



Apache Ant

Another Neat Tool



INTRODUCTION

Apache Ant is a Java based build tool from Apache Software Foundation. Apache Ant's build files are written in XML and take advantage of the open standard, portable and easy to understand nature of XML.

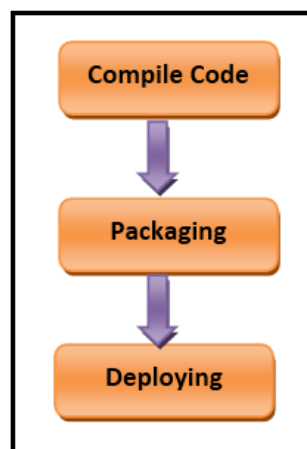
Features of Apache Ant:

- Ant is the most complete Java build and deployment tool available.
- Ant is platform neutral and can handle platform specific properties such as file separators.
- Ant can be used to perform platform specific tasks such as modifying the modified time of a file using 'touch' command.
- Ant scripts are written using plain XML. If you are already familiar with XML, you can learn Ant pretty quickly.
- Ant is good at automating complicated repetitive tasks.
- Ant comes with a big list of predefined tasks.
- Ant provides an interface to develop custom tasks.
- Ant can be easily invoked from the command line and it can integrate with free and commercial IDEs.

Why do you need a build tool?

- Compile code
- Package the binaries
- Deploy the binaries to the test server
- Test your changes
- Copy code from one location to another

Basic Workflow:



Workflow of Ant

Compile Code:

Ant can compile multiple java source code (**.java**) files automatically from the specified path and it will keep all the java class (**.class**) files into given path.

Packaging:

Ant can build or create the binaries from the different sources with variety kind of files.

For example: we can create **jar/war/ear**.

Binary or Project Artifact	Description
Jar (Java Archive)	Jar is the collection of .class files
War (Web Archive)	War is the web application, so it will have the different combination of files. For example: .class files, jar files, jsp's, Servlets, Html, Images, lib and web.xml etc.
Ear (Enterprise Archive)	Ear also one of the web application, and it will have different kind of files like war.

Deploying:

Whatever the binaries we have generated from the source code we can deploy that binaries into web server or application server using ant script.

Ant Script:

Ant script is the code written in **build.xml** file. This is the default ant script file to compile, package and deploy the java based web applications.

You can create the build.xml file anywhere in the system, by default ant will look for the build.xml file only, or else we can create with the other name also. We can see that in the next topics.

INSTALLING AND CONFIGURING

Installing Apache Ant:

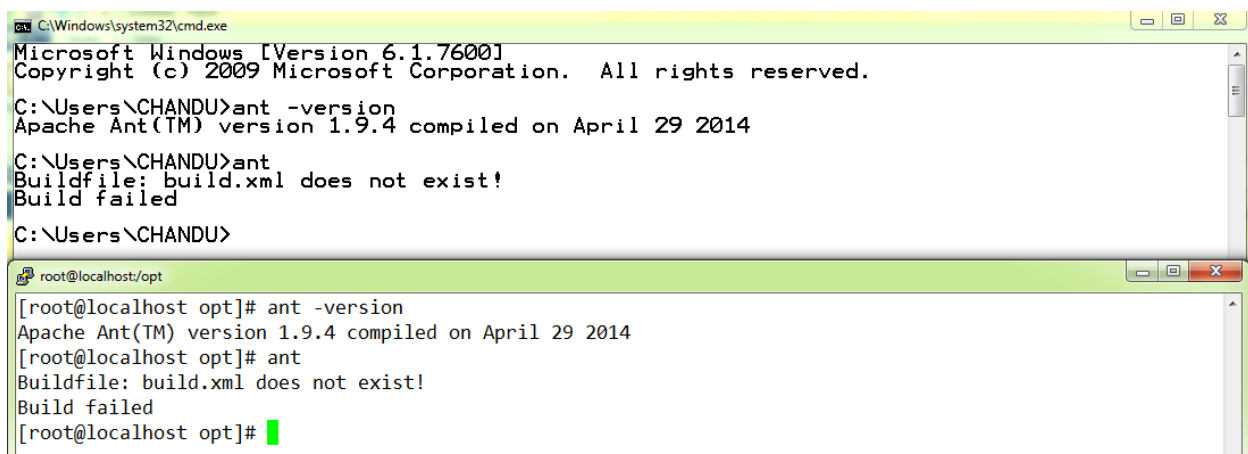
Windows:

- Ensure that the **JAVA_HOME** environment variable is set to the folder where your JDK is installed.
- Download the binaries from **<http://ant.apache.org>**
- Unzip the zip file to a convenient location using WinZip, WinRAR, 7-zip or similar tools, say **c:\ folder**.
- Create a new environment variable called **ANT_HOME** that points to the Ant installation folder, in this case **c:\apache-ant-1.9.4-bin** folder.
- Append the path to the Apache Ant batch file to the **PATH** environment variable. In our case this would be the **c:\apache-ant-1.9.4-bin\bin** folder

Linux:

- Ensure that the **JAVA_HOME** environment variable is set to the directory where your JDK is installed.
- Download the binaries from **<http://ant.apache.org>**
- Extract the zip file to a convenient location using **tar** or **unzip** commands.
- Create a new environment variable called **ANT_HOME** that points to the Ant installation directory, in this case **/opt/ant** directory.
- Append the path to the Apache Ant file to the **PATH** environment variable. In our case this would be the **/opt/ant/bin** directory.

