	Description	Required
file1	First file to test	Yes
file2	Second file to test	Yes
textfile	Whether to ignore line endings when comparing files; defaults to *false*, while *true* triggers a binary comparison. **Since Ant 1.7**	No

contains

Tests whether a string contains another one.

Attribute	Description	Required
string	The string to search in.	Yes
substring	The string to search for.	Yes
casesensitive	Perform a case sensitive comparision. Default is true.	No

istrue

Tests whether a string equals any of the ant definitions of true, that is "true","yes", or "on"

Attribute	Description	Required
value	value to test	Yes

```
<istrue value="${someproperty}"/>
<istrue value="false"/>
```

isfalse

Tests whether a string is not true, the negation of <istrue>

Attribute	Description	Required
value	value to test	Yes

```
<isfalse value="${someproperty}"/>
<isfalse value="false"/>
```

isreference

Test whether a given reference has been defined in this project and - optionally - is of an expected type.

This condition has been added in Apache Ant 1.6.

Attribute	Description	Required
refid	The id of the reference to test.	Yes
type	Name of the data type or task this reference is expected to be.	No

issigned

Test whether a jarfile is signed. If the name of the signature is passed, the file is checked for presence of that particular signature; otherwise the file is checked for the existence of any signature. It does not perform rigorous signature validation; it only looks for the presence of a signature.

This condition was added in Apache Ant 1.7.

Attribute	Description	Required
file	The jarfile that is to be tested for the presence of a signature.	Yes
name	The signature name to check for.	No

isfileselected

Test whether a file passes an embedded selector.

This condition was added in Apache Ant 1.6.3.

Attribute	Description	Required
file	The file to check if is passes the embedded selector.	Yes
basedir	The base directory to use for name based selectors. It this is not set, the project's basedirectory will be used.	No

Example usage:

```
<isfileselected file="a.xml">
  <date datetime="06/28/2000 2:02 pm" when="equal"/>
</isfileselected>
```

typefound

Test whether a given type is defined, and that its implementation class can be loaded. Types include tasks, datatypes, scriptdefs, macrodefs and presetdefs.

This condition was added in Apache Ant 1.7.

Attribute	Description	Required
name	name of the type	Yes
uri	The uri that this type lives in.	No

Example usages:

```
<typefound name="junit"/>
<typefound uri="antlib:org.apache.maven.artifact.ant" name="artifact"/>
```

scriptcondition

Evaluate a condition based on a script in any Apache BSF or JSR 223 supported language.

See the Script task for an explanation of scripts and dependencies.

This condition was added in Apache Ant 1.7.

Attribute	Description	Required
language	script language	Yes
manager	The script engine manager to use. See the script task for using this attribute.	No - default is "auto"
value	default boolean value	No -default is "false"
src	filename of script source	No
setbeans	whether to have all properties, references and targets as global variables in the script. *since Ant 1.8.0*	No, default is "true".
classpath	The classpath to pass into the script.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No

Parameters specified as nested elements

classpath

See the script task for using this nested element.

Description

The script supports script language inline, this script has access to the same beans as the `<script>` task, and to the `self` bean, which refers back to the condition itself. If the script evaluates to a boolean result, this is the result of the condition's evaluation (*since Ant 1.7.1*). Alternatively, `self.value` can be used to set the evaluation result.

Example:

```
<scriptcondition language="javascript"
        value="true">
    self.setValue(false);
</scriptcondition>
```

Sets the default value of the condition to true, then in the script, sets the value to false. This condition always evaluates to "false"

parsersupports

Tests whether Ant's XML parser supports a given feature or property, as per the SAX/JAXP specifications, by attempting to set the appropriate property/feature/

This condition was added in Apache Ant 1.7.

Attribute	Description	Required
property	property to set	one of property or feature
feature	feature to set	one of property or feature
value	string (property) or boolean (feature)	For property tests, but not for feature tests

```
<parsersupports feature="http://xml.org/sax/features/namespaces"/>
```

Check for namespace support. All SAX2 parsers should have this.

```
<or>
  <parsersupports
    feature="http://apache.org/xml/features/validation/schema"/>
  <parsersupports
    feature="http://java.sun.com/xml/jaxp/properties/schemaSource"/>
</or>
```

Check for XML Schema support.

```
<parsersupports
  property="http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation"
  value="document.xsd"/>
```

Check for Xerces-specific definition of the location of the no namespace schema.

isreachable

Uses Java1.5+ networking APIs to probe for a (remote) system being reachable. Exactly what probe mechanisms are used is an implementation feature of the JVM. They may include ICMP "ping" packets, UDP or TCP connections to port 7 "echo service" or other means. On Java1.4 and earlier, being able to resolve the hostname is considered success. This means that if DNS is not working or a URL/hostname is bad, the test will fail, but otherwise succeed even if the remote host is actually absent.

This condition turns unknown host exceptions into false conditions. This is because on a laptop, DNS is one of the first services when the network goes; you are implicitly offline.

If a URL is supplied instead of a host, the hostname is extracted and used in the test - all other parts of the URL are discarded.

The test may not work through firewalls, that is, something may be reachable using a protocol such as HTTP, while the lower level ICMP packets get dropped on the floor. Similarly, a host may detected as reachable with ICMP, but not reachable on other ports (i.e. port 80), because of firewalls.

This condition was added in Apache Ant 1.7.

Attribute	Description	Required
host	host to check for	one of url or host
url	URL containing hostname	one of url or host
timeout	timeout in seconds	no, default is 30s

```
<condition property="offline">
  <isreachable url="http://ibiblio.org/maven/" />
</condition>
```

Probe for the maven repository being reachable.

```
<condition property="offline">
  <isreachable host="ibiblio.org" timeout="10" />
</condition>
```

Probe for the maven repository being reachable using the hostname, ten second timeout..

length

This condition is a facet of the Length task. It is used to test the length of a string or one or more files. **Since Ant 1.6.3**

```
<length string=" foo " trim="true" length="3" />
```

Verify a string is of a certain length.

```
<length file="foo" when="greater" length="0" />
```

Verify that file *foo* is not empty.

isfailure

Test the return code of an executable (see the Exec task) for failure. **Since Ant 1.7**

Attribute	Description	Required
code	The return code to test.	Yes

resourcecount

This condition is a facet of the ResourceCount task. It is used to test the size of a resource collection. **Since Ant 1.7**

```
<resourcecount refid="myresourcecollection" when="greater" count="0" />
```

Verify that a resource collection is not empty.

resourcesmatch

Test resources for matching. Nonexistence of one or more resources results in "false", although if none exists they are considered equal in terms of content. By default this test does a byte for byte comparision, so test time scales with

byte size. NB: if the files are different sizes, one of them is missing or the filenames match the answer is so obvious the detailed test is omitted. The resources to check are specified as nested resource collections, meaning that more than two resources can be checked; in this case all resources must match. **Since Ant 1.7**

Attribute	Description	Required
astext	Whether to ignore line endings when comparing resource content; defaults to *false*, while *true* triggers a binary comparison.	No

resourcecontains

Tests whether a resource contains a given (sub)string.

The resources to check are specified via references or - in the case of file resources via the resource attribute. **Since Ant 1.7.1**

Attribute	Description	Required
resource	Name of a file that is the resource to test.	One of the two
refid	Reference to a resource defined inside the project.	
substring	The string to search for.	Yes
casesensitive	Perform a case sensitive comparision. Default is true.	No

hasmethod

Tests for a class having a method or field. If the class is not found or fails to load, the build fails. **Since Ant 1.7**

Attribute	Description	Required
classname	name of the class to load	yes
field	name of a field to look for	one of field or method
method	name of a method to look for	one of field or method
ignoreSystemClasses	should system classes be ignored?	No - default is false
classpath	a class path	No
classpathref	reference to a class path	No

There is also a nested <classpath> element, which can be used to specify a classpath.

```
<hasmethod classname="java.util.ArrayList" method="trimToSize"  />
```

Looks for the method trimToSize in the ArrayList class.

matches

Test if the specified string matches the specified regular expression pattern. **Since Ant 1.7**

Attribute	Description	Required
string	The string to test.	Yes
pattern	The regular expression pattern used to test.	Yes, unless there is a nested `<regexp>` element.
casesensitive	Perform a case sensitive match. Default is true.	No
multiline	Perform a multi line match. Default is false.	No
singleline	This allows '.' to match new lines. SingleLine is not to be confused with multiline, SingleLine is a perl regex term, it corresponds to dotall in java regex. Default is false.	No

There is also an optional <regexp> element, which can be used to specify a regular expression instead of the "pattern" attribute. See Regexp Type for the description of the nested element regexp and of the choice of regular expression implementation.

