Contents

[Insallation 2](#_Toc430616124)

[Tutorial context 3](#_Toc430616125)

[Ant - Data Types 4](#_Toc430616126)

[Fileset 4](#_Toc430616127)

[Pattern set 4](#_Toc430616128)

[File list 4](#_Toc430616129)

[Filter set 5](#_Toc430616130)

[Path 5](#_Toc430616131)

[Ant - Building Projects 6](#_Toc430616132)

[Ant - Build Documentation 9](#_Toc430616133)

[Attributes 9](#_Toc430616134)

[Putting it all together 9](#_Toc430616135)

[Ant - Creating JAR files 11](#_Toc430616136)

# Insallation

C:\Program Files\NetBeans 8.0.1\extide\ant – Ant, который идет с NetBeans (свойства вызываются иконкой настройки в области исполнения ant в среде NetBeans)

Необходимо прописать переменные среды (WinKey + Pause и далее):

JAVA\_HOME – путь к установленному jdk

ANT\_HOME – в случае с использованием NetBeans тот самый путь, прописанный в свойствах Ant

PATH - %PATH%;%ANT\_HOME%\bin. То есть тот же путь что и ANT\_HOME только с папкой bin

Не забываем нажимать OK и перегрузить командную строку

Можно выполнить проверку.

C:\>ant -version

Apache Ant(TM) version 1.8.2 compiled on December 20 2010

# Tutorial context

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | The XML element **project** has three attributes :   |  |  | | --- | --- | | **Attributes** | **Description** | | *name* | The Name of the project. (Optional) | | *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. (Mandatory) | | *basedir* | The base directory (or) the root folder for the project. (Optional) |   The **target** element has the following attributes:   |  |  | | --- | --- | | **Attributes** | **Description** | | *name* | The name of the target (Required) | | *depends* | Comma separated list of all targets that this target depends on. (Optional) | | *description* | A short description of the target. (optional) | | *if* | Allows the execution of a target based on the trueness of a conditional attribute. (optional) | | *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. (Optional) | |

By default, Ant provides the following pre-defined properties that can be used in the build file:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| *ant.file* | The full location of the build file. |
| *ant.version* | The version of the Apache Ant installation. |
| *basedir* | The basedir of the build, as specified in the **basedir** attribute of the **project** element. |
| *ant.java.version* | The version of the JDK that is used by Ant. |
| *ant.project.name* | The name of the project, as specified in the **name** atrribute of the **project** element. |
| *ant.project.default-target* | The default target of the current project. |
| *ant.project.invoked-targets* | Comma separated list of the targets that were invoked in the current project. |
| *ant.core.lib* | The full location of the Ant jar file. |
| *ant.home* | The home directory of Ant installation. |
| *ant.library.dir* | The home directory for Ant library files - typically ANT\_HOME/lib folder. |

Ant also makes the system properties (Example: file.separator) available to the build file.

# Ant - Data Types

Ant provides a number of predefined data types. Do not confuse the term "data types" with those that are available in the programming language, instead consider them as a set of services that are built into the product already.

The following data types are provided by Apache Ant.

## Fileset

The fileset data types represents a collection of files. It is used as a filter to include or exclude files that match a particular pattern.

For example, refer the following code. Here, the src attribute points to the source folder of the project.

The fileset selects all .java files in the source folder except those contain the word 'Stub'. The case-sensitive filter is applied to the fileset which means a file with the name Samplestub.java will not be excluded from the fileset.

<fileset dir="${src}" casesensitive="yes">

<include name="\*\*/\*.java"/>

<exclude name="\*\*/\*Stub\*"/>

</fileset>

## Pattern set

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

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

The following example depicts the usage of a pattern set.

<patternset id="java.files.without.stubs">

<include name="src/\*\*/\*.java"/>

<exclude name="src/\*\*/\*Stub\*"/>

</patternset>

The patternset can then be reused with a fileset as follows:

<fileset dir="${src}" casesensitive="yes">

<patternset refid="java.files.without.stubs"/>

</fileset>

## File list

The filelist data type is similar to the file set except the following differences:

* filelist contains explicitly named lists of files and it does not support wild cards.
* filelist data type can be applied for existing or non-existing files.

Let us see the following example of the filelist data type. Here, the attribute **webapp.src.folder** points to the web application source folder of the project.

<filelist id="config.files" dir="${webapp.src.folder}">

<file name="applicationConfig.xml"/>

<file name="faces-config.xml"/>

<file name="web.xml"/>

<file name="portlet.xml"/>

</filelist>

## Filter set

Using a filterset data type along with the copy task, you can replace certain text in all files that matches the pattern with a replacement value.

A common example is to append the version number to the release notes file, as shown in the following code.

<copy todir="${output.dir}">

<fileset dir="${releasenotes.dir}" includes="\*\*/\*.txt"/>

<filterset>

<filter token="VERSION" value="${current.version}"/>

</filterset>

</copy>

In this Code:

* The attribute **output.dir** points to the output folder of the project.
* The attribute **releasenotes.dir** points to the release notes folder of the project.
* The attribute **current.version** points to the current version folder of the project.
* The copy task, as the name suggests, is used to copy files from one location to another.