Once if you have done this process, then you can check whether it is configured or not using the commands called **ant** or **ant -version**.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\CHANDU>ant -version
Apache Ant(TM) version 1.9.4 compiled on April 29 2014

C:\Users\CHANDU>ant
Buildfile: build.xml does not exist!
Build failed

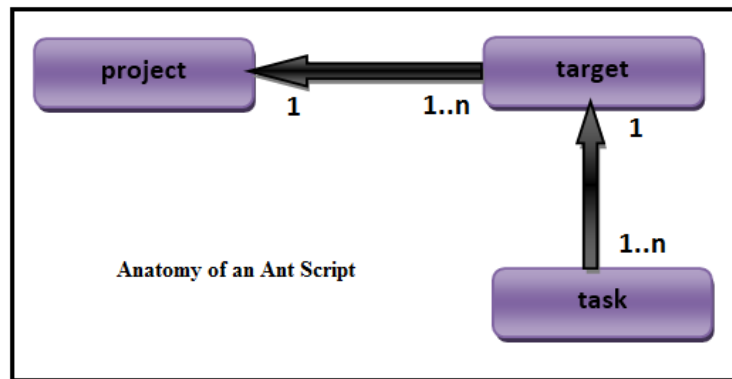
C:\Users\CHANDU>

root@localhost:/opt
[root@localhost opt]# ant -version
Apache Ant(TM) version 1.9.4 compiled on April 29 2014
[root@localhost opt]# ant
Buildfile: build.xml does not exist!
Build failed
[root@localhost opt]#
```

WORKING WITH ANT

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.



Project element:

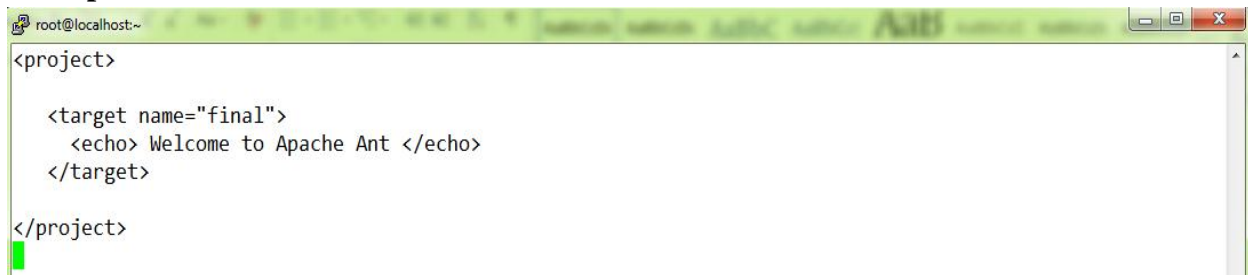
All ant scripts requires a project element, the XML project element has three attributes:

Attribute	Description	Required
name	Name of the project.	No
basedir	The base directory (or) the root folder for the project.	No
default	The default target for the build script. A project may contain any number of targets. This attribute specifies which target should be considered as the default.	No

Target element:

A target is a collection of tasks that you want to run as one unit. In our example, we have a simple target to provide an informational message to the use.

Example:

A screenshot of a terminal window with a green title bar. The terminal shows the following XML code:

```
<project>

  <target name="final">
    <echo> Welcome to Apache Ant </echo>
  </target>

</project>
```

Target attributes:

Attribute	Description	Required
name	The name and identifier of this target.	Yes
depends	Comma separated list of all targets that this target depends on.	No
description	A short description of the target.	No
if	Allows the execution of a target based on the trueness of a conditional attribute	No
unless	Adds the target to the dependency list of the specified Extension Point. An Extension Point is similar to a target, but it does not have any tasks.	No
extensionOf	Add the target to the depends list of the named extension point.	No

Targets can have dependencies on other targets. For example, a **deploy** target may have a dependency on the **package** target, and the **package** target may have a dependency on the **compile** target and so forth. Dependencies are denoted using the **depends** attribute.

For example:

```
<project name="example" default="deploy">

  <target name="compile">
    <echo> This is compile target </echo>
  </target>

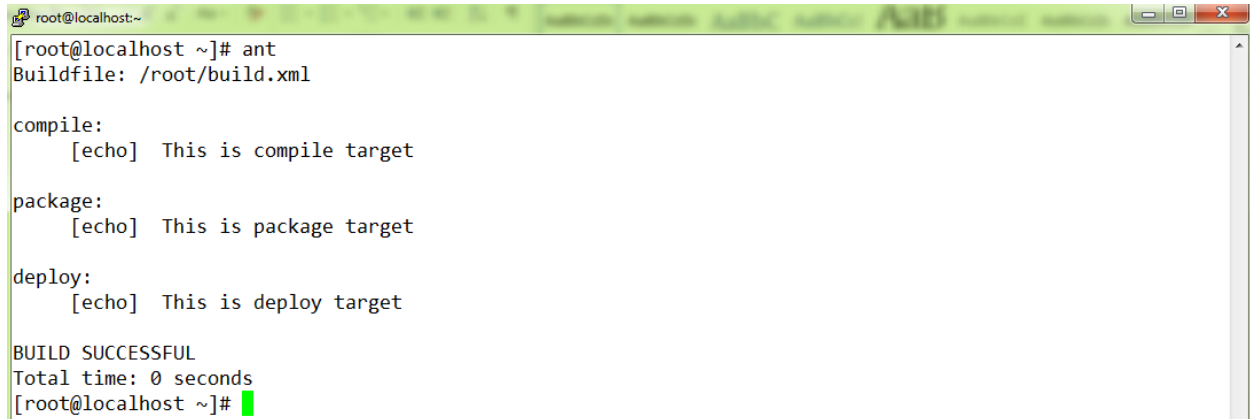
  <target name="package" depends="compile">
```

```
<echo> This is package target </echo>
</target>

<target name="deploy" depends="package">
  <echo> This is deploy target </echo>
</target>

</project>
```

Output:

A terminal window titled 'root@localhost:~' showing the output of an Ant build. The window has a standard Linux terminal interface with a title bar and window controls. The output text is as follows:

