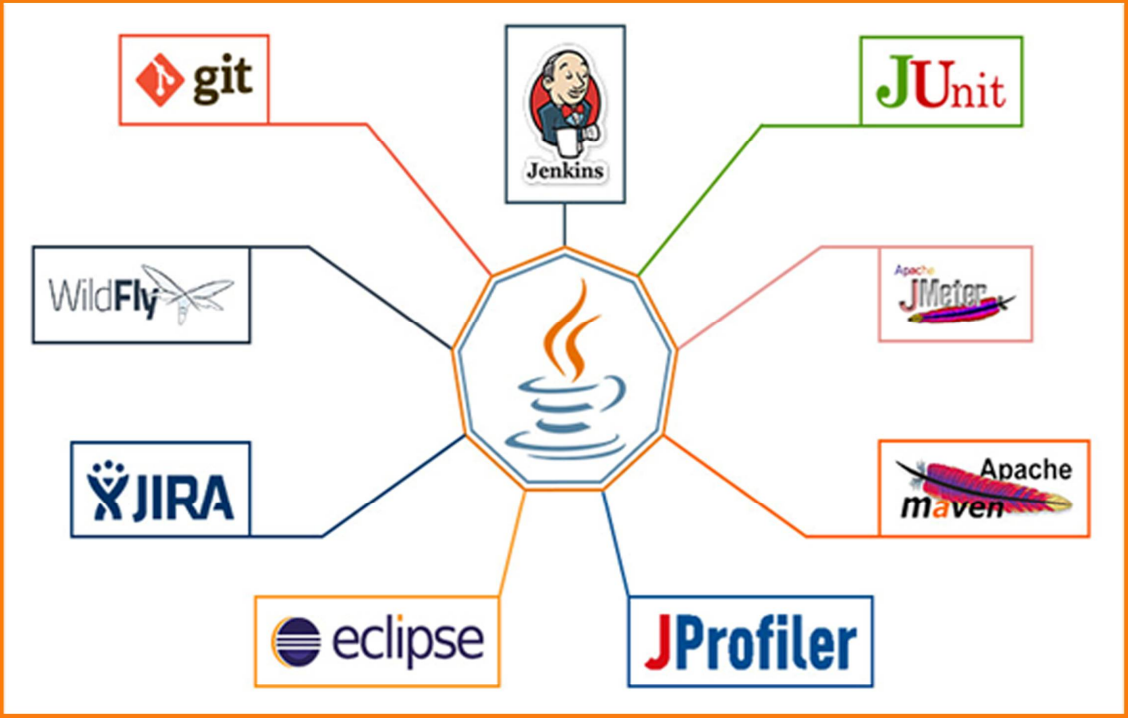
JAVA means DURGA SOFT

Java Real Time Tools

ANT



India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Ant Tool

Contents:

1. Introduction to Ant (Another Neat Tool):

Ant is an open source build technology developed by Apache intended to build processes in Java environment. It is a similar kind of tool like ***make,*** but it does not use shell commands to extend the functionality. The use of shell commands in *make* brings about the integrity with other languages too but this also makes it platform specific. In contrast, Ant is based on XML and uses java classes in automatic generation of build processes that makes it platform independent. It is applicable to any integrated development environment (IDE) that uses java. A build file is generally named as ***build.xml***.

The best features of the Ant technology can be summarized as below -

☐ **Easy to Use**: It is not a programming language, it is an XML based scripting tool, therefore easy to understand and implement.

☐ **Portable and Cross-platform based**: Use of Java classes makes it portable, i.e., it can be run on any operating system.

☐ **Extended Functionality**: Ant is based on java platform, that's why its functionality can be extended to any development environment based on java. It is easier to implement than any specific IDE because it is automated and ubiquitous.

☐ **Build Automation**: Ant provides automated build process that is faster and more efficient than manual procedures and other build tools can also be integrated with it.

☐ **Compilation of Source Code**: Ant can use and compile source code from a variety of version controls and packaging of the compiled code and resources can also be done.

☐ **Handling Dependencies between Targets**: An Ant Project describes the target and tasks associated with it and also handle dependencies between various targets and tasks.

Ant Definition

Apache Ant is an open source, cross-platform based build tool that is used to describe a build process and its dependencies and implemented in XML scripts using Java classes that ensures its

extensibility to any development environment (based on Java) and its integrity with other build tools.

2. Ant Installation:

Ant is free and open source build tool, written in Java, helps in automating the entire build process of a Java development project.

