CONTENTS INCLUDE:



# Getting Started with Apache Ant

By James Sugrue

## ABOUT APACHE ANT

Apache Ant is an XML based tool for automating software build processes. Starting out as part of the Apache Tomcat codebase, Ant got its first standalone release in July 2000. Today it is the most widely used build tool for Java projects, enabling developers to adopt agile principles: most importantly test-driven development.

Hot Tip

**Download Instructions**

You can download the latest Ant distribution (1.8.1) as a standalone tool from http://Ant.apache.org/. Ant is also built into most Java development IDEs, such as Eclipse, and NetBeans which uses it as its internal build system.

## The Anatomy of an Ant Script

A typical Ant script consists of a single build.xml file. The root element of the build script is the project tag. Within the project element there are one or more targets specified. A target contains a set of tasks to be executed.

The project element can specify a default target if no target is chosen during execution of the build script.

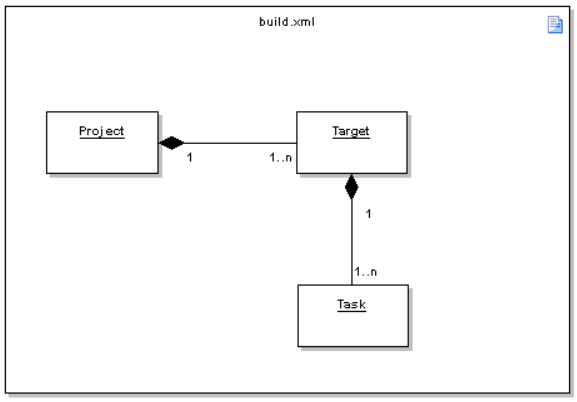



**Figure 1:** The basic structure of an Ant build script.

The most important concept in Ant is that of dependencies. Each target may depend on other targets. When running, Ant resolves these dependencies so that your script gets executed in the order you have specified.

| Attribute | Description | Required |
|---|---|---|
| name | Name of the project. | No |
| basedir | Base directory from which all path calculations are done | No |
| default | The default target to run when no other target has been specified | No |

**Table 1:** Project element attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name and identifier of this target | Yes |
| depends | Comma separated list of other targets that this target depends on | No |
| if | The name of a property which must be set for this target to run | No |
| unless | The name of a property which must not be set for this target to run | No |
| description | Description of this target | No |
| extensionOf | Add the target to the depends list of the named extension point | No |

**Table 2:** Target element attributes

Hot Tip

**Extension Points**

Introduced in Ant 1.8.0 <extension-point> is similar to a <target> with its name and depends attributes. However, it does not contain any tasks and is used to collect targets that contribute to a state in the dependency relationships of the script.

## Properties

Properties can be defined within the build script or in separate property files to provide more customizable build scripts.

| Attribute | Description | Required |
|---|---|---|
| name | Name of the property | No |
| value | The value of the property | One of these when using name attribute |
| location | Set value to absolute filename. If the value passed through is not an absolute path it will be made relative to the project basedir | |
| refId | Reference to an object defined elsewhere | |





Getting Started with Apache Ant www.dzone.com

| | | |
|---|---|---|
| file | Location of the properties file | When not using the name attribute (i.e. loading external properties) |
| resource | Location in the classpath of the properties file | |
| url | URL pointing to properties file | |
| basedir | The basedir to calculate the relative path from | No. Default is project basedir |
| classpath / classpathref | The classpath to use when looking up a resource | No |
| environment | Prefix to use when accessing environment variables | No |
| prefix | Prefix to apply to properties loaded using file, resource or url | No |

**Table 3:** Property element attributes

All system properties that can be accessed from the System. `getProperties()` methods in Java can be used in Ant. Additionally, the following built-in properties are available:

| Property Name | Description |
|---|---|
| basedir | Absolute path of projects basedir |
| ant.core.lib | Absolute path of the ant.jar file |
| ant.file | Absolute path of buildfile |
| ant.home | Home directory of Ant |
| ant.java.version | Java version detected |
| ant.project.default-target | Name of the default target |
| ant.project.invoked-targets | List of targets specified in the command line when invoking this project |
| ant.project.name | Name of the project that is running |
| ant.version | Version of Ant |