```
[root@localhost ~]# ant
Buildfile: /root/build.xml

compile:
  [echo] This is compile target

package:
  [echo] This is package target

deploy:
  [echo] This is deploy target

BUILD SUCCESSFUL
Total time: 0 seconds
[root@localhost ~]#
```

ANT PROPERTIES

Properties:

Properties can be defined within the build script or in separate property files to provide more customizable build scripts. This allows the properties to be changed from one build to another, or from one environment to another.

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 in somewhere else.	
file	Location of the property file.	When not using the name attribute (i.e. loading external properties)
resource	Location in the classpath of the property file	
url	URL pointing to properties file	
basedir	The basedir to calculate the relative path from	No (Default is project basedir)
classpath/classpathref	The class 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

Pre-defined ant properties:

By default, Ant provides the following pre-defined (built-in) properties that can be used in the build file.

Attribute	Description	Required
ant.file	The full location of the build file	No
ant.version	The version of the Apache ant installed on your system	No
basedir	The basedir of the build, as specified in the basedir attribute of the project element	No
ant.java.version	The version of the JDK that is used by Ant	No
ant.project.name	The name of the project, as specified in the name attribute of the project element	No
ant.project.default-target	The default target of the current project	No
ant.project.invoked-targets	Comma separated list of the targets that were invoked in the current project	No
ant.core.lib	The full location of the ant jar file	No
ant.home	The home directory of Ant installation	No
ant.library.dir	The home directory for Ant library files - typically ANT_HOME/lib folder	No

In addition to the above, the user can define additional properties using the **property** element. An example is presented below which shows how to define a property called **print.name**.

build.xml:

```

<project name="example" default="info">
  <property name="print.name" value="CHANDRA SHEKHAR REDDY" />
  <target name="info">
    <echo> This is ${print.name} </echo>
    <echo> Welcome to ${ant.version}. </echo>
  </target>
</project>

```

Output:

```
root@localhost:~  
[root@localhost ~]# ant  
Buildfile: /root/build.xml  
  
info:  
  [echo] This is CHANDRA SHEKHAR REDDY  
  [echo] Welcome to Apache Ant(TM) version 1.9.4 compiled on April 29 2014.  
  
BUILD SUCCESSFUL  
Total time: 0 seconds  
[root@localhost ~]#
```

Ant – Property files:

Setting properties directly in the build file is okay if you are working with a handful of properties. However, for a large project, it makes sense to store the properties in a separate property file.

Storing the properties in a separate file allows you to reuse the same build file, with different property settings for different execution environment. For example, build properties file can be maintained separately for DEV, TEST and PROD environments.

Specifying properties in a separate file is useful when you do not know the values for a property (in a particular environment) up front. This allows you to perform the build in other environments where the property value is known.

There is no hard and fast rule, but typically the property file is named **build.properties** and is placed alongside the **build.xml** file. You could create multiple build properties file based on the deployment environments - such as **build.properties.dev** and **build.properties.test**.

The contents of the build property file are similar to the normal java property file. They contain one property per line. Each property is represented by a name and a value pair. The name and value pair are separated by an equals sign. It is highly recommended that the properties are annotated with proper comments.

The following shows a **build.xml** and an associated **build.properties** file.

build.xml:

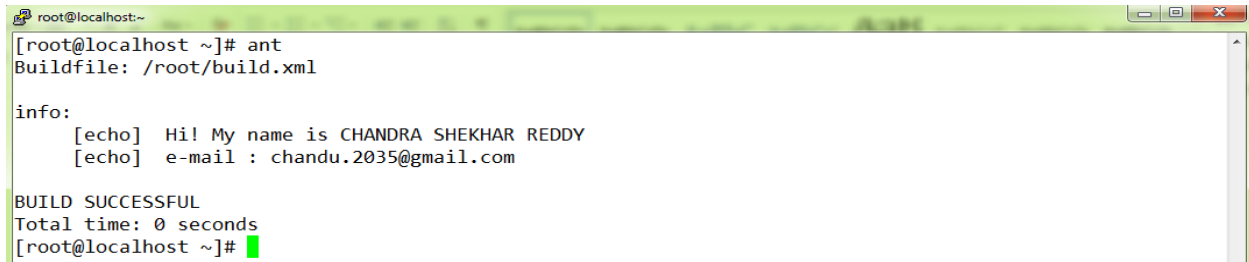
```
<project name="example" default="info">  
  <target name="info">  
    <property file="build.properties" />  
    <echo> Hi! My name is ${print.name} </echo>  
  </target>  
</project>
```

build.properties

```
# build.property file
```

```
print.name=CHANDRA SHEKHAR REDDY  
email=chandu.2035@gmail.com
```

Output:



```
root@localhost:~  
[root@localhost ~]# ant  
Buildfile: /root/build.xml  
  
info:  
  [echo] Hi! My name is CHANDRA SHEKHAR REDDY  
  [echo] e-mail : chandu.2035@gmail.com  
  
BUILD SUCCESSFUL  
Total time: 0 seconds  
[root@localhost ~]#
```

ANT 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 Set:

In order to bundle files together, Ant provides the `<fileset>` tag. It usually used as a filter to include and exclude files that match a particular pattern.

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.	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. <i>Since Apache Ant 1.7.1</i>	No

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>
```

Pattern Set

A pattern set is a pattern that allows to easily filter files or folders based on certain patterns. Patterns can be created using the following metacharacters.

- `?` - Matches one character only
- `*` - Matches zero or many characters
- `**` - Matches zero or many directories recursively

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.

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

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

Examples:

```
<filelist
  id="libfiles"
  dir="{lib.src}"
  files="csr.jar,servlet-api.jar"/>
```

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 refidattribute. 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.