An example:

```
<condition property="legal-password">
  <matches pattern="[1-9]" string="${user-input}"/>
</condition>
<fail message="Your password should at least contain one number"
      unless="legal-password"/>
```

The following example sets the property "ok" if the property "input" is three characters long, starting with 'a' and ending with 'b'.

```
<condition property="ok">
  <matches string="${input}" pattern="^a.b$"/>
</condition>
```

The following defines a reference regular expression for matching dates and then uses antunit to check if the property "today" is in the correct format:

```
<regexp id="date.pattern" pattern="^[0123]\d-[01]\d-[12]\d\d\d$"/>

<au:assertTrue xmlns:au="antlib:org.apache.ant.antunit">
  <matches string="${today}">
    <regexp refid="date.pattern"/>
  </matches>
</au:assertTrue>
```

The following example shows the use of the singleline and the casesensitive flags.

```
<au:assertTrue>
  <matches string="AB${line.separator}C" pattern="^ab.*C$"
           casesensitive="false"
           singleline="true"/>
</au:assertTrue>
<au:assertFalse>
  <matches string="AB${line.separator}C" pattern="^ab.*C$"
           casesensitive="false"
           singleline="false"/>
</au:assertFalse>
```

antversion

This condition is identical to the Antversion task, all attributes are supported, the property attribute is redundant and will be ignored.

hasfreespace

Tests a partition to see if there is enough space. **Since Ant 1.7.0**

Needed attribute can be specified using standard computing terms:

- K : Kilobytes (1024 bytes)
- M : Megabytes (1024 K)
- G : Gigabytes (1024 M)
- T : Terabytes (1024 G)
- P : Petabytes (1024 T)

Attribute	Description	Required
partition	The partition or filesystem to check for freespace	Yes
needed	The amount of freespace needed.	Yes

An example:

```
<hasfreespace partition="c:" needed="100M"/>
```

islastmodified

Tests the last modified date of a resource. *Since Ant 1.8.0*

Attribute	Description	Required
millis	Specifies the expected modification time of the resource in milliseconds since midnight Jan 1 1970.	Exactly one of the two.
datetime	Specifies the expected modification time of the resource. The special value "now" indicates the current time.	
pattern	SimpleDateFormat-compatible pattern string. Defaults to MM/DD/YYYY HH:MM AM_or_PM or MM/DD/YYYY HH:MM:SS AM_or_PM.	No
mode	How to compare the timestamp. Accepted values are "equals", "before", "not-before", "after" and "not-after".	No, defaults to "equals".

The actual resource to test is specified as a nested element.

An example:

```
<islastmodified dateTime="08/18/2009 04:41:19 AM" mode="not-before">
  <file file="${file}"/>
</islastmodified>
```

resourceexists

Tests a resource for existance. *since Ant 1.8.0*

The actual resource to test is specified as a nested element.

An example:

```
<resourceexists>
  <file file="${file}"/>
</resourceexists>
```

componentdef

Description

Adds a component definition to the current project. A compenent definition is the same as a typedef except:

1. that it can only be used in other types or tasks that accept components (by having an *add()* method).
2. multiple components may have the same name, provided they implement different interfaces.

The purpose of this is to allow internal Ant definitions to be made for tags like "and" or "or".

Examples

```
<componentdef name="or" onerror="ignore"
   classname="com.apache.tools.ant.taskdefs.conditions.Or"/>
 <componentdef name="or" onerror="ignore"
   classname="com.apache.tools.ant.types.resources.selectors.Or"/>
```

defines two components with the same name "or"; one is a condition (see conditions) and one is a selector (see selectors).

CvsVersion

Description

This task allows to retrieve a CVS client and server version. *Since Ant 1.6.1.*

Parameters

Attribute	Description	Required
Attributes from parent Cvs task which are meaningful here		
cvsRoot	the `CVSROOT` variable.	No
cvsRsh	the `CVS_RSH` variable.	No
dest	directory containing the checked out version of the project	No, default is project's basedir.
package	the package/module to check out.	No
port	Port used by CVS to communicate with the server.	No, default port 2401.
passfile	Password file to read passwords from.	No, default file ~/.cvspass.
failonerror	Stop the build process if the command exits with a return code other than `0`. Defaults to false	No
Specific attributes		
clientversionproperty	Name of a property where the cvsclient version should be stored	No
serverversionproperty	Name of a property where the cvs server version should be stored	No

Examples

```
<cvsversion cvsRoot=":pserver:anoncvs@cvs.apache.org:/home/cvspublic"
     passfile="/home/myself/.cvspass"
     serverversionproperty="apachecvsversion"
     clientversionproperty="localcvsversion"
/>
```

finds out the cvs client and server versions and stores the versions in the properties called apachecvsversion and localcvsversion

Diagnostics

Diagnostics

Runs Ant's `-diagnostics` code inside Ant itself. This is good for debugging Ant's configuration under an IDE. **Since Ant 1.7.0**

Examples

```
    <target name="diagnostics" description="diagnostics">
        <diagnostics/>
    </target>
```

Prints out the current diagnostics dump.

EchoXML

Description

Echo nested XML to the console or a file. **Since Ant 1.7**

Parameters

Attribute	Description	Required
file	The file to receive the XML. If omitted nested XML will be echoed to the log.	No
append	Whether to append `file`, if specified.	No
namespacePolicy	Sets the namespace policy as defined by `org.apache.tools.ant.util.DOMElementWriter.XmlNamespacePolicy`. Valid values are "ignore," "elementsOnly," or "all." Default is "ignore".	No

Parameters specified as nested elements

Nested XML content is required.

Examples

```
<echoxml file="subbuild.xml">
  <project default="foo">
    <target name="foo">
      <echo>foo</echo>
    </target>
  </project>
</echoxml>
```

Creates an Ant buildfile, `subbuild.xml`.

Length

Description

Display or set a property containing length information for a string, a file, or one or more nested Resource Collections. Can also be used as a condition. **Since Ant 1.6.3**

Parameters

Attribute	Description	Required
property	The property to set. If omitted the results are written to the log. Ignored when processing as a condition.	No
file	Single file whose length to report.	One of these, or one or more nested filesets
resource	Single resource whose length to report (using extended properties handling). *Since Ant 1.8.1*	
string	The string whose length to report.	
mode	File length mode; when "all" the resulting value is the sum of all included resources' lengths; when "each" the task outputs the absolute path and length of each included resource, one per line. Ignored when processing as a condition.	No; default is "all"
trim	Whether to trim when operating on a string. Default *false*.	No; only valid when string is set
length	Comparison length for processing as a condition.	Yes, in condition mode
when	Comparison type: "equal", "eq", "greater", "gt", "less", "lt", "ge" (greater or equal), "ne" (not equal), "le" (less or equal) for use when operating as a condition.	No; default is "equal"

Parameters specified as nested elements

Resource Collections

You can include resources via nested Resource Collections.

Examples

```
<length string="foo" property="length.foo" />
```

Stores the length of the string "foo" in the property named *length.foo*.

```
<length file="bar" property="length.bar" />
```

Stores the length of file "bar" in the property named *length.bar*.

```
<length property="length" mode="each">
    <fileset dir="." includes="foo,bar"/>
</length>
```

Writes the file paths of *foo* and *bar* and their length into the property *length*.

```
<length property="length" mode="all">
    <fileset dir="." includes="foo,bar"/>
</length>
```

Adds the length of *foo* and *bar* and stores the result in property *length*.