**Table 4:** Built-in properties

## Path Structures

Path structures can be created using a series of `<pathelement>` tags. For example, a classpath can be created using:

```
<classpath>
   <pathelement location="/path/jarfile.jar"/>
   <pathelement path="/path/lib/jar1.jar;/path/lib/jar2.jar"/>
</classpath>
```

## File Sets

In order to bundle files together, Ant provides the `<fileset>` tag:

| Property Name | Description |
|---|---|
| casesensitive | Whether include/excludes patterns must be treated with case sensitivity. Default true |
| dir | Root of the directory tree of this fileset |
| defaultexcludes | If default excludes should be used (set of definitions that are always excluded) |
| erroronmissingdir | If true causes a build error, if false the fileset is ignored |
| excludes | List of patterns of files to exclude |
| excludesfile | Name of a file to exclude |
| file | Shortcut for specifying a fileset containing a single file |
| followsymlinks | Whether symbolic links should be followed. Default true |

**Table 5:** Fileset attributes

## Built-in Tasks

The following sections list out the most commonly used tasks in Ant build scripts. Required attributes are marked in bold.

## Core Java Related Tasks

This section gives a complete reference of all tasks and their attributes that are most commonly used by Java Developers.

### Compiling Java Code

Compilation is achieved with the <javac> task.

| Attribute | Description |
|---|---|
| srcdir | Location of java files to compile |
| bootclasspath/ boothclasspathref | Location of bootstrap class files or reference to a predefined path |
| classpath/ classpathref | Classpath to use for compilation or reference to a predefined path |
| compiler | Compiler implementation to use. Default is current VM |
| debug | Whether source should be compiled with debug information (-g parameter). Defaults to off |
| debuglevel | Can be lines, var or source. Used in the –g parameter for the compiler |
| depend | Enables dependency tracking on jikes and classic compilers |
| deprecation | Whether source should compile with deprecation information |
| destdir | Destination for compiled .class files |
| encoding | Encoding of source files |
| errorProperty | Property to set to true if compilation fails |
| excludes | Comma separated list of files that must be excluded from compilation. Wildcard patterns can be used |
| excludesFile | File that contains the exclusion list |
| executable | Path to javac executable to use when fork set to yes |
| extdirs | Location of installed extensions |
| fork | Whether to execute javac using the JDK compiler externally. Default no |
| failonerror | Whether compilation error fails the build. Default true |
| includes | Comma separated list of files that must be included in compilation. Wildcard patterns can be used |
| includeAntRuntime | Whether to include ANT runtime libraries in the classpath. Default yes |
| includeDestClasses | Whether classes compiled to the dest directory are included in the classpath. Default true. When false, causes recompilation of dependent classes |
| includesFile | File containing the exclusion list |
| includeJavaRuntime | Whether to include default libraries from VM. Default no |
| listfiles | Whether source files to be compiled will be listed. Default no |
| memoryInitialSize / memoryMaximumSize | Initial and maximum memory sizes for VM if run externally |
| nowarn | Whether –nowarn should be used with the compiler. Defaults to off |
| optimize | Whether to compile with optimization, ignored since JDK1.3 |
| source | Value of the –source command line switch |
| sourcepath/ sourcepathref | Defaults to value of srcdir or reference to a predefined path |
| target | Generate class files for a particular VM version |
| tempdir | Temporary file location if fork set to yes |
| updatedProperty | Property to set for successful compilation |
| verbose | Use verbose output on the compiler |

**Table 6:** javac tasks properties

Additional command line arguments can be passed through to the compiler using the `<compilerarg>` nested element.

**Hot Tip**

**Compiler Choice**

To use different compilers set the `build.compiler` property to classic (1.1, 1.2) modern (1.3-1.6) or choose a separate compiler such as jikes, jvc, kjc, gcj or sj.