## Path

The **path** data type is commonly used to represent a class-path. Entries in the path are separated using semicolons or colons. However, these characters are replaced at the run-time by the executing system's path separator character.

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>

In this code:

* The attribute **env.J2EE\_HOME** points to the environment variable **J2EE\_HOME**.
* The attribute **j2ee.jar** points to the name of the J2EE jar file in the J2EE base folder.

# Ant - Building Projects

Now that we have learnt about the data types in Ant, it is time to put that knowledge into action. We will build a project in this chapter. The aim of this chapter is to build an Ant file that compiles the java classes and places them in the WEB-INF\classes folder.

Consider the following project structure:

* The database scripts are stored in the **db** folder.
* The java source code is stored in the **src** folder.
* The images, js, META-INF, styles (css) are stored in the **war** folder.
* The JSPs are stored in the **jsp** folder.
* The third party jar files are stored in the **lib** folder.
* The java class files are stored in the **WEB-INF\classes** folder.

This project forms the **Hello World** Fax Application for the rest of this tutorial.

C:\work\FaxWebApplication>tree

Folder PATH listing

Volume serial number is 00740061 EC1C:ADB1

C:.

+---db

+---src

. +---faxapp

. +---dao

. +---entity

. +---util

. +---web

+---war

+---images

+---js

+---META-INF

+---styles

+---WEB-INF

+---classes

+---jsp

+---lib

Here is the build.xml required for this project. Let us consider it piece by piece.

<?xml version="1.0"?>

<project name="fax" basedir="." default="build">

<property name="src.dir" value="src"/>

<property name="web.dir" value="war"/>

<property name="build.dir" value="${web.dir}/WEB-INF/classes"/>

<property name="name" value="fax"/>

<path id="master-classpath">

<fileset dir="${web.dir}/WEB-INF/lib">

<include name="\*.jar"/>

</fileset>

<pathelement path="${build.dir}"/>

</path>

<target name="build" description="Compile source tree java files">

<mkdir dir="${build.dir}"/>

<javac destdir="${build.dir}" source="1.5" target="1.5">

<src path="${src.dir}"/>

<classpath refid="master-classpath"/>

</javac>

</target>

<target name="clean" description="Clean output directories">

<delete>

<fileset dir="${build.dir}">

<include name="\*\*/\*.class"/>

</fileset>

</delete>

</target>

</project>

First, let us declare some properties for the source, web, and build folders.

<property name="src.dir" value="src"/>

<property name="web.dir" value="war"/>

<property name="build.dir" value="${web.dir}/WEB-INF/classes"/>

In this example:

* **src.dir** refers to the source folder of the project where the java source files can be found.
* **web.dir** refers to the web source folder of the project, where you can find the JSPs, web.xml, css, javascript and other web related files
* **build.dir** refers to the output folder of the project compilation.

Properties can refer to other properties. As shown in the above example, the **build.dir** property makes a reference to the **web.dir** property.

In this example, the **src.dir** refers to the source folder of the project.

The default target of our project is the **compile** target. But first let us look at the **clean** target.

The clean target, as the name suggests, deletes the files in the build folder.

<target name="clean" description="Clean output directories">

<delete>

<fileset dir="${build.dir}">

<include name="\*\*/\*.class"/>

</fileset>

</delete>

</target>

The master-classpath holds the classpath information. In this case, it includes the classes in the build folder and the jar files in the lib folder.

<path id="master-classpath">

<fileset dir="${web.dir}/WEB-INF/lib">

<include name="\*.jar"/>

</fileset>

<pathelement path="${build.dir}"/>

</path>

Finally, the build target to build the files. First of all, we create the build directory, if it does not exist. Then we execute the javac command (specifying jdk1.5 as our target compilation). We supply the source folder and the classpath to the javac task and ask it to drop the class files in the build folder.

<target name="build" description="Compile main source tree java files">

<mkdir dir="${build.dir}"/>

<javac destdir="${build.dir}" source="1.5" target="1.5" debug="true"

deprecation="false" optimize="false" failonerror="true">

<src path="${src.dir}"/>

<classpath refid="master-classpath"/>

</javac>

</target>

Executing Ant on this file compiles the java source files and places the classes in the build folder.

The following outcome is the result of running the Ant file:

C:\>ant

Buildfile: C:\build.xml

BUILD SUCCESSFUL

Total time: 6.3 seconds

The files are compiled and placed in the **build.dir** folder.

# Ant - Build Documentation

Documentation is a must in any project. Documentation plays a great role in the maintenance of a project. Java makes documentation easier by the use of the in-built **javadoc** tool. Ant makes it even easier by generating the documentation on demand.

As you know, the javadoc tool is highly flexible and allows a number of configuration options. Ant exposes these configuration options via the javadoc task. If you are unfamiliar with javadocs, we suggest that you start with this [Java Documentation Tutorial](http://www.tutorialspoint.com/java/java_documentation.htm).

The following section lists the most commonly used javadoc options that are used in Ant.