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.	<i>none</i>	No
recurse	Indicates whether the replacement text of tokens should be searched for more tokens.	<i>true</i>	No
onmissingfiltersfile	Indicate behavior when a nonexistent <i>filtersfile</i> is specified. One of "fail", "warn", "ignore".	"fail"	No

Attribute	Description	Required
token	The token to replace (eg., @DATE@)	Yes
value	The value to replace it with (eg., Thursday, April 26, 2001).	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>
```

Path:

The **path** type is commonly used to represent a classpath. Entries in the path are separated using a semicolon or colon. However, these characters are replaced at the run time by the running system's path separator character.

Most commonly, the classpath is set to the list of jar files and classes in the project, as shown in the example below:

```
<path id="build.classpath.jar">
  <pathelement path="${env.J2EE_HOME}/${j2ee.jar}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
</path>
```

Depend:

A task to manage Java class file dependencies. 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 analyzes 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.

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 analyzed. 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..	No, default=true

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.

Examples:

```
<depend srcdir="${java.dir}" destdir="${build.classes}"
        cache="depcache" closure="yes">
  <include name="**/*.java"/>
  <excludesfile name="${java.dir}/build_excludes"/>
</depend>
```

ANT JAVA RELATED TASKS

Every task in ant can be implemented by **Task interface** in java, so ant can be work with only java programming language.

Javac task:

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: Apache 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.

Attribute	Description	Required
sreaddir	Location of the java files.	Yes,
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 .javafiles 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.	No
classpathref	The classpath to use, given as a reference to a path defined elsewhere.	No
sourcepathref	The source path 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 default, nothing will be appended to -g. If debug is not turned on, this attribute will be ignored.	No
optimize	Indicates whether source should be compiled with optimization; defaults to off. Note that this flag is just	No

	ignored by Sun's javac starting with JDK 1.3 (since compile-time optimization is unnecessary).	
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	No
includeAntRuntime	Whether to include the Ant run-time libraries in the classpath; defaults to yes, unless build.sysclasspath is set. <i>It is usually best to set this to false</i> 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".	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.	
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.	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.	No
source	Value of the -source command-line switch; will be ignored by all implementations prior to javac1.4 (or modernwhen 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 ant.build.javac.source property.	No
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.	No; default is <i>java.io.tmpdir</i> .
updatedProperty	The property to set (to the value "true") if compilation has taken place and has been successful.	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.	No - default is "true"
createMissingPackageInfoClass	Some package level annotations in package-info.java files don't create any package-info.class files so Ant would recompile the same file every time. Starting with Ant 1.8 Ant will create an empty package-info.class for each package-info.java if there isn't one created by the compiler. In some setups this additional class causes problems and it can be suppressed by setting this attribute to "false".	No - default is "true"

Examples:

```
<javac srcdir="${src}"
    destdir="${build}"
    classpath="xyz.jar"
    debug="on"
    source="1.4"
/>
```

(or)

```
<javac srcdir="${src.home}" destdir="${work.home}/WEB-INF/classes">
  <classpath refid="compile.classpath" />
</javac>
```

Java task:

Executes a Java class within the running (Apache 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.

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.

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 <code><arg></code> 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 <code>java.lang.Runtime.exec()</code> . 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".	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 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 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)	No

Examples:

```

<java classname="test.Main">
  <arg value="-h"/>
  <classpath>
    <pathelement location="dist/test.jar"/>
    <pathelement path="${java.class.path}"/>
  </classpath>
</java>

```

PACKAGING USING ANT

Ant - Creating JAR files:

The next logical step after compiling your java source files, is to build the java archive, i.e the JAR file. Creating JAR files with Ant is quite easy with the **jar** task. Presented below are the commonly used attributes of the jar task.

Attribute	Description	Required
destfile	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 <i>keepcompression</i> 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 <i>zipfilesets</i> or while updating the archive), keep the compression as it has been originally instead of using the <i>compress</i> attribute. Defaults false.	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.	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 <i>index</i> 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.	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.	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).	No

strict	<p>Configures how to handle breaks of the packaging version specification:</p> <ul style="list-style-type: none"> • fail = throws a BuildException • warn = logs a message on warn level • ignore = logs a message on verbose level (default) 	No, defaults to ignore.
preserve0permissions	<p>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.</p> <p>Set this attribute to true if you really want to preserve the original permission field.</p>	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.	No, default is true
createUnicodeExtraFields	<p>Whether to create unicode extra fields to store the file names a second time inside the entry's metadata.</p> <p>Possible values are "never", "always" and "not-encodeable" which will only add Unicode extra fields if the file name cannot be encoded using the specified encoding.</p>	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.	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. Unless you also set flattenAttributes to true this may result in manifests containing multiple	No, default is false

	Class-Path attributes which violates the manifest specification.	
flattenAttributes	Whether to merge attributes occurring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute.	No, default is false
zip64Mode	When to use Zip64 extensions for entries. The possible values are "never", "always" and "as-needed".	No, default is "never"

Examples:

```
<jar destfile="${dist}/lib/app.jar"
    basedir="${build}/classes"
    excludes="**/Test.class"
/>
```

SignJar task:

Signing a jar allows users to authenticate the publisher.

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.

Attribute	Description	Required
jar	the jar file to sign	Yes
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
strict	(true false) strict checking when signing.	No; default false
internalsf	(true false) include the .SF file inside the signature block	No; default false
sectiononly	(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.
tsurl	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 Apache Ant is running in). Must support the same command line options as the Sun JDK jarsigner command.	No
force	Whether to force signing of the jar file even if it doesn't seem to be out of date or already signed.	No; default false
sigalg	name of signature algorithm	No
digestalg	name of digest algorithm	No

Examples:

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>
```

Manifest task:

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.