Class file dependencies can be managed using the `<depend>` task.

| Attribute | Description |
|---|---|
| srcdir | Location of Java files to compile. Will be examined to determine which files are out of date |
| cache | Directory where dependency information is stored and retrieved. No cache if not used |
| classpath | Classpath from which dependencies also need to be checked |
| closure | If true, all classes depending on an out-of-date class are deleted |
| destdir | Location of class file to be analyzed |
| dump | If true dependency info is written to the debug level log |
| warnOnRmiStubs | Disables warnings about files that look like rmic generated stubs but no .java source |

**Table 7:** Depend task properties

**Hot Tip**

**Ivy For Dependency Management**

Ivy (http://ant.apache.org/ivy), a sub project of Ant, can also be used to manage dependencies.

### Distributing Compiled Code

Jar files can be created using the `<jar>` task.

| Attribute | Description |
|---|---|
| destfile | JAR file to create |
| basedir | The directory to build from |
| compress | Compress data in jar. Defaults true |
| createUnicodeExtraFields | Whether to create unicode extra fields to store file names a second time inside the entry's metadata |
| defaultexcludes | Whether default excludes should be used. Default true |
| duplicate | Behavior when a duplicate file is found – add (default), preserve, fail |
| keepcompression | For entries coming from other archives, keep its compression, overriding compress |
| encoding | Character encoding for filenames. Default UTF8 |
| excludes | List of patterns of files to exclude |
| excludesfile | The name of a file that defines an exclude pattern |
| fallbacktoUTF8 | If the specified encoding cannot be used, whether to fallback to UTF8 |
| filesonly | Store only file entries. Default false |
| filesetmanifest | Behavior for when a manifest is found in a zipfileset. Can merge, mergewithoutmain or skip (default) |
| flattenAttributes | Merge attributes occurring more than once in a section into one single attribute |
| includes | List of patterns of files to include |
| includesfile | The name of a file that defines an include pattern |
| index | Whether to create an index list to speed up classloading. Default false |
| indexMetaInf | Whether to include META-INF in the index. Default false |
| level | Compression level for files from 0 (none) to 9 (maximum) |
| manifest | Location of manifest for jar |
| manifestencoding | Encoding to use for manifest. Default is platform encoding |
| mergeClassPathAttributes | Merge classpath attributes of different manifests when merging |
| preserve0permissions | If a file has permissions value of 0, it will preserve this instead of applying default values |
| roundup | Whether to round up file modification time to the next even number of seconds |
| strict | How to handle breaks of packaging version specification Fail, warn or ignore (default) |
| update | Whether to overwrite files that already exist. Default false |
| useLanguageEncodingFlag | Whether to set language encoding if encoding set to UTF8 only |
| whenmanifestonly | Behavior when no files match – fail, skip or create (default) |

**Table 8:** Jar task properties

**Hot Tip**

**War and Ear Archive Tasks**

Both <war> and <ear> tasks have similar attributes to the <jar> task, adding in attributes for web.xml or application.xml respectively.

Additionally, you can sign jar archives using the `<signjar>` task.

| Attribute | Description |
|---|---|
| alias | The alias to sign under |
| jar | The jar file to sign |
| storepass | The password for keystore integrity |
| executable | Specific jarsigner executable to use in place of default in JDK |
| force | Force signing if already signed or not out of date |
| internals | Whether to include the .SF file inside signature block. Default false |
| keypass | The password for the private key |
| keystore | Keystore location |
| lazy | Whether a signature file being present means the jar is signed |
| maxmemory | Maximum memory the VM will use when signing |
| preservelastmodified | Signed files keep the same modification time as original jar files |
| sectionsonly | Whether to compute the hash of entire manifest |
| sigfile | The name of the .SF or .DSA file |
| signedjar | The name of the signed jar file |
| storetype | The keystore type |
| tsacert | Alias in keystore for timestamp authority |
| tsaurl | URL for timestamp authority |
| verbose | Whether to use verbose output. Default false |