LoadResource

Since Ant 1.7

Description

Load a text resource into a single property. Unless an encoding is specified, the encoding of the current locale is used. Resources to load are specified as nested resource elements or single element resource collections. If the resource content is empty (maybe after processing a filterchain) the property is not set.

Since properties are immutable, the task will not change the value of an existing property.

Parameters

Attribute	Description	Required
property	property to save to	Yes
encoding	encoding to use when loading the resource	No
failonerror	Whether to halt the build on failure	No, default "true"
quiet	Do not display a diagnostic message (unless Ant has been invoked with the `-verbose` or `-debug` switches) or modify the exit status to reflect an error. Setting this to "true" implies setting failonerror to "false".	No, default "false"

The LoadResource task supports nested FilterChains.

Examples

```
<loadresource property="homepage">
  <url url="http://ant.apache.org/index.html"/>
</loadresource>
```

Load the entry point of Ant's homepage into property "homepage"; an `<echo>` can print this.

For more examples see the loadfile task.

Manifestclasspath

Description

Converts a Path into a property whose value is appropriate for a Manifest's `Class-Path` attribute.

This task is often used to work around command line limitations on Windows when using very long class paths when launching an application. The long class path normally specified on the command line is replaced by a single (possibly empty) jar file which an in-manifest Class-Path attribute whose value lists all the jar and zip files the class path should contain. The files referenced from this attribute must be found relatively to the jar file itself, usually in the same directory. The Java VM automically uses all file entries listed in the Class-Path attributes of a jar to locate/load classes. Note though that it silently ignores entries for which it cannot find any corresponding file.

Note that the property value created may be longer than a manifest's maximum 72 characters per line, but will be properly wrapped as per the Jar specification by the `<manifest>` element, where the defined property is re-referenced.

since Ant 1.7

Parameters

Attribute	Description	Required
property	the name of the property to set. This property must not already be set.	Yes
jarfile	the filename for the Jar which will contain the manifest that will use the property this task will set. This file need not exist yet, but its parent directory must exist.	Yes
maxParentLevels	The maximum number of parent directories one is allowed to traverse to navigate from the jar file to the path entry. Put differently, the maximum number of .. which is allowed in the relative path from the jar file to a given class path enty. Specify 0 to enforce a path entry to be in the same directory (or one of its sub-directories) as the jar file itself. Defaults to 2 levels.	No

Parameters specified as nested elements

classpath

A Path-like element, which can be defined in-place, or refer to a path defined elsewhere using the `<classpath refid="pathid" />` syntax. This classpath must not be empty, and is required.

Examples

```
<manifestclasspath property="jar.classpath"
                   jarfile="build/acme.jar">
  <classpath refid="classpath" />
</manifestclasspath>
```

Assuming a path of id "classpath" was already defined, convert this path relatively to the build/ directory that will contain acme.jar, which can later be created with `<jar>` with a nested `<manifest>` element that lists an `<attribute name="Class-Path" value="${jar.classpath}" />`.

Nice

Description

Provide "nice-ness" to the current thread and/or query the current value.

Parameters

Attribute	Description	Required
currentpriority	the name of the property whose value should be set to the current "nice-ness" level.	No
newpriority	the value to which the "nice-ness" level should be set. Must be a valid Java Thread priority.	No

Examples

```
<nice newpriority="10"/>
```

Set the Thread priority to 10 (highest).

```
<nice currentpriority="priority"/>
```

Store the current Thread priority in the user property "priority".

```
<nice currentpriority="currentpriority" newpriority="1"/>
```

Set the current Thread priority to 1 (lowest), storing the original priority in the user property "currentpriority". This can be used to set the priority back to its original value later.

ResourceCount

Description

Display or set a property containing the size of a nested Resource Collection. Can also be used as a condition. **Since Ant 1.7**

Parameters

Attribute	Description	Required
property	The property to set. If omitted the results are written to the log. Ignored when processing as a condition.	No
refid	A reference to a Resource Collection.	Yes, unless a nested Resource Collection is supplied
count	Comparison count for processing as a condition.	Yes, in condition mode
when	Comparison type: "equal", "eq", "greater", "gt", "less", "lt", "ge" (greater or equal), "ne" (not equal), "le" (less or equal) for use when operating as a condition.	No; default is "equal"

Parameters specified as nested elements

Resource Collection

A single Resource Collection should be specified via a nested element or the `refid` attribute.

Examples

```
<resourcecount property="count.foo">
  <filelist dir="." files="foo,bar" />
</resourcecount>
```

Stores the number of resources in the specified filelist (two) in the property named *count.foo*.

```
<project>
  <property name="file" value="${ant.file}"/>
  <resourcecount property="file.lines">
    <tokens>
      <concat>
        <filterchain>
          <tokenfilter>
            <linetokenizer/>
          </tokenfilter>
        </filterchain>
        <fileset file="${file}"/>
      </concat>
    </tokens>
  </resourcecount>
  <echo>The file '${file}' has ${file.lines} lines.</echo>
</project>
```

Stores the number of lines of the current buildfile in the property `file.lines`. Requires Ant 1.7.1+ as <concat> has to be resource.

Retry

Description

Retry is a container which executes a single nested task until either: there is no failure; or: its *retrycount* has been exceeded. If this happens a BuildException is thrown. *Since Ant 1.7.1*

Parameters

Attribute	Description	Required
retrycount	number of times to attempt to execute the nested task	Yes

Any valid Ant task may be embedded within the retry task.

Example

```
<retry retrycount="3">
  <get src="http://www.unreliable-server.com/unreliable.tar.gz"
       dest="/home/retry/unreliable.tar.gz" />
</retry>
```

This example shows how to use `<retry>` to wrap a task which must interact with an unreliable network resource.

Truncate

Description

Set the length of one or more files, as the intermittently available `truncate` Unix utility/function. In addition to working with a single file, this Task can also work on resources and resource collections. **Since Ant 1.7.1**.

Parameters

Attribute	Description	Required
file	The name of the file.	Unless a nested resource collection element has been specified.
length	Specifies the new file length (in bytes) to set. The following suffixes are supported: • K : Kilobytes (1024 bytes) • M : Megabytes (1024 K) • G : Gigabytes (1024 M) • T : Terabytes (1024 G) • P : Petabytes (1024 T)	At most one of these. Omitting both implies `length="0"`.
adjust	Specifies the number of bytes (and positive/negative direction) by which to adjust file lengths. The same suffixes are supported for this attribute as for the `length` attribute.	
create	Whether to create nonexistent files.	No, default *true*.
mkdirs	Whether to create nonexistent parent directories when creating new files.	No, default *false*.

Parameters specified as nested elements

any resource collection

You can use any number of nested resource collection elements to define the resources for this task and refer to resources defined elsewhere. **Note:** resources passed to this task are expected to be filesystem-based.

Examples

```
<truncate file="foo" />
```

Sets the length of file `foo` to zero.

```
<truncate file="foo" length="1K" />
```

Sets the length of file `foo` to 1 kilobyte (1024 bytes).

```
<truncate file="foo" adjust="1K" />
```

Adjusts the length of file `foo` upward by 1 kilobyte.

```
  <truncate file="foo" adjust="-1M" />
```

Adjusts the length of file `foo` downward by 1 megabyte.