## Attributes

Source can be specified using **sourcepath**, **sourcepathref** or **sourcefiles**.

* **sourcepath** is used to point to the folder of the source files (e.g. src folder).
* **sourcepathref** is used to refer a path that is referenced by the path attribute (e.g, delegates.src.dir).
* **sourcefiles** is used when you want to specify the individual files as a comma separated list.

Destination path is specified using the **destdir** folder (e.g build.dir).

You could filter the **javadoc** task by specifying the package names to be included. This is achieved by using the **packagenames** attribute, a comma separated list of package files.

You could filter the javadoc process to show only the public, private, package, or protected classes and members. This is achieved by using the **private**, **public**, **package** and **protected** attributes.

You could also tell the javadoc task to include the author and version information using the respective attributes.

You could also group the packages together using the **group** attribute, so that it becomes easy to navigate.

## Putting it all together

Let us continue our theme of the **Hello world** Fax application. Let us add a documentation target to our Fax application project.

Given below is an example javadoc task used in our project. In this example, we have specified the javadoc to use the **src.dir** as the source directory, and **doc** as the target.

We have also customized the window title, the header, and the footer information that appear on the java documentation pages.

Also, we have created three groups:

* one for the utility classes in our source folder,
* one for the user interfaces classes, and
* one for the database related classes.

You may notice that the data package group has two packages -– faxapp.entity and faxapp.dao.

<target name = "generate-javadoc">

<javadoc packagenames="faxapp.\*" sourcepath="${src.dir}"

destdir = "doc" version = "true" windowtitle = "Fax Application">

<doctitle><![CDATA[= Fax Application =]]></doctitle>

<bottom>

<![CDATA[Copyright © 2011. All Rights Reserved.]]>

</bottom>

<group title = "util packages" packages = "faxapp.util.\*"/>

<group title = "web packages" packages = "faxapp.web.\*"/>

<group title = "data packages" packages = "faxapp.entity.\*:faxapp.dao.\*"/>

</javadoc>

<echo message = "java doc has been generated!" />

</target>

Let us execute the javadoc Ant task. It generates and places the java documentation files in the doc folder.

When the **javadoc target** is executed, it produces the following outcome:

C:\>ant generate-javadoc

Buildfile: C:\build.xml

java doc has been generated!

BUILD SUCCESSFUL

Total time: 10.63 second

The java documentation files are now present in the **doc** folder.

Typically, the javadoc files are generated as a part of the release or package targets.

# 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. The commonly used attributes of the jar task are as follows:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| basedir | The base directory for the output JAR file. By default, this is set to the base directory of the project. |
| compress | Advises Ant to compress the file as it creates the JAR file. |
| keepcompression | While the **compress** attribute is applicable to the individual files, the **keepcompression** attribute does the same thing, but it applies to the entire archive. |
| destfile | The name of the output JAR file. |
| duplicate | Advises Ant on what to do when duplicate files are found. You could add, preserve, or fail the duplicate files. |
| excludes | Advises Ant to not include these comma separated list of files in the package. |
| excludesfile | Same as above, except the exclude files are specified using a pattern. |
| inlcudes | Inverse of excludes. |
| includesfile | Inverse of excludesfile. |
| update | Advises Ant to overwrite files in the already built JAR file. |

Continuing our **Hello World** Fax Application project, let us add a new target to produce the jar files. But before that, let us consider the jar task given below.

<jar destfile = "${web.dir}/lib/util.jar"

basedir = "${build.dir}/classes"

includes = "faxapp/util/\*\*"

excludes = "\*\*/Test.class" />

Here, the **web.dir** property points to the path of the web source files. In our case, this is where the util.jar will be placed.

The **build.dir** property in this example points to the build folder where the class files for the util.jar can be found.

In this example, we create a jar file called **util.jar** using the classes from the **faxapp.util.\*** package. However, we are excluding the classes that end with the name Test. The output jar file will be placed in the web application lib folder.

If we want to make the util.jar an executable jar file we need to add the **manifest** with the **Main-Class** meta attribute.

Therefore, the above example will be updated as:

<jar destfile = "${web.dir}/lib/util.jar"

basedir = "${build.dir}/classes"

includes = "faxapp/util/\*\*"

excludes = "\*\*/Test.class">

<manifest>

<attribute name = "Main-Class" value = "com.tutorialspoint.util.FaxUtil"/>

</manifest>

</jar>

To execute the jar task, wrap it inside a target, most commonly, the build or package target, and execute them.

<target name="build-jar">

<jar destfile="${web.dir}/lib/util.jar"

basedir="${build.dir}/classes"

includes="faxapp/util/\*\*"

excludes="\*\*/Test.class">

<manifest>

<attribute name="Main-Class" value="com.tutorialspoint.util.FaxUtil"/>

</manifest>

</jar>

</target>

Running Ant on this file creates the util.jar file for us.

The following outcome is the result of running the Ant file:

C:\>ant build-jar

Buildfile: C:\build.xml

BUILD SUCCESSFUL

Total time: 1.3 seconds

The util.jar file is now placed in the output folder.