**Table 9:** Signjar task properties

Manifests can be included using the `<manifest>` task.

| Attribute | Description |
|---|---|
| file | Manifest file to create or update |
| encoding | Encoding to read existing manifest |

| | |
|---|---|
| flattenAttributes | Merge attributes occurring more than once in a section into one single attribute |
| mergeClassPathAttributes | Merge classpath attributes of different manifests when merging |
| mode | Either update or replace (default) |

**Table 10:** Manifest task properties

### Generating Documentation

JavaDoc generation is done through the `<javadoc>` task.

| Attribute | Description |
|---|---|
| sourcepath<br>sourcepathref<br>sourcefiles | Location of the source files for the task. At least one of these attributes required for this specification |
| destdir | Destination directory required unless a doclet is specified |
| access | Access mode (public, private, protected, package) |
| additionalparam | Additional parameters to pass through |
| author | Include @author parts |
| bootclasspath / bootclasspathref | Location of class files loaded by bootstrap class loader |
| bottom | Bottom text for each page |
| breakiterator | Use new breakiterator algorithm |
| charset | Charset for cross platform viewing |
| classpath/classpathref | Location of classpath |
| defaultexcludes | Whether default excludes should be used |
| docencoding | Encoding of output file |
| docfilessubdirs | Deep copy of doc-file subdirectories |
| doclet/docletpathref | Classfile that starts to doclet used |
| doctitle | Title for the package index page |
| encoding | Source file encoding |
| excludepackagenames | Packages to exclude from javadoc |
| executable | Specify a javadoc executable instead of VM default |
| extdirs | Location of installed extensions |
| failonerror | Stops build process if command fails |
| footer | Footer text for each page |
| group | Group specified packages together in overview page |
| header | Header text for each page |
| helpfile | Specifies help file to include |
| includenosourcepackages | When true includes packages with no source |
| link | Create links to javadoc output at given url |
| linksource | Generate links to source files |
| locale | Locale to be used |
| maxmemory | Maximum amount of memory to allocate in VM |
| nodeprecated | Do not include @deprecated information |
| nodeprecatedlist | Do not generate deprecated list |
| notree | Do not generate class hierarchy |
| noindex | Do not generate index |
| nohelp | Do not generate help link |
| nonavbar | Do not generate naviation bar |
| noqualifier | Enables –noqualifier argument for a list of packages (or all) |
| overview | Read overview documentation from HTML file |
| packagenames | List of package files to use |
| packageList | File containing packages to use |
| public | Show only public classes and members |
| protected | Show only protected/public classes and members |
| package | Show only package/protected/public classes and members |
| private | Show all classes and members |
| serialwarn | Warn about @serial tag |
| source | Source level used for compilation |
| splitindex | Split index into one file per letter |
| stylesheetfile | Specifies CSS stylesheet |
| use | Create class and package usage pages |
| verbose | Output all messages of javadoc process |
| version | Include @version parts |
| windowtitle | Browser window title for documentation |

**Table 11:** JavaDoc task properties

### Executing Java Classes

Java classes can be executed from Ant using the `<java>` task.

| Attribute | Description |
|---|---|
| classname<br>jar | The jar file or classname to execute. If using jar, fork must be set to true |
| append | Whether output/error files should be appended or overwritten |
| args | Arguments for class. Better to use nested <arg> elements |
| classpath / classpathref | Classpath to use for execution |
| clonevm | Clones system properties and bootclasspath of forked VM to be the same as the Ant VM |
| dir | Directory to invoke VM in |
| error | File to write error (System.err) output to |
| errorproperty | Property to store error output from command |
| failonerror | Stops Ant build process if failure occurs |
| fork | Executes the class in another VM |
| input | File where standard input is taken from |
| inputstring | String where standard input is taken from |
| jvm | Command used to invoke JVM. Default is java |
| javmargs | Arguments for forked JVM. Better to use nested <jvmarg> |
| logError | Shows error output in Ant log |
| maxmemory | Maximum memory to allocate to the forked VM |
| newenvironment | When fork=true, do not propogate current environment variables |
| output | File to write output to |
| outputproperty | Property to store output of the command |
| resultproperty | Property where return code of command is stored |
| spawn | Need fork set to true. Will spawn a process independent of calling Ant process |
| timeout | Timeout for the command to execute within before being stopped |