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. 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 occurring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute.	No, default is false

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>
```

Output:

```
Manifest-Version: 1.0
Built-By: bodewig
Created-By: Apache Ant 1.9

Name: common
Specification-Title: Example
Specification-Vendor: Example Organization
Implementation-Vendor: Example Corp.
Specification-Version: 1.2
Implementation-Version: 1.2 September 10, 2013
Implementation-Title: common

Name: common/class1.class
Sealed: false
```

Ant – Creating War files:

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.

Attribute	Description	Required
destfile	WAR file to create.	Exactly one of the two.
warfile	<i>Deprecated</i> name of the file to create - use destfile instead.	
webxml	The servlet configuration descriptor to use (WEB-INF/web.xml).	Yes
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.	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 <i>keepcompression</i> 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 <i>zipfilesets</i> or while updating the archive), keep the compression as it has been originally instead of using the <i>compress</i> attribute. Defaults false.	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.	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" 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.	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).	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.	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.	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-encodeable" which will only add Unicode extra fields if the file name cannot be encoded using the specified encoding.	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.	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. <i>Since Ant 1.8.0.</i> 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 occurring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute.	No, default is false
zip64Mode	When to use Zip64 extensions for entries. The possible values are "never", "always" and "as-needed".	No, default is "never"

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>
```

Generated war file 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
```

Ant – Creating Ear files:

An extension of the [Jar](#) task with special treatment for files that should end up in an Enterprise Application archive.

Attribute	Description	Required
destfile	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 <i>keepcompression</i> 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 <i>zipfilesets</i> or while updating the archive), keep the compression as it has been originally instead of using the <i>compress</i> attribute. Defaults false.	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.	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" will merge all	No

	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 <i>index</i> is false. Oracle'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.	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	No

	slightly more recent than precompiled pages, rendering precompilation useless. Defaults to true.	
level	Non-default level at which file compression should be performed. Valid values range from 0 (no compression/fastest) to 9 (maximum compression/slowest).	No
preservePermissions	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.	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.	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-encodeable" which will only add Unicode extra fields if the file name cannot be encoded using the specified encoding. <i>Since Ant 1.8.0.</i>	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.	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. <i>Since Ant 1.8.0.</i> 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 occurring more than once in a section (this can only happen for the Class-Path attribute) into a single attribute.	No, default is false
zip64Mode	When to use Zip64 extensions for entries. The possible values are "never", "always" and "as-needed".	No, default is "never"

Examples:

```
<ear destfile="${build.dir}/myapp.ear" appxml="${src.dir}/metadata/application.xml">
  <fileset dir="${build.dir}" includes="*.jar,*.war"/>
</ear>
```

GENERATING DOCUMENTATION

Javadoc / Javadoc2:

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.

Attribute	Description	Required
sourcepath	Specify where to find source files	At least one of the three or nested<sourcepath>,<fileset> or<packageset>
sourcepathref	Specify where to find source files by reference to a PATH defined elsewhere.	
sourcefiles	Comma separated list of source files -- see also the nested source element.	
destdir	Destination directory for output files	Yes
maxmemory	Max amount of memory to allocate to the javadoc VM	No
packagenames	Comma separated list of package files (with terminating wildcard) -- see also the nested packageelement.	No

packageList	The name of a file containing the packages to process	No
classpath	Specify where to find user class files	No
Bootclasspath	Override location of class files loaded by the bootstrap class loader	No
classpathref	Specify where to find user class files by reference to a PATH defined elsewhere.	No
bootclasspathref	Override location of class files loaded by the bootstrap class loader by reference to a PATH defined elsewhere.	No
Extdirs	Override location of installed extensions	No
Overview	Read overview documentation from HTML file	No
access	Access mode: one of public, protected, package, or private	No (defaultprotected)
Public	Show only public classes and members	No
Protected	Show protected/public classes and members (default)	No
Package	Show package/protected/public classes and members	No
Private	Show all classes and members	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.	No
Verbose	Output messages about what Javadoc is doing	No
Locale	Locale to be used, e.g. en_US or en_US_WIN	No
Encoding	Source file encoding name	No
Version	Include @version paragraphs	No

Use	Create class and package usage pages	No
Author	Include @author paragraphs	No
Splitindex	Split index into one file per letter	No
Windowtitle	Browser window title for the documentation (text)	No
Doctitle	Include title for the package index(first) page (html-code)	No
Header	Include header text for each page (html-code)	No
Footer	Include footer text for each page (html-code)	No
bottom	Include bottom text for each page (html-code)	No
link	Create links to javadoc output at the given URL -- see also the nested link element.	No
linkoffline	Link to docs at <url> using package list at <url2> - separate the URLs by using a space character -- see also the nested link element.	No
group	Group specified packages together in overview page. The format is as described below -- see also the nested group element.	No
nodeprecated	Do not include @deprecated information	No
nodeprecatedlist	Do not generate deprecated list	No
notree	Do not generate class hierarchy	No
noindex	Do not generate index	No
nohelp	Do not generate help link	No
nonavbar	Do not generate navigation bar	No

serialwarn	Generate warning about @serial tag	No
helpfile	Specifies the HTML help file to use	No
stylesheetfile	Specifies the CSS stylesheet to use	No
charset	Charset for cross-platform viewing of generated documentation	No
docencoding	Output file encoding name	No
doclet	Specifies the class file that starts the doclet used in generating the documentation -- see also the nesteddoclet element.	No
docletpath	Specifies the path to the doclet class file that is specified with the -doclet option.	No
docletpathref	Specifies the path to the doclet class file that is specified with the -doclet option by reference to a PATH defined elsewhere.	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.	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	No
failonwarning	Stop the buildprocess if a warning is emitted - i.e. if javadoc's output contains the word "warning".	No
excludepackagenames	comma separated list of packages you don't want docs for -- see also the nested excludepackage element.	No
defaultexcludes	indicates whether default excludes should be used (yes no); default excludes are used when omitted.	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	No

	nested package elements. Also applies to all the other command line options. (yes no). Default is 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.	No
linksource	Generate hyperlinks to source files. (yes no). Default is no.	No
breakiterator	Use the new breakiterator algorithm. (yes no). Default is no.	No
noqualifier	Enables the -noqualifier argument - must be all or a colon separated list of packages.	No
includenosource packages	If set to true, packages that don't contain Java source but a package.html will get documented	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).	No
docfilessubdirs	Enables deep-copying of doc-files subdirectories. Defaults to false.	No
excludedocfilessubdir	Colon-separated list of doc-files' subdirectories to exclude if docfilessubdirs is true.	No
postProcessGeneratedJavadocs	Whether to post-process the generated javadocs in order to mitigate CVE-2013-1571. Defaults to true.	No

JUNIT TASKS

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 noJUnit4TestAdapter.

Note: This task depends on external libraries not included in the Apache Ant distribution.

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>.

Attribute	Description	Required
printsummary	Print one-line statistics for each testcase. Can take the values on, off, and withOutAndErr. withOutAndErr is the same as onbut 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.	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 <code>java.lang.Runtime.exec()</code> . 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	Send any output generated by tests to the test formatters. This is "true" by default.	No
tempdir	Where Ant should place temporary files.	No;
reloading	Whether or not a new classloader should be instantiated for each test case. Ignore if fork is set to true.	No; default is true.
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).	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.	No
enableTestListenerEvents	Whether Ant should send fine grained information about the running tests to Ant's logging system at the verbose	No

	level. Such events may be used by custom test listeners to show the progress of tests. Defaults to false.	
threads	A number of threads to run the tests in.	No

Example:

```

<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>

```

SCM RELATED TASKS

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

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

Example: Handles packages/modules retrieved from a **CVS** repository.

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 ascompression="false"	No. Defaults to no compression.
cvsRoot	CVSROOT variable.	No
cvsRsh	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 Apache 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.	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

INFRASTRUCTURE TASKS

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

Attribute	Description
attrib / chmod	Change permissions of a file (attrib for windows, chmod for Unix)
checksum	Generates a checksum for files
concat	Concatenates multiple files into single file or to the console
copy	Copy a file or multiple files 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 URL
import / include	Import or includes another build file into the current Ant script
mail	Send mail over SMTP
mkdir	Creates a new directory

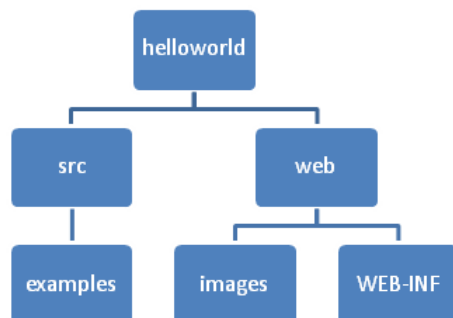
move	Moves a file or collection of files to another location
record	Listens to the build process and outputs to a file
replace	Replaces a occurrences of a string in file or collection of files
replaceregexp	Replaces a occurrences of a string in file or collection of files using regular expressions
sql	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.

EXAMPLES

War creation – using Ant script:

This is a simple web application named as helloworld.

Directory Structure:



Create Hello.java:

Create a new file and name it as Hello.java and place this file into the **helloworld/src/examples/**

Code:

```

package examples;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```

public final class Hello extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException {

        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        writer.println("<html>");
        writer.println("<head>");
        writer.println("<title>Sample Application Servlet Page</title>");
        writer.println("</head>");
        writer.println("<body bgcolor=white>");

        writer.println("<table border=\"0\" cellpadding=\"10\">");
        writer.println("<tr>");
        writer.println("<td>");
        writer.println("<img src=\"images/springsource.png\">");
        writer.println("</td>");
        writer.println("<td>");
        writer.println("<h1>Sample Application Servlet</h1>");
        writer.println("</td>");
        writer.println("</tr>");
        writer.println("</table>");

        writer.println("This is the output of a servlet that is part of");
        writer.println("the Hello, World application.");

        writer.println("</body>");
        writer.println("</html>");
    }
}

```

Create Hello.jsp:

Create file called Hello.jsp and place this file into **helloworld/web/**

Code:

```

<html>
<head>
    <title>Sample Application JSP Page</title>
</head>

<body bgcolor=white>

<table border="0" cellpadding="10">
<tr>
    <td align=center>
        
    </td>
    <td>

```

```

        <h1>Sample Application JSP Page</h1>
    </td>
</tr>
</table>

<br />
<p>This is the output of a JSP page that is part of the HelloWorld application.</p>

<%= new String("Hello!") %>

</body>
</html>

```

Create index.html:

This can be your home page of the helloworld web application. Create the file called index.html and place it into the path **helloworld/web/**

Code:

```

<html>
<head>
    <title>Sample "Hello, World" Application</title>
</head>
<body bgcolor=white>

    <table border="0" cellpadding="10">
        <tr>
            <td>
                
            </td>
            <td>
                <h1>Sample "Hello, World" Application</h1>
            </td>
        </tr>
    </table>

    <p>This is the home page for the HelloWorld Web application. </p>
    <p>To prove that they work, you can execute either of the following links:
    <ul>
        <li>To a <a href="hello.jsp">JSP page</a>.
        <li>To a <a href="hello">servlet</a>.
    </ul>

    </body>
</html>

```

Create web.xml:

Web.xml file is nothing but a deployment descriptor. Every web application can be controlled or navigated according to web.xml file. Create the web.xml file and place it into the path **helloworld/web/WEB-INF/**

Code:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>HelloWorld Application</display-name>
  <description>
    This is a simple web application with a source code organization
    based on the recommendations of the Application Developer's Guide.
  </description>

  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>examples.Hello</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

Create Ant Script file:

Ant script is nothing but your **build.xml** file. So create the build.xml file and write/paste this below code into ant script file. Place this file into the root directory of you application. In this examples you root directory is **helloworld/**

Ant Script:

```
<project name="My Project" default="help" basedir=". ">
  <!-- Define the properties used by the build -->
  <property name="app.name" value="helloWorld"/>
  <property name="app.version" value="0.1-dev"/>
  <property name="tcserver.home" value="path/location of tomcat server directory" />
  <property name="work.home" value="${basedir}/work"/>
  <property name="dist.home" value="${basedir}/dist"/>
  <property name="src.home" value="${basedir}/src"/>
  <property name="web.home" value="${basedir}/web"/>