- Ant uses XML build files.
- By default, Ant looks for a build file named build.xml.
- The build file contains information about how to build a particular project.
- Each project contains multiple targets like creating directory, compiling source codes.
- Target can depend on other targets.
- Targets contain tasks.
- Behind each task is a Java class that performs the described work.

To install Ant follow the steps given below.

a) Down ant latest or required version from Apache Foundation website
   http://ant.apache.org/bindownload.cgi
b) Extract Zip file to your local Disk say "E:" drive
   D:\ apache-ant-1.8.2
c) Set the ANT_HOME environment variable to point to to the ant installation directory.
   ANT_HOME=E:\apache-ant-1.8.2

d) Set the JAVA_HOME environment variable to point to the JDK location.
   JAVA_HOME=E: \JDK1.5
e) Add ANT_HOME/bin and JAVA_HOME/bin to your system's **PATH** environment variable.
   PATH=E:\apache-ant-1.8.2\bin; D:\JDK1.5\bin;.

To make sure the installation is proper, go to command prompt and execute the command **ant.**

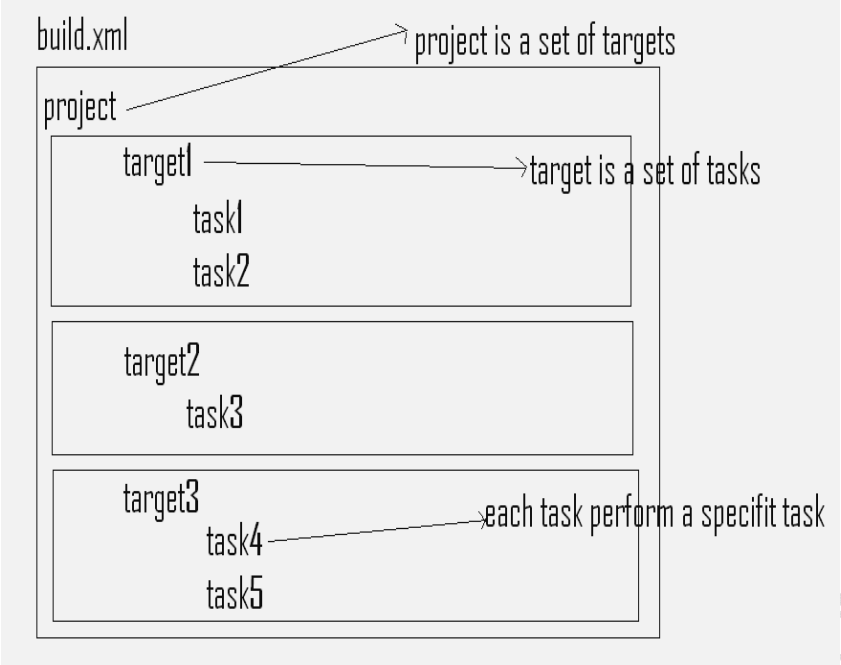
```
C:\>ant
Buildfile: build.xml does not exist!
Build failed
C:\>
```

build.xml File

Ant is a build tool that means the main aim of Ant tool is to automation JAVA Project process.
By default ant uses build.xml file for writing build script. It is an XML file.
It is a standard file name followed by everyone. It is not mandatory
We can use any file name like banking.xml, bank-build.xml etc.
We can use ant tool to do the following things

- ✓ Create directories
- ✓ Delete directories
- ✓ Copy files from one directory to another directory
- ✓ Create new files
- ✓ Compile and run java files
- ✓ Create a war file
- ✓ Create a jar file
- ✓ Create an ear file
- ✓ Deploy war or ear into a server etc.

build.xml File Architecture



Content of build.xml file:

Each JAVA project contains one or more build script files. Each Build script file contains a project. Default build file name is 'build.xml', but we can use any name.
Ant's build files are written in XML.

Project:
We can represent this project by using <project> element and it contains some attributes like attribute "name" is use to specify project name etc.
That means build.xml contains <project> as root element.
Example:-
<project name="bank" .....>
</project>
The <project> tag has three attributes:

- ✓ name
- ✓ default
- ✓ basedir

<Project> attributes description:

- ✓ The name attribute gives the project a name.
- ✓ The default attribute refers to a target name within the buildfile. If you run Ant without specifying a target on the command line, Ant executes the default target. If the default target doesn't exist, Ant returns an error.

- ✓ The basedir attribute defines the root directory of a project. Typically, it is ".", the directory in which the buildfile resides, regardless of the directory you're in when you run Ant. However, basedir can also define different points of reference.

Target:

Each <project> contains a set of targets. At least one target is mandatory.