**Table 12:** Java task properties

### Running Unit Tests

JUnit tests are executed from Ant through the `<junit>` task

| Attribute | Description |
|---|---|
| clonevm | Clones system properties and bootclasspath of forked VM to be the same as the Ant VM |
| dir | Directory from which to invoke the VM |
| errorproperty | Property to set when errors occur |

| | |
|---|---|
| failureproperty | Property to set when failure occurs |
| filtertrace | Filter out JUnit or Ant stack frames from stack traces in errors or failures |
| fork | Run JUnit in a separate VM |
| forkmode | How many VMs to create when forking tests. (preTest, perBatch or once) |
| haltonerror | Stop build process if an error happens |
| haltonfailure | Stops build process if a test fails |
| includeAntruntime | Stops build process if a test fails |
| jvm | Command used to invoke VM, default java |
| logfailedtests | Log a FAILED message for each failed test to Ant's logging system |
| maxmemory | Maximum memory to allocate to forked VM |
| newenvironment | When fork=true, do not propogate current environment variables |
| outputformatters | Send output generated by tests to test formatters |
| printsummary | Print statistics for each test case. (on, off, withOutAndErr) |
| reloading | Whether a new classloader should be instAntiated for each test |
| showoutput | Send output to Ant log |
| tempdir | Location for temporary files |
| timeout | Cancel tests if not complete in specified time |

**Table 13:** JUnit task properties

Tests are defined in nested elements within the `<junit>` task. Batch tests are defined using the `<batchtest>` tag:

| Attribute | Description |
|---|---|
| name | Name of test class |
| errorproperty | Property to set when errors occur |
| failureproperty | Property to set when failure occurs |
| filtertrace | Filter out JUnit or Ant stack frames from stack traces in errors or failures |
| fork | Run tests in a separate VM, overriding the value set in the <junit> task |
| haltonerror | Stop build process if an error happens |
| haltonfailure | Stops build process if a test fails |
| if | Property condition to run test against, if set |
| todir | Directory to write reports to |
| unless | Property condition to run test against, if not set |

**Table 14:** JUnit batch test definition

**Hot Tip**

**Single Tests**

If you just need to run a single test, the <test> tag can be used to specify the test. This contains similar attributes to the <batchtest> tag.

Test results can be written to different formats, using the `outputformatters` attribute in the `<junit>` task. The following table shows the options for formatter definition:

| Attribute | Description |
|---|---|
| type | A predefined formatter type of either xml, plain, brief or failure |
| classname | If no type specified, use a customer formatter implementing `org.apache.tools.Ant.taskdefs.optional.junit.JUnitResultFormatter` |
| if | Use formatter if property is set |
| unless | Use formatter if property is not set |
| usefile | If output should be sent to a file. Default true |

**Table 15:** JUnit formatter definition

Once tests are completed, reports can be generated for the tests using the `<junitreport>` task.

| Attribute | Description |
|---|---|
| todir | The directory that will contain the results |
| tofile | The name of the XML file that will aggregate all previously generated results |

**Table 16:** JUnitReport task properties

The `<junitreport>` task contains a fileset that collects all the reports from previous Junit tests.