  <target name="help">
    <echo>You can use the following targets:</echo>
    <echo> </echo>
    <echo> help : (default) Prints this message </echo>
    <echo> all : Cleans, compiles, and packages application</echo>
```

```

<echo> clean : Deletes work directories</echo>
<echo> compile : Compiles servlets into class files</echo>
<echo> dist : Packages artifacts into a deployable WAR</echo>
<echo></echo>
<echo>For example, to clean, compile, and package all at once, run:</echo>
<echo>prompt> ant all </echo>
</target>

<!-- Define the CLASSPATH -->
<path id="compile.classpath">
  <fileset dir="${tcserver.home}/bin">
    <include name="*.jar"/>
  </fileset>
  <pathelement location="${tcserver.home}/lib"/>
  <fileset dir="${tcserver.home}/lib">
    <include name="*.jar"/>
  </fileset>
</path>

<target name="all" depends="clean,compile,dist"
  description="Clean work dirs, then compile and create a WAR"/>

<target name="clean"
  description="Delete old work and dist directories">
  <delete dir="${work.home}"/>
  <delete dir="${dist.home}"/>
</target>

<target name="prepare" depends="clean"
  description="Create working dirs and copy static files to work dir">
  <mkdir dir="${dist.home}"/>
  <mkdir dir="${work.home}/WEB-INF/classes"/>
  <!-- Copy static HTML and JSP files to work dir -->
  <copy todir="${work.home}">
    <fileset dir="${web.home}"/>
  </copy>
</target>

<target name="compile" depends="prepare"
  description="Compile Java sources and copy to WEB-INF/classes dir">
  <javac srcdir="${src.home}"
    destdir="${work.home}/WEB-INF/classes">
    <classpath refid="compile.classpath"/>
  </javac>
  <copy todir="${work.home}/WEB-INF/classes">
    <fileset dir="${src.home}" excludes="**/*.java"/>
  </copy>
</target>

```

```

<target name="dist" depends="compile"
    description="Create WAR file for binary distribution">
    <jar jarfile="${dist.home}/${app.name}-${app.version}.war"
        basedir="${work.home}"/>
</target>

</project>

```

Note: In this code you need to provide the tomcat server home directory path to compile the java source code. We have created Servlet and Jsp's file, to compile Java Servlets you need to have **servlet-api.jar** in the similar way for the jsp's also.

Tomcat server will contains the collection of **.jar** files to compile and executes the java based applications.

Copying image:

Just copy any image into the **helloworld/web/images/** directory and rename as **springsource.png**.

Then your directory structure and contents tree structure will be like this.



```

root@localhost:~/helloworld
[root@localhost helloworld]# ls -R
.:
build.xml  src  web

./src:
examples

./src/examples:
Hello.java

./web:
hello.jsp  images  index.html  WEB-INF

./web/images:
springsource.png

./web/WEB-INF:
web.xml
[root@localhost helloworld]#

```

Executing Ant Script:

To execute ant script just type the command **ant** and hit enter, by default it will search for the build.xml in the current directory and if found then executes it.

Syntax:

\$ ant (or) **\$ ant -f <file>**

Examples:

\$ ant	→ By default it will executes the build.xml
\$ ant -f myscript.xml	→ to execute custom ant script file.

```
root@localhost:~/helloworld
[root@localhost helloworld]# ant
Buildfile: /root/helloworld/build.xml

help:
[echo] You can use the following targets:
[echo]
[echo] help    : (default) Prints this message
[echo] all     : Cleans, compiles, and packages application
[echo] clean    : Deletes work directories
[echo] compile  : Compiles servlets into class files
[echo] dist     : Packages artifacts into a deployable WAR
[echo]
[echo] For example, to clean, compile, and package all at once, run:
[echo] prompt> ant all

BUILD SUCCESSFUL
Total time: 0 seconds
[root@localhost helloworld]#
```

By default it calls the default target of the project, if you want to call specific targets you should pass the parameters.

Syntax:

\$ant <target name> (or) **\$ ant -f <file name> <target name>**

Example:

```
root@localhost:~/helloworld
[root@localhost helloworld]# ant compile
Buildfile: /root/helloworld/build.xml

clean:

prepare:
[mkdir] Created dir: /root/helloworld/dist
[mkdir] Created dir: /root/helloworld/work/WEB-INF/classes
[copy] Copying 4 files to /root/helloworld/work

compile:
[javac] /root/helloworld/build.xml:57: warning: 'includeantruntime' was not set, defaulting to build
.sysclasspath=last; set to false for repeatable builds
[javac] Compiling 1 source file to /root/helloworld/work/WEB-INF/classes

BUILD SUCCESSFUL
Total time: 1 second
[root@localhost helloworld]# ant -f build.xml clean
Buildfile: /root/helloworld/build.xml

clean:
[delete] Deleting directory /root/helloworld/work
[delete] Deleting directory /root/helloworld/dist

BUILD SUCCESSFUL
Total time: 0 seconds
[root@localhost helloworld]#
```

Calling specific target

Calling specific target with custom filename

Creating war:

Execute the target **all**, then it will perform all the targets that was there in the build.xml and creates the war file.

Note: You will be notified where your war file is created and placed.