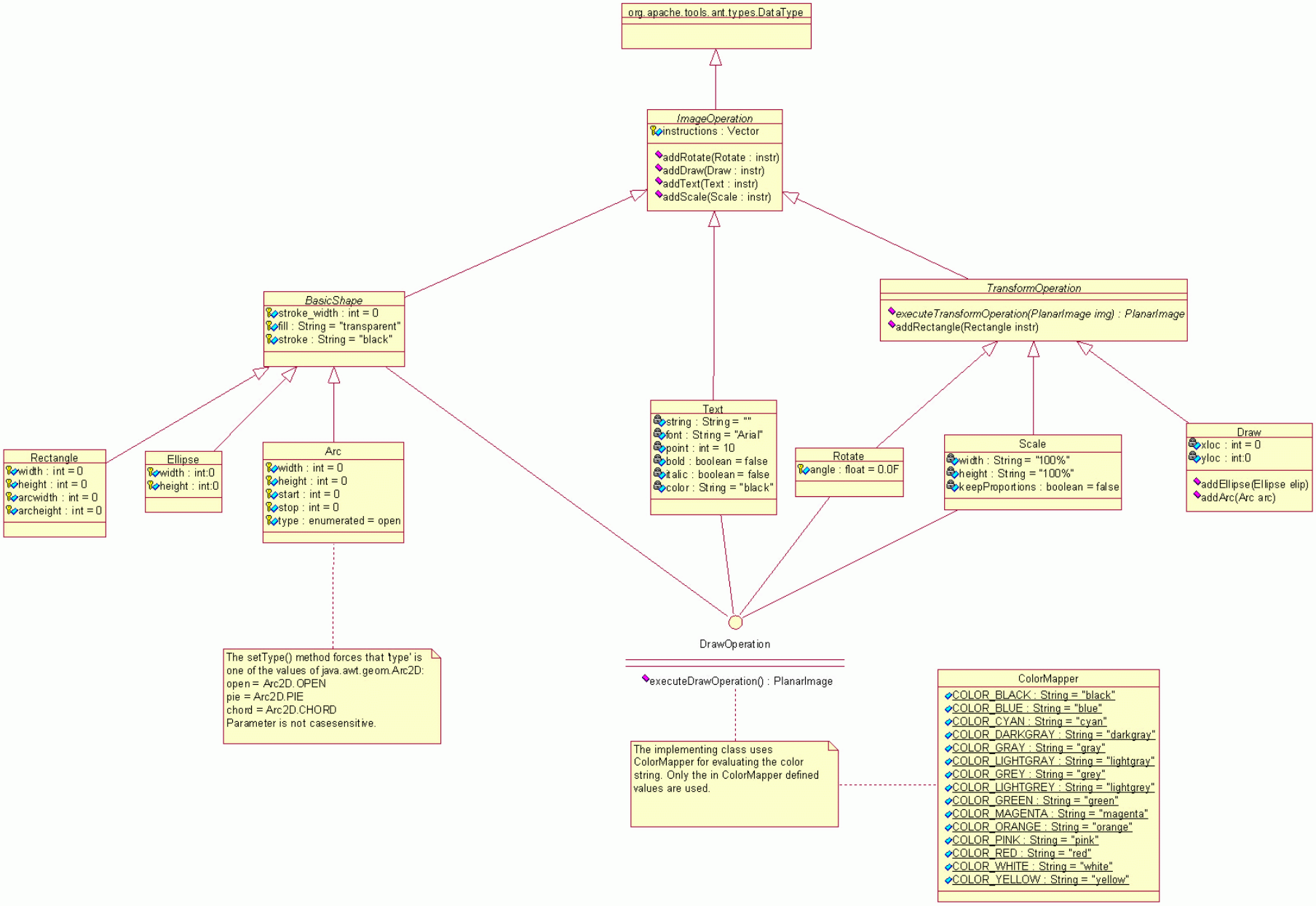
Image

Description

Applies a chain of image operations on a set of files.

Requires Java Advanced Image API from Sun.

Overview of used datatypes


org.apache.tools.ant.types.DataType
ImageOperation
instructions : Vector
addRotate(Rotate : instr)
addDraw(Draw : instr)
addText(Text : instr)
addScale(Scale : instr)
BasicShape
stroke_width : int = 0
fill : String = "transparent"
stroke : String = "black"
TransformOperation
executeTransformOperation(PlanarImage img) : PlanarImage
addRectangle(Rectangle instr)
Rectangle
width : int = 0
height : int = 0
arcwidth : int = 0
archeight : int = 0
Ellipse
width : int:0
height : int:0
Arc
width : int = 0
height : int = 0
start : int = 0
stop : int = 0
type : enumerated = open
Text
string : String = ""
font : String = "Arial"
point : int = 10
bold : boolean = false
italic : boolean = false
color : String = "black"
Rotate
angle : float = 0.0F
Scale
width : String = "100%"
height : String = "100%"
keepProportions : boolean = false
Draw
xloc : int = 0
yloc : int:0
addEllipse(Ellipse elip)
addArc(Arc arc)
The setType() method forces that 'type' is one of the values of java.awt.geom.Arc2D:
open = Arc2D.OPEN
pie = Arc2D.PIE
chord = Arc2D.CHORD
Parameter is not casesensitive.
DrawOperation
executeDrawOperation() : PlanarImage
The implementing class uses ColorMapper for evaluating the color string. Only the in ColorMapper defined values are used.
ColorMapper
COLOR_BLACK : String = "black"
COLOR_BLUE : String = "blue"
COLOR_CYAN : String = "cyan"
COLOR_DARKGRAY : String = "darkgray"
COLOR_GRAY : String = "gray"
COLOR_LIGHTGRAY : String = "lightgray"
COLOR_GREY : String = "grey"
COLOR_LIGHTGREY : String = "lightgrey"
COLOR_GREEN : String = "green"
COLOR_MAGENTA : String = "magenta"
COLOR_ORANGE : String = "orange"
COLOR_PINK : String = "pink"
COLOR_RED : String = "red"
COLOR_WHITE : String = "white"
COLOR_YELLOW : String = "yellow"

Parameters

Attribute	Description	Required
failonerror	Boolean value. If false, note errors to the output but keep going.	no (defaults to *true*)
srcdir	Directory containing the images.	yes, unless nested fileset is used
encoding	Image encoding type. Valid (caseinsensitive) are: jpg, jpeg, tif, tiff	no (defaults to *JPEG*)
overwrite	Boolean value. Sets whether or not to overwrite a file if there is naming conflict.	no (defaults to *false*)
gc	Boolean value. Enables garbage collection after each image processed.	no (defaults to *false*)
destdir	Directory where the result images are stored.	no (defaults to value of *srcdir*)
includes	comma- or space-separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma- or space-separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
caseSensitive	Boolean value. Sets case sensitivity of the file system.	no (defaults to *false*)
followSymlinks	Boolean value. Sets whether or not symbolic links should be followed.	no (defaults to *true*)

Parameters specified as nested elements

This task forms an implicit FileSet and supports most attributes of `<fileset>` as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

ImageOperation

Adds an ImageOperation to chain.

Nested Elements

ImageOperation can handle nested Rotate, Draw, Rectangle, Text and Scale objects.

Rotate

Adds a Rotate ImageOperation to chain.

Parameters

Attribute	Description	Required
angle	Float value. Sets the angle of rotation in degrees.	no (defaults to *0.0F*)

Scale

Adds a Scale ImageOperation to chain.

Parameters

Attribute	Description	Required
proportions	Sets which dimension to control proportions from. Valid values are: • "ignore" - treat the dimensions independently. • "height" - keep proportions based on the width. • "width" - keep proportions based on the height. • "cover" - keep proportions and fit in the supplied dimensions. • "fit" - keep proportions and cover the supplied dimensions.	no (defaults to *ignore*)
width	Sets the width of the image, either as an integer or a %.	no (defaults to *100%*)
height	Sets the height of the image, either as an integer or a %.	no (defaults to *100%*)

Draw

Adds a Draw ImageOperation to chain. DrawOperation DataType objects can be nested inside the Draw object.

Parameters

Attribute	Description	Required
xloc	X-Position where to draw nested image elements.	no (defaults to *0*)
yloc	Y-Position where to draw nested image elements.	no (defaults to *0*)

mapper

Since Ant 1.8.0

You can define filename transformations by using a nested mapper element. The default mapper used by `<image>` is the identity mapper.

You can also use a filenamemapper type in place of the mapper element.

Examples

```
<image destdir="samples/low" overwrite="yes">
    <fileset dir="samples/full">
        <include name="**/*.jpg"/>
    </fileset>
    <scale width="160" height="160" proportions="fit"/>
</image>
```

Create thumbnails of my images and make sure they all fit within the 160x160 size whether the image is portrait or landscape.

```
<image srcdir="src" includes="*.png">
    <scale proportions="width" width="40"/>
</image>
```

Creates a thumbnail for all PNG-files in *src* in the size of 40 pixel keeping the proportions and stores the *src*.

```
<image srcdir="src" destdir="dest" includes="*.png">
    <scale proportions="width" width="40"/>
</image>
```

Same as above but stores the result in *dest*.

```
<image srcdir="src" destdir="dest" includes="*.png">
    <scale proportions="width" width="40"/>
    <globmapper from="*" to="scaled-*"/>
</image>
```

Same as above but stores the resulting file names will be prefixed by "scaled-".