To output to a report file, use the internal `<report>` tag.

| Attribute | Description |
|---|---|
| format | Format of report (frames or noframes) |
| styledir | Directory containing stylesheets. Files must be junit-frames.xsl or junit-noframes.xsl |
| todir | The directory that files are written to. By default this is the current directory |

**Table 17:** Report tag for <junitreport>

## Infrastructure Tasks

There are a number of other core tasks related to file operations. The following is an overview of these tasks:

| Task | Description |
|---|---|
| attrib /chmod | Changes permission of a file (attrib for Windows, chmod for Unix) |
| checksum | Generates a checksum for files |
| concat | Concatenates multiple files into one single file, or to the console |
| copy | Copies a file, or collection of files, to another location |
| delete | Deletes a file, directory or collection of files |
| exec | Executes a system command |
| ftp | Provides basic FTP functionality |
| get | Gets a file from a URL |
| import / include | Imports or includes another build file into the current Ant script |
| mail | Sends mail over SMTP |
| mkdir | Creates a new directory |
| move | Moves a file, or collection of files, to another location |
| record | Listens to the build process and outputs to a file |
| replace | Replaces occurances of a string in file or collection of files |
| replaceregexp | Replaces occurances of a string in file or collection of files using regular expressions |
| sql | Executes SQL statements to a database using JDBC |
| sync | Synchronizes a target directory with a list of files |
| zip / unzip tar / untar | Creates a zip file / unzips an existing zip. Also provides functionality for tar files |

**Table 18:** Overview of basic infrastructure tasks

## SCM Related Tasks

ANT provides a number of tasks for connecting with different source control management systems. The core support deals with CVS.

| Task | Description |
|---|---|
| cvs | Handles commands to be run against CVS, defaulting with checkout |
| cvschangelog | Generates a report of change logs recorded in CVS |
| cvspass | Adds entries to a .cvspass file |
| cvstagdiff | Generates a report of the changes between two tags/dates from CVS |
| cvsversion | Retrieves CVS client and server version |
| patch | Applies a diff file to originals |

**Table 19:** CVS task properties

There are also tasks available for interfacing with ClearCase, Visual Source Safe, Pvcs, Perforce and Continuus. Additional tasks can be found online for DCVS systems such as Git and Mercurial.

## Property Tasks

Ant provides some tasks that deal with managing properties throughout your build process.

| Task | Description |
|---|---|
| available | Sets property if file, directory or class in classpath is available at runtime |
| basename | Sets property to last element of specified path |
| buildnumber | Reads build number from a specified file |
| condition | Set property if a condition is true |
| dirname | Set property to the directory path of a specified file |
| echoproperties | Display all properties to file or console |
| loadfile | Loads a file into a property |
| loadproperties | Load a property file contents as Ant properties |
| makeurl | Converts filenames into URLs |
| pathconvert | Converts a file list or path structure to a separated string for the target platform |
| property | Set a property |
| propertyfile | Creation and modification of property files |
| uptodate | Set property if specified target file is newer than source files |
| whichresource | Find class or resource |
| xmlproperty | Loads properties from property file written in XML |

**Table 20:** Ant Property Tasks

**Hot Tip**

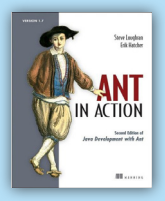
**Writing Your Own Tasks**

If Ant doesn't provide you with the functionality that you need, you can write your own Ant tasks in Java by extending org.apache.tools.ant.Task.

## ABOUT THE AUTHOR

**James Sugrue** has been editor at both Javalobby and EclipseZone for over two years, and loves every minute of it. By day, James is a software architect at Pilz Ireland, developing killer desktop software using Java and Eclipse all the way. While working on desktop technologies such as Eclipse RCP and Swing, James also likes meddling with up and coming technologies such as Eclipse e4. His current obsession is developing for the iPhone and iPad, having convinced himself that it's a turning point for the software industry.

## RECOMMENDED BOOKS




A single application of increasing complexity, followed throughout the book, shows how an application evolves and how to handle the problems of building and testing. Reviewers have praised the book's coverage of large-projects, Ant's advanced features, and the details and depth of the discussion-all unavailable elsewhere.

**BUY NOW**

books.dzone.com/books/ant-action



DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

**Refcardz Feedback Welcome**
refcardz@dzone.com

**Sponsorship Opportunities**
sales@dzone.com

ISBN-13: 978-1-934238-75-2
ISBN-10: 1-934238-75-9
50795
9 781934 238752
$7.95

Version 1.0