```
root@localhost:~/helloworld
[root@localhost helloworld]# ant all
Buildfile: /root/helloworld/build.xml

clean:


prepare:
  [mkdir] Created dir: /root/helloworld/dist
  [mkdir] Created dir: /root/helloworld/work/WEB-INF/classes
  [copy] Copying 4 files to /root/helloworld/work

compile:
  [javac] /root/helloworld/build.xml:57: warning: 'includeantruntime' was not set, defaulting to build
.sysclasspath=last; set to false for repeatable builds
  [javac] Compiling 1 source file to /root/helloworld/work/WEB-INF/classes

dist:
  [jar] Building jar: /root/helloworld/dist/helloWorld-0.1-dev.war

all:

BUILD SUCCESSFUL
Total time: 2 seconds
[root@localhost helloworld]#
```



Now you can deploy your web application (helloworld-0.1-dev.war) into web server/application server.

Deploying web application:

Windows:

Just copy your .war file into the tomcat webapps directory.

Example: C:\apache-tomcat-7.0.57\webapps\ → It depends where you installed tomcat.

Linux:

Use the **cp** command to copy your .war file into tomcat server.

Syntax:

\$ cp <filename> <destination path>

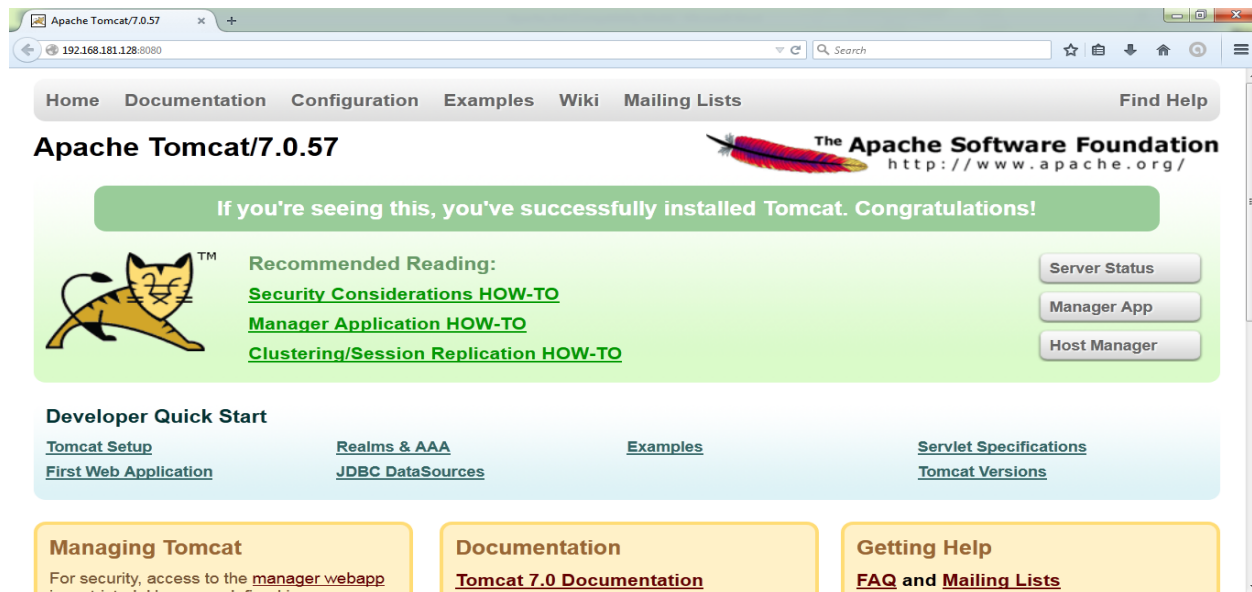
Example:

\$ cp helloworld-0.1-dev.war /opt/apache-tomcat-7.0.57/webapps/

Deploying from web browser:

Start your tomcat server, and go to the web browser then pass the URL of the tomcat server along with corresponding port number.

In this case tomcat URL is: <http://192.168.181.128:8080/>



Click on the **Manager App** button, then it will ask for the username and password to provide authentication.

You can find/configure username and password in tomcat-users.xml file, and it will be available in conf directory of tomcat server. Once get authenticated into tomcat the page will be like this.

Message: OK - Undeployed application at context path /helloWorld-0.1-dev

Manager

List Applications HTML Manager Help Manager Help Server Status

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/jenkins	None specified	Jenkins v1.588	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	2	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

Deploy

WAR file to deploy

Select WAR file to upload No file selected.

Deploy

Diagnostics

Check to see if a web application has caused a memory leak on or

This diagnostic check will trigger a full garbage collection. Use it with extreme caution on production systems.

Server Information

Once if you login into Manager App, just scroll down your page then you will find the option to deploy your .war files to tomcat server.

Once if you done this, then your web application deployment phase has finished.

Running your Web Application:

In your tomcat server Manager App section you will find the list of web applications that are available in your tomcat server. Click on the web application that you want to run.

(OR)

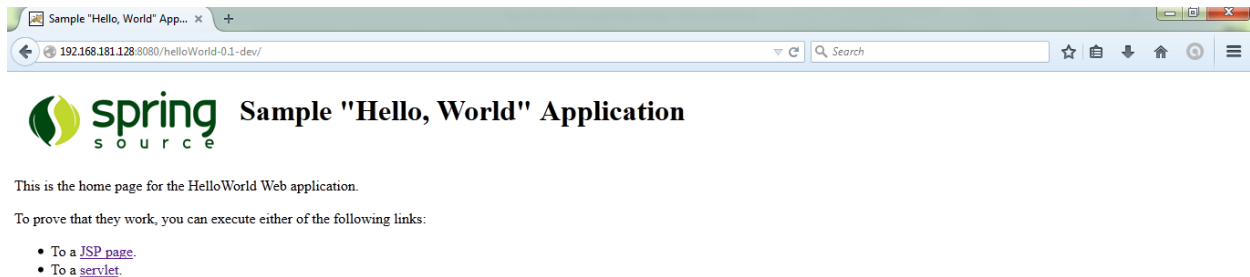
Directly you can pass the absolute URL, it means along with you tomcat sever URL you have to pass your web application name (context).

Syntax:

<http://tomcat-server-ip:port-no/context/>

Example:

<http://192.168.181.128:8080/helloWorld-0.1-dev/>



In this tutorial we have done writing of the ant script to build project artifacts, using the ant script building the installable components, and deployment of the artifacts to the web server (tomcat server).

THE END