SchemaValidate

Description

This task validates XML files described by an XML Schema. The task extends the XmlValidate task with XSD-specific features.

1. The parser is created validating and namespace aware
2. Validation is turned on.
3. Schema validation is turned on.
4. Any no-namespace schema URL or file supplied is used as the no-namespace schema
5. All nested schema declarations are turned into the list of namespace-url bindings for schema lookup.

Note that nested catalogs are still used for lookup of the URLs given as the sources of schema documents, so you can still delegate lookup to a catalog, you just need to list all schema URIs and their URL equivalents.

This task supports the use of nested

- `<xmlcatalog>` elements
- `<schema>` elements, that bind a namespace URI to a URL or a local filename.
- `<dtd>` elements which are used to resolve DTDs and entities.
- `<attribute>` elements which are used to set features on the parser. These can be any number of `http://xml.org/sax/features/` or other features that your parser may support.
- `<property>` elements, containing string properties

The task only supports SAX2 or later parsers: it is an error to specify a SAX1 parser.

Parameters

Attribute	Description	Required
classname	the parser to use.	No
classpathref	where to find the parser class. Optionally can use an embedded `<classpath>` element.	No
disableDTD	Flag to disable DTD support. DTD support is needed to validate XSD files themselves, amongst others.	No - default false
failonerror	fails on a error if set to true (defaults to true).	No
file	the file(s) you want to check. (optionally can use an embedded fileset)	No
fullchecking	enable full schema checking. Slow but strict.	No - default true
lenient	if true, only check the XML document is well formed	No
noNamespaceFile	filename of a no-namespace XSD file to provide the schema for no-namespace XML content.	No
noNamespaceURL	URL of a no-namespace XSD file to provide the schema for no-namespace XML content.	No
warn	log parser warn events.	No

Nested Elements

schema

Identify the name and location of a schema that may be used in validating the document(s).

Attribute	Description	Required
namespace	URI of the schema namespace	Yes
url	URL of the schema	One of url or file is required
file	file of the schema	One of url or file is required

dtd

`<dtd>` is used to specify different locations for DTD resolution.

Attribute	Description	Required
publicId	Public ID of the DTD to resolve	Yes
location	Location of the DTD to use, which can be a file, a resource, or a URL	Yes

xmlcatalog

The `<xmlcatalog>` element is used to perform entity resolution.

attribute

The `<attribute>` element is used to set parser features.
Features usable with the xerces parser are defined here : Setting features
SAX features are defined here: `http://xml.org/sax/features/`

Attribute	Description	Required
name	The name of the feature	Yes
value	The boolean value of the feature	Yes

property

The `<property>` element is used to set properties. These properties are defined here for the xerces XML parser implementation : XML Parser properties Properties can be used to set the schema used to validate the XML file.

Attribute	Description	Required
name	The name of the feature	Yes
value	The string value of the property	Yes

Examples

```
    <schemavalidate
```

```
      noNamespaceFile="document.xsd"
      file="xml/endpiece.xml">
    </schemavalidate>
```

Validate a document against an XML schema. The document does not declare any schema itself, which is why the `noNamespaceFile` is needed.

```
    <presetdef name="validate-soap">
      <schemavalidate >
        <schema namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
          file="${soap.dir}/ws-addressing.xsd" />
        <schema namespace="http://www.w3.org/2003/05/soap-envelope"
          file="${soap.dir}/soap12.xsd" />
        <schema namespace="http://schemas.xmlsoap.org/wsdl/"
          file="${soap.dir}/wsdl.xsd" />
        <schema namespace="http://www.w3.org/2001/XMLSchema"
          file="${soap.dir}/XMLSchema.xsd" />
        </schemavalidate>
    </presetdef>
```

Declare a new preset task, `<validate-soap>`, that validates XSD and WSDL documents against the relevant specifications. To validate XSD documents, you also need XMLSchema.dtd and datatypes.dtd in the same directory as XMLSchema.xsd, or pointed to via the catalog. All these files can be fetched from the W3C.

```
    <validate-soap file="xml/test.xsd"/>
```

Use the preset task defined above to validate an XML Schema document.

Scriptdef

Description

Scriptdef can be used to define an Ant task using a scripting language. Ant scripting languages supported by Apache BSF or JSR 223 may be used to define the script. Scriptdef provides a mechanism to encapsulate control logic from a build within an Ant task minimizing the need for providing control style tasks in Ant itself. Complex logic can be made available while retaining the simple structure of an Ant build file. Scriptdef is also useful for prototyping new custom tasks. Certainly as the complexity of the script increases it would be better to migrate the task definition into a Java based custom task.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information.

The attributes and nested elements supported by the task may be defined using `<attribute>` and `<element>` nested elements. These are available to the script that implements the task as two collection style script variables `attributes` and `elements`. The elements in the `attributes` collection may be accessed by the attribute name. The `elements` collection is accessed by the nested element name. This will return a list of all instances of the nested element. The instances in this list may be accessed by an integer index.

Note: Ant will turn all attribute and element names into all lowercase names, so even if you use name="SomeAttribute", you'll have to use "someattribute" to retrieve the attribute's value from the `attributes` collection.

The name "self" (*since Ant 1.6.3*) is a pre-defined reference to the script def task instance. It can be used for logging, or for integration with the rest of ant. the `self.text attribute` contains any nested text passed to the script

If an attribute or element is not passed in, then `attributes.get()` or `elements.get()` will return null. It is up to the script to perform any checks and validation. `self.fail(String message)`can be used to raise a `BuildException`.

The name "project" is a pre-defined reference to the Ant Project. For more information on writing scripts, please refer to the `<script>` task

Parameters

Attribute	Description	Required
name	the name of the task to be created using the script	Yes
language	The programming language the script is written in. Must be a supported Apache BSF or JSR 223 language	Yes
manager	The script engine manager to use. See the script task for using this attribute.	No - default is "auto"
src	The location of the script as a file, if not inline	No
uri	The XML namespace uri that this definition should live in.	No
classpath	The classpath to pass into the script.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No
loaderRef	the name of the loader that is used to load the script, constructed from the specified classpath. This allows multiple script defintions to reuse the same class loader.	No

Nested elements

attribute

Attribute	Description	Required
name	the name of the attribute	Yes

element

Attribute	Description	Required
name	the name of the nested element to be supported by the task defined by the script	Yes
classname	the classname of the class to be used for the nested element. This specifies the class directly and is an alternative to specifying the Ant type name.	No
type	This is the name of an Ant task or type which is to be used when this element is to be created. This is an alternative to specifying the class name directly. If the type is in a namespace, the URI and a : must be prefixed to the type. For example `type="antlib:example.org:newtype"`	No
any resource or resource collection	Since Ant1.7.1, this task can load scripts from any resource supplied as a nested element. when	No

classpath

See the script task for using this nested element.

Examples

The following definition creates a task which supports an attribute called attr and two nested elements, one being a fileset and the other a path. When executed, the resulting task logs the value of the attribute and the basedir of the first fileset.

```
  <scriptdef name="scripttest" language="javascript">
    <attribute name="attr1"/>
    <element name="fileset" type="fileset"/>
    <element name="path" type="path"/>
    <![CDATA[

      self.log("Hello from script");
      self.log("Attribute attr1 = " + attributes.get("attr1"));
      self.log("First fileset basedir = "
        + elements.get("fileset").get(0).getDir(project));

    ]]>
  </scriptdef>

  <scripttest attr1="test">
    <path>
      <pathelement location="src"/>
    </path>
    <fileset dir="src"/>
    <fileset dir="main"/>
  </scripttest>
```

The following variation on the above script lists the number of fileset elements and iterates through them

```
  <scriptdef name="scripttest2" language="javascript">
    <element name="fileset" type="fileset"/>
    <![CDATA[
      filesets = elements.get("fileset");
      self.log("Number of filesets = " + filesets.size());
      for (i = 0; i < filesets.size(); ++i) {
        self.log("fileset " + i + " basedir = "
          + filesets.get(i).getDir(project));
      }
    ]]>
  </scriptdef>

  <scripttest2>
    <fileset dir="src"/>
    <fileset dir="main"/>
  </scripttest2>
```

When a script has a syntax error, the scriptdef name will be listed in the error. For example in the above script, removing the closing curly bracket would result in this error

```
build.xml:15: SyntaxError: missing } in compound statement (scriptdef <scripttest2>; line 10)
```

Script errors are only detected when a script task is actually executed.