We can represent this target using <target> element
Just like <project>, each target contains one name. We can represent this target name using “name” attribute.
We should provide unique target names within one build script file

Example:-

```
<project  name=”bank”  …..>

      <target  name=”target-1” …>
      </target>

      <target  name=”target-2” …>
      </target>

      <target  name=”target-3” …>
      </target>

</project>
```

The <target> tag has three attributes:

- ✓ name
- ✓ depends
- ✓ description

The name attribute gives the target a name.
The depends attribute are a comma-separated list of names of targets on which this target depends.
The description attribute a short description of this target's function.

Task:

Each target contains one or more number of tasks.
We can write targets without tasks also that means tasks are not mandatory.

Syntax:

```
<project name="someprojectname">
<target name="dosomething">
      <task1 param1="value1" param2="value2">
      <task2 param3="value3" >
      ...
</target>
<target name="target2">
      <task1 param1="value1" param2="value2">
      <task2 param3="value3" >
      ...
</target>
</project>
```


3. Ant Sample Build File - JAR:

Example 1:

In this example you will see how to compile a java program and compress it into a **.jar** file using Ant build file. The following listing shows the **build.xml** file.

```
01. <? xml version="1.0" ?>
02. <project name="Hello World" default="compress">
03.
04. <target name="compile">
05.     <javac srcdir="."/>
06.     <echo> Compilation Complete! </echo>
07. </target>
08.
09. <target name="compress" depends="compile">
10.     <jar destfile="HelloWorld.jar" basedir="."
                    includes="*.class" />
11.     <echo> Building .jar file Complete! </echo>
12. </target>
13.
14. </project>
```

- The ***<project>*** element is the root element in Ant build files. The ***name*** attribute of the **<project>** element indicates the project name. Each project element can contain multiple ***<target>*** elements.
- A **<target>** represents a single stage in the build process. A build process can have multiple **<targets>.** Here we have two targets ***compile*** and ***compress*.**
- The **default** attribute of <**project>** element indicates the default target to be executed. Here the default target is ***compress*.**
- When you see the <***compress>*** target, it in turn depends on the ***<compile>*** target, that is indicated by the ***depends*** attribute. So the <***compile>*** target will be executed first.
- The <**compile>** target has two task elements ***<javac>*** and ***<echo>*.** The ***javac*** task is used to compile the java files. The attribute ***srcdir="."*** indicates all the java files in the current directory. *Echo* task is used to display message on the console.
- The compress target also performs two tasks, first the <*jar*> element as the name indicates, is used to build the jar file. The attributes ***destfile="HelloWorld.jar", basedir="."*** and ***includes="*.class"*** indicates all the **.class** files in the current directory should be compressed into ***HelloWorld.jar*** file. Later the **echo** task is used to display the success message on the console.

To run the ***build.xml*** file, open the command prompt, go to the example directory, type the command "*ant*". You will see the following information.

```
C:\WINDOWS\system32\cmd.exe
E:\ant examples\Example1>ant
Buildfile: build.xml

compile:
    [javac] Compiling 1 source file
     [echo]  Compilation Complete!

compress:
      [jar] Building jar: E:\ant examples\Example1\HelloWorld.jar
     [echo]  Building .jar file Complete!

BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example1>_
```


Example2:

In this example we will see how to structure the project. If the grows bigger, it will become a problem to manage the files if all the files are there in the same directory.

For a easier maintenance all the source file should be kept in the *src* directory, the compressed jar file should be kept in the *dist* directory and all the intermediate class files should be kept in the *build/classes* directory.

By imposing structure cleaning the project becomes easy, we can just delete the directory and recreate it.

Using the <mkdir> task we create *build/classes* and *dist* directory.

```
1. <target name="init">
2.    <mkdir dir="build/classes" />
3.    <mkdir dir="dist" />
4. </target>
```

During the compilation process all the java files in the **src** directory will be compiled and the generated class files will be placed in the **build/classes** directory.

Since we placed all the class files under the **build/classes** directory, creating jar file becomes easier, you can simply specify the ***basedir*** attribute as **"*build/classes*"** instead of specifying ***basedir="."*** and ***includes="*.class"*.** After creating the jar file we place it in the dist directory **(*destfile="dist/HelloWorld.jar"*).**

```
1. <target name="compile" depends="init">
2.    <javac srcdir="src" destdir="build/classes" />
3. </target>
4.
5. <target  name="compress" depends="compile">
6.     <jar destfile="dist/HelloWorld.jar"
                        basedir="build/classes"/>
7. </target>
```

You can use the *java* task to execute a class file as shown below. The **classname** attribute refers to **the java class to be executed** and the **classpath** attribute refers to **the directory in which the class is located.**

```
1. <target name="execute" depends="compile">
2. <java classname="com.vaannila.helloworld.HelloWorld"
          classpath="build/classes" />
3. </target>
```

Since all the class files and jar files are isolated, we can easily clean the project by deleting the respective directories.

```
1. <target name="clean">
2. <delete dir="build" />
3. <delete dir="dist" />
4. </target>
```

The default target is ***compress***, so when you run the build file the ***compress*** target will be executed. The ***compress*** target depends on ***compile*** target which in turn depends on the ***init*** target, so first the ***init*** target will be executed and the two directories will be created, then the ***compile*** target will be execute, later the jar file is created.

```
C:\WINDOWS\system32\cmd.exe

E:\ant examples\Example2>ant
Buildfile: build.xml

init:
    [mkdir] Created dir: E:\ant examples\Example2\build\classes
    [mkdir] Created dir: E:\ant examples\Example2\dist

compile:
    [javac] Compiling 1 source file to E:\ant examples\Example2\build\classes

compress:
      [jar] Building jar: E:\ant examples\Example2\dist\HelloWorld.jar

BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example2>_
```

When you run the **"*ant execute*"** command the ***execute*** target will be invoked. ***Execute*** target also depends on ***compile*** target which in turn depends on ***init*** target, but now the directories won't be created again because it already exist. Since the java file is already compiled it won't be compiled again. Ant checks the timestamp of the file and compiles only the updated files. The *HelloWorld.class* file will be executed and the "*Hello World!*" message will be displayed on

the console.

```
C:\WINDOWS\system32\cmd.exe
E:\ant examples\Example2>ant execute
Buildfile: build.xml

init:

compile:

execute:
     [java] Hello World!

BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example2>_
```

To clean and execute the program run the ***"ant clean execute"*** command. First the ***clean*** target will be executed and the directories will be deleted, later the execute task will be invoked.

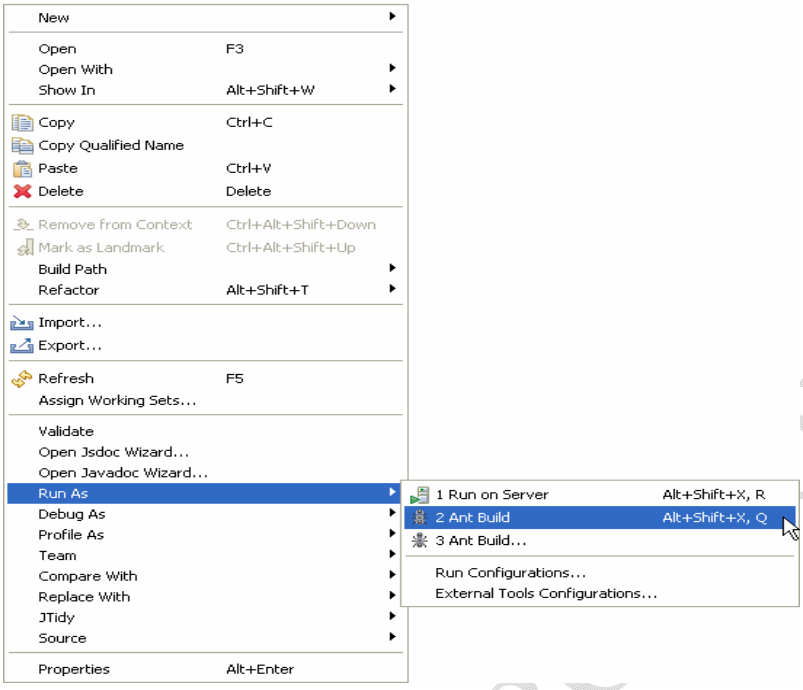
```
C:\WINDOWS\system32\cmd.exe
E:\ant examples\Example2>ant clean execute
Buildfile: build.xml

clean:
   [delete] Deleting directory E:\ant examples\Example2\build
   [delete] Deleting directory E:\ant examples\Example2\dist

init:
    [mkdir] Created dir: E:\ant examples\Example2\build\classes
    [mkdir] Created dir: E:\ant examples\Example2\dist

compile:
    [javac] Compiling 1 source file to E:\ant examples\Example2\build\classes

execute:
     [java] Hello World!

BUILD SUCCESSFUL
Total time: 0 seconds
E:\ant examples\Example2>_
```

4. Ant Eclipse IDE