The next example does uses nested text in Jython. It also declares the script in a new xml namespace, which must be used to refer to the task. Declaring scripts in a new namespace guarantees that Ant will not create a task of the same (namespace,localname) name pair.

```
<target name="echo-task-jython">
  <scriptdef language="jython"
      name="echo"
      uri="http://example.org/script">
      <![CDATA[
self.log("text: " +self.text)
    ]]>
    </scriptdef>
</target>

<target name="testEcho" depends="echo-task-jython"
    xmlns:s="http://example.org/script">
  <s:echo>nested text</s:echo>
</target>
```

The next example shows the use of <classpath> and "loaderref" to get access to the beanshell jar.

```
    <scriptdef name="b1" language="beanshell"
               loaderref="beanshell-ref">
      <attribute name="a"/>
      <classpath
        path="${user.home}/scripting/beanshell/bsh-1.3b1.jar"/>
      self.log("attribute a is " + attributes.get("a"));
    </scriptdef>

    <scriptdef name="b2" language="beanshell"
               loaderref="beanshell-ref">
      <attribute name="a2"/>
      self.log("attribute a2 is " + attributes.get("a2"));
    </scriptdef>

    <b1 a="this is an 'a'"/>
    <b2 a2="this is an 'a2' for b2"/>
```

Testing Scripts

The easiest way to test scripts is to use the AntUnit ant library. This will run all targets in a script that begin with "test" (and their dependencies).

SSHSESSION

Description

since Ant 1.8.0

A Task which establishes an SSH connection with a remote machine running SSH daemon, optionally establishes any number of local or remote tunnels over that connection, then executes any nested tasks before taking down the connection.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies for more information. This task has been tested with jsch-0.1.33 and above and won't work with versions of jsch earlier than 0.1.28.

See also the sshexec and scp tasks

Parameters

Attribute	Description	Required
host	The hostname or IP address of the remote host to which you wish to connect.	Yes
username	The username on the remote host to which you are connecting.	Yes
port	The port to connect to on the remote host.	No, defaults to 22.
localtunnels	A comma-delimited list of colon-delimited `lport:rhost:rport` triplets defining local port forwarding. If nested localtunnel elements are also provided, both sets of tunnels will be established.	No
remotetunnels	A comma-delimited list of colon-delimited `rport:lhost:lport` triplets defining remote port forwarding. If nested remotetunnel elements are also provided, both sets of tunnels will be established.	No
trust	This trusts all unknown hosts if set to yes/true. **Note** If you set this to false (the default), the host you connect to must be listed in your knownhosts file, this also implies that the file exists.	No, defaults to No.
knownhosts	This sets the known hosts file to use to validate the identity of the remote host. This must be a SSH2 format file. SSH1 format is not supported.	No, defaults to ${user.home}/.ssh/known_hosts.
failonerror	Whether to halt the build if the command does not complete successfully.	No; defaults to true.
password	The password.	Not if you are using key based authentication or the password has been given in the file or todir attribute.
keyfile	Location of the file holding the private key.	Yes, if you are using key based

		authentication.
passphrase	Passphrase for your private key.	No, defaults to an empty string.
timeout	Give up if the connection cannot be established within the specified time (given in milliseconds). Defaults to 0 which means "wait forever".	No

Parameters specified as nested elements

localtunnel

Optionally, any number of localtunnel elements can be used to define local port forwarding over the SSH connection. If the localtunnels parameter was also specified, both sets of tunnels will be established.

Attribute	Description	Required
lport	The number of the local port to be forwarded.	Yes
rhost	The hostname or IP address of the remote host to which the local port should be forwarded.	Yes
rport	The number of the port on the remote host to which the local port should be forwarded.	Yes

remotetunnel

Optionally, any number of remotetunnel elements can be used to define remote port forwarding over the SSH connection. If the remotetunnels parameter was also specified, both sets of tunnels will be established.

Attribute	Description	Required
rport	The number of the remote port to be forwarded.	Yes
lhost	The hostname or IP address of the local host to which the remote port should be forwarded.	Yes
lport	The number of the port on the local host to which the remote port should be forwarded.	Yes

sequential

The sequential element is a required parameter. It is a container for nested Tasks which are to be executed once the SSH connection is established and all local and/or remote tunnels established.

Examples

Connect to a remote machine using password authentication, forward the local cvs port to the remote host, and execute a cvs command locally, which can use the tunnel.

```
<sshsession host="somehost"
  username="dude"
  password="yo"
  localtunnels="2401:localhost:2401"
>
  <sequential>
    <cvs  command="update ${cvs.parms} ${module}"
      cvsRoot="${cvs.root}"
      dest="${local.root}"
      failonerror="true"
    />
  </sequential>
</sshsession>
```

Do the same thing using nested localtunnel element.

```
<sshsession host="somehost"
  username="dude"
  password="yo"
>
  <localtunnel lport="2401" rhost="localhost" rport="2401"/>
  <sequential>
    <cvs  command="update ${cvs.parms} ${module}"
      cvsRoot="${cvs.root}"
      dest="${local.root}"
      failonerror="true"
  />
  </sequential>
</sshsession>
```

Connect to a remote machine using key authentication, forward port 1080 to port 80 of an intranet server which is not directly accessible, then run a get task using that tunnel.

```
<sshsession host="somehost"
username="dude"
keyfile="${user.home}/.ssh/id_dsa"
passphrase="yo its a secret"/>
  <LocalTunnel lport="1080" rhost="intranet.mycomp.com" rport="80"/>
  <sequential>
    <get src="http://localhost:1080/somefile" dest="temp/somefile"/>
  </sequential>
</sshsession>
```

Security Note: Hard coding passwords or passphrases and/or usernames in sshsession task can be a serious security hole. Consider using variable substitution and include the password on the command line. For example:

```
<sshsession host="somehost"
username="${username}"
password="${password}"
localtunnels="2401:localhost:2401">
  <sequential>
    <sometask/>
  </sequential>
</sshsession>
```

Invoking ant with the following command line:

```
   ant -Dusername=me -Dpassword=mypassword target1 target2
```

Is slightly better, but the username/password is exposed to all users on an Unix system (via the ps command). The best approach is to use the `<input>` task and/or retrieve the password from a (secured) .properties file.

Symlink

Description

Manages symbolic links on Unix based platforms. Can be used to make an individual link, delete a link, create multiple links from properties files, or create properties files describing links in the specified directories. Existing links are not overwritten by default.

FileSets are used to select a set of links to record, or a set of property files to create links from.

Parameters

Attribute	Description	Required
action	The type of action to perform, may be "single", "record", "recreate" or "delete".	No, defaults to single.
link	The name of the link to be created or deleted. **Note** this attribute is resolved against the current working directory rather than the project's basedir for historical reasons. It is recommended you always use an absolute path or a path like `${basedir}/some-path` as its value.	required for action="single" or "delete". Ignored in other actions.
resource	The resource the link should point to.	required for action="single". Ignored in other actions.
linkfilename	The name of the properties file to create in each included directory.	required for action="record". Ignored in other actions.
overwrite	Overwrite existing links or not.	No; defaults to false.
failonerror	Stop build if true, log a warning message, but do not stop the build, when the an error occurs if false.	No; defaults to true.

Parameters specified as nested elements

fileset

FileSets are used when action = "record" to select directories and linknames to be recorded. They are also used when action = "recreate" to specify both the name of the property files to be processed, and the directories in which they can be found. At least one fileset is required for each case.

Examples

Make a link named "foo" to a resource named "bar.foo" in subdir:

```
<symlink link="${dir.top}/foo" resource="${dir.top}/subdir/bar.foo"/>
```

Record all links in subdir and it's descendants in files named "dir.links"

```
<symlink action="record" linkfilename="dir.links">
```

```
    <fileset dir="${dir.top}" includes="subdir/**"/>
  </symlink>
```

Recreate the links recorded in the previous example:

```
  <symlink action="recreate">
     <fileset dir="${dir.top}" includes="subdir/**/dir.links"/>
  </symlink>
```

Delete a link named "foo":

```
 <symlink action="delete" link="${dir.top}/foo"/>
```

Java 1.2 and earlier: Due to limitations on executing system level commands in Java versions earlier than 1.3 this task may have difficulty operating with a relative path in ANT_HOME. The typical symptom is an IOException where ant can't find /some/working/directory${ANT_HOME}/bin/antRun or something similar. The workaround is to change your ANT_HOME environment variable to an absolute path, which will remove the /some/working/directory portion of the above path and allow ant to find the correct commandline execution script.

LIMITATIONS: Because Java has no direct support for handling symlinks this task divines them by comparing canonical and absolute paths. On non-unix systems this may cause false positives. Furthermore, any operating system on which the command `ln -s <linkname> <resourcename>` is not a valid command on the command line will not be able to use action="single" or action="recreate". Action="record" and action=delete should still work. Finally, the lack of support for symlinks in Java means that all links are recorded as links to the **canonical** resource name. Therefore the link: `link --> subdir/dir/../foo.bar` will be recorded as `link=subdir/foo.bar` and restored as `link --> subdir/foo.bar`

Programming your own Selectors

Selector Programming API

Want to define your own selectors? It's easy!

First, pick the type of selector that you want to define. There are three types, and a recipe for each one follows. Chances are you'll want to work with the first one, Custom Selectors.

1. Custom Selectors

   This is the category that Ant provides specifically for you to define your own Selectors. Anywhere you want to use your selector you use the `<custom>` element and specify the class name of your selector within it. See the Custom Selectors section of the Selector page for details. The `<custom>` element can be used anywhere the core selectors can be used. It can be contained within Selector Containers, for example.

   To create a new Custom Selector, you have to create a class that implements `org.apache.tools.ant.types.selectors.ExtendFileSelector`. The easiest way to do that is through the convenience base class `org.apache.tools.ant.types.selectors.BaseExtendSelector`, which provides all of the methods for supporting `<param>` tags. First, override the `isSelected()` method, and optionally the `verifySettings()` method. If your custom selector requires parameters to be set, you can also override the `setParameters()` method and interpret the parameters that are passed in any way you like. Several of the core selectors demonstrate how to do that because they can also be used as custom selectors.

2. Core Selectors

   These are the selectors used by Ant itself. To implement one of these, you will have to alter some of the classes contained within Ant.

   - First, create a class that implements `org.apache.tools.ant.types.selectors.FileSelector`. You can either choose to implement all methods yourself from scratch, or you can extend `org.apache.tools.ant.types.selectors.BaseSelector` instead, a convenience class that provides reasonable default behaviour for many methods.

     There is only one method required. `public boolean isSelected(File basedir, String filename, File file)` is the real purpose of the whole exercise. It returns true or false depending on whether the given file should be selected from the list or not.

     If you are using `org.apache.tools.ant.types.selectors.BaseSelector` there are also some predefined behaviours you can take advantage of. Any time you encounter a problem when setting attributes or adding tags, you can call setError(String errmsg) and the class will know that there is a problem. Then, at the top of your `isSelected()` method call `validate()` and a BuildException will be thrown with the contents of your error message. The `validate()` method also gives you a last chance to check your settings for consistency because it calls `verifySettings()`. Override this method and call `setError()` within it if you detect any problems in how your selector is set up.

     You may also want to override `toString()`.

   - Put an `add` method for your selector in `org.apache.tools.ant.types.selectors.SelectorContainer`. This is an interface, so you will also have to add an implementation for the method in the classes which implement it, namely `org.apache.tools.ant.types.AbstractFileSet`, `org.apache.tools.ant.taskdefs.MatchingTask` and

`org.apache.tools.ant.types.selectors.BaseSelectorContainer`. Once it is in there, it will be available everywhere that core selectors are appropriate.

3. Selector Containers

   Got an idea for a new Selector Container? Creating a new one is no problem:

   - Create a new class that implements `org.apache.tools.ant.types.selectors.SelectorContainer`. This will ensure that your new Container can access any new selectors that come along. Again, there is a convenience class available for you called `org.apache.tools.ant.types.selectors.BaseSelectorContainer`.
   - Implement the `public boolean isSelected(String filename, File file)` method to do the right thing. Chances are you'll want to iterate over the selectors under you, so use `selectorElements()` to get an iterator that will do that.
   - Again, put an `add` method for your container in `org.apache.tools.ant.types.selectors.SelectorContainer` and its implementations `org.apache.tools.ant.types.AbstractFileSet` and `org.apache.tools.ant.types.selectors.BaseSelectorContainer`.

Testing Selectors

For a robust component (and selectors are (Project)Components) tests are necessary. For testing Tasks we use JUnit TestCases - more specific `org.apache.tools.ant.BuildFileTest extends junit.framework.TestCase`. Some of its features like configure the (test) project by reading its buildfile and execute targets we need for selector tests also. Therefore we use that BuildFileTest. But testing selectors requires some more work: having a set of files, instantiate and configure the selector, check the selection work and more. Because we usually extend `BaseExtendSelector` its features have to be tested also (e.g. setError()).

That's why we have a base class for doing our selector tests:
`org.apache.tools.ant.types.selectors.BaseSelectorTest`.

This class extends TestCase and therefore can included in the set of Ant's unit tests. It holds an instance of preconfigured BuildFileTest. Configuration is done by parsing the src/etc/testcases/types/selectors.xml. BaseSelectorTest then gives us helper methods for handling multiple selections.

Because the term "testcase" or "testenvironment" are so often used, this special testenvironment got a new name: *bed*. Like you initialize the test environment by calling setUp() and cleaning by calling tearDown() (*or like to make your bed before go sleeping*) you have to do that work with your *bed* by calling `makeBed()` respecitive `cleanupBed()`.

A usual test scenario is

1. make the bed
2. instantiate the selector
3. configure the selector
4. let the selector do some work
5. verify the work
6. clean the bed

For common way of instantiation you have to override the `getInstance()` simply by returning a new object of your selector. For easier "selection and verification work" BaseSelectorTest provides the method `performTests()` which iterates over all files (and directories) in the String array `filenames` and checks whether the given selector returns the expected result. If an error occurred (especially the selector does not return the expected result) the test fails and the

failing filenames are logged.

An example test would be:

```
package org.apache.tools.ant.types.selectors;

public class MySelectorTest extends BaseSelectorTest {

    public MySelectorTest(String name) {
        super(name);
    }

    public BaseSelector getInstance() {
        return new MySelector();
    }

    public void testCase1() {
        try {
            // initialize test environment 'bed'
            makeBed();

            // Configure the selector
            MySelector s = (MySelector)getSelector();
            s.addParam("key1", "value1");
            s.addParam("key2", "value2");
            s.setXX(true);
            s.setYY("a value");

            // do the tests
            performTests(s, "FTTTTTTTTTTT");  // First is not selected - rest is

        } finally {
            // cleanup the environment
            cleanupBed();
        }
    }
}
```

As an example of an error JUnit could log

```
    [junit]     FAILED
    [junit] Error for files: .;copy.filterset.filtered;tar/gz/asf-logo.gif.tar.gz
    [junit] expected:<FTTTFTTTF...> but was:<TTTTTTTTT...>
    [junit] junit.framework.ComparisonFailure: Error for files:
.;copy.filterset.filtered;tar/gz/asf-logo.gif.tar.gz
    [junit] expected:<FTTTFTTTF...> but was:<TTTTTTTTT...>
    [junit]     at junit.framework.Assert.assertEquals(Assert.java:81)
    [junit]     at
org.apache.tools.ant.types.selectors.BaseSelectorTest.performTest(BaseSelectorTest.java:194)
```

Described above the test class should provide a `getInstance()` method. But that isn't used here. The used `getSelector()` method is implemented in the base class and gives an instance of an Ant Project to the selector. This is usually done inside normal build file runs, but not inside this special environment, so this method gives the selector the ability to use its own Project object (`getProject()`), for example for logging.

Logging

During development and maybe later you sometimes need the output of information. Therefore Logging is needed. Because the selector extends BaseExtendSelector or directly BaseSelector it is an Ant `DataType` and therefore a `ProjectComponent`.
That means that you have access to the project object and its logging capability. `ProjectComponent` itself provides *log* methods which will do the access to the project instance. Logging is therefore done simply with:

```
        log( "message" );
```

or

```
        log( "message" , loglevel );
```

where the `loglevel` is one of the values

- org.apache.tools.ant.Project.MSG_ERR
- org.apache.tools.ant.Project.MSG_WARN
- org.apache.tools.ant.Project.MSG_INFO (= default)
- org.apache.tools.ant.Project.MSG_VERBOSE
- org.apache.tools.ant.Project.MSG_DEBUG

BorlandGenerateClient

by Benoit Moussaud (benoit.moussaud@criltelecom.com)

Description

The BorlandGenerateClient is a task dedicated to Borland Application Server v 4.5. It offers to generate the client jar file corresponding to an ejb jar file.

Parameters

Attribute	Description	Required
ejbjar	ejb jar file	yes
debug	If true, turn on the debug mode for each borland tools (java2iiop, iastool ...) default = false	no
clientjar	client jar file name. If missing the client jar file name is build using the ejbjar file name: ejbjar = hellobean-ejb.jar => hellobean-ejbclient.jar	no
mode	choose the command launching mode. Two values: java or fork. default = fork. java is not supported for version=5.Possibility to specify a classpath.	no
version	set the Borland Application Version. • 4 means B.A.S (Borland Application Server 4.x) • 5 means B.E.S (Borland Application Server 5.x)	No, defaults to 4

Examples

The following build.xml snippet is an example of how to use Borland element into the ejbjar task using the java mode.

```
<blgenclient ejbjar="lib/secutest-ejb.jar" clientjar="lib/client.jar" debug="true" mode="fork">
version="5">
    <classpath>
        <pathelement location="mymodule.jar"/>
    </classpath>
</blgenclient>
```

BorlandDeployTool

by Benoit Moussaud (benoit.moussaud@criltelecom.com)

Description

The BorlandDeployTool is a vendor specific nested element for the Ejbjar optional task.

BorlandDeploymentTool is dedicated to the Borland Application Server 4.5.x and Borland Enterprise Server 5.x. It generates and compiles the stubs and skeletons for all ejb described into the Deployment Descriptor, builds the jar file including the support files and verify whether the produced jar is valid or not.

Benoit Moussaud maintains a separate FAQ for this task at his homepage.

Borland element

Attribute	Description	Required
destdir	The base directory into which the generated borland ready jar files are deposited	yes
debug	If true, turn on the debug mode for each borland tools (java2iiop, iastool ...) default = false	no
verify	If true, turn on the verification at the end of the jar production (default = false)	no
verifyargs	extra parameter for verify command	no
suffix	String value appended to the basename of the deployment descriptor to create the filename of the Borland EJB jar file.	No, defaults to '-ejb.jar'.
basdtd	**Deprecated**. Defines the location of the DTD which covers the Borland specific deployment descriptors. This should not be necessary if you have borland in your classpath. If you do not, you should use a nested `<dtd>` element, described in the ejbjar task documentation.	no
ejbdtd	**Deprecated**. Defines the location of the ejb-jar DTD in the class hierarchy. This should not be necessary if you have borland in your classpath. If you do not, you should use a nested `<dtd>` element, described in the ejbjar task documentation.	no
generateclient	If true, turn on the generation of the corresponding ejbjar (default = false)	no
version	set the Borland Application Version. • 4 means B.A.S (Borland Application Server) 4.x, target will add ejb-inprise.xml file • 5 means B.E.S (Borland Application Server) 5.x, target will add ejb-borland.xml file	No, defaults to 4
java2iiopParams	If filled, the params are added to the java2iiop command (ex: -no_warn_missing_define)	no

Examples

The following build.xml snippet is an example of how to use Borland element into the ejbjar task

```
    <ejbjar srcdir="${build.classes}" basejarname="vsmp" descriptordir="${rsc.dir}/hrmanager">
        <borland destdir="lib" verify="on" generateclient="on" version="5">
          <classpath refid="classpath"/>
        </borland>
        <include name="**\ejb-jar.xml"/>
        <support dir="${build.classes}">
          <include name="demo\*.class"/>
          <include name="demo\helper\*.class"/>
         </support>
     </ejbjar>
```

```
The borland element will generate into the lib dir an ejb jar file using the deployment
descriptor placed into the ${rsc.dir}/hrmanager directory.
The verify phase is turned on and the generate client phase as well.
```

Assertions

The `assertions` type enables or disables the Java 1.4 assertions feature, on a whole Java program, or components of a program. It can be used in `<java>` and `<junit>` to add extra validation to code.

Assertions are covered in the J2SDK 1.4 documentation, and the Java Language Specification.

The key points to note are that a `java.lang.AssertionError` is thrown when an assertion fails, and that the facility is only available on Java 1.4 and later. To enable assertions one must set `source="1.4"` (or later) in `<javac>` when the source is being compiled, and that the code must contain `assert` statements to be tested. The result of such an action is code that neither compiles or runs on earlier versions of Java. For this reason Ant itself currently contains no assertions.

When assertions are enabled (or disabled) in a task through nested assertions elements, the class loader or command line is modified with the appropriate options. This means that the JVM executed must be a Java 1.4 or later JVM, even if there are no assertions in the code. Attempting to enable assertions on earlier VMs will result in an "Unrecognized option" error and the JVM will not start.

Attributes

Attribute	Description	Required
enableSystemAssertions	Flag to turn system assertions on or off.	No; default is "unspecified"

When system assertions have been neither enabled nor disabled, then the JVM is not given any assertion information - the default action of the current JVMs is to disable system assertions.

Note also that there is no apparent documentation for what parts of the JRE come with useful assertions.

Nested elements

enable

Enable assertions in portions of code. If neither a package nor class is specified, assertions are turned on in *all* (user) code.

Attribute	Description	Required
class	The name of a class on which to enable assertions.	No
package	The name of a package in which to enable assertions on all classes. (Includes subpackages.) Use "..." for the anonymous package.	No

disable

Disable assertions in portions of code.

Attribute	Description	Required
class	The name of a class on which to disable assertions.	No
package	The name of a package in which to disable assertions on all classes. (Includes subpackages.) Use "..." for the anonymous package.	No

Because assertions are disabled by default, it only makes sense to disable assertions where they have been enabled in a parent package.

Examples

Example: enable assertions in all user classes

All classes not in the JRE (i.e. all non-system classes) will have assertions turned on.

```
<assertions>
  <enable/>
</assertions>
```

Example: enable a single class

Enable assertions in a class called Test

```
<assertions>
  <enable class="Test"/>
</assertions>
```

Example: enable a package

Enable assertions in the `org.apache` package and all packages starting with the `org.apache.` prefix

```
<assertions>
  <enable package="org.apache"/>
</assertions>
```

Example: System assertions

Example: enable system assertions and assertions in all `org.apache` packages except for Ant (but including `org.apache.tools.ant.Main`)

```
<assertions enableSystemAssertions="true">
  <enable package="org.apache"/>
  <disable package="org.apache.tools.ant"/>
  <enable class="org.apache.tools.ant.Main"/>
</assertions>
```

Example: disabled and anonymous package assertions

Disable system assertions; enable those in the anonymous package

```
<assertions enableSystemAssertions="false">
  <enable package="..."/>
</assertions>
```

Example: referenced assertions

This type is a datatype, so you can declare assertions and use them later

```
<assertions id="project.assertions">
  <enable package="org.apache.test"/>
</assertions>

<assertions refid="project.assertions"/>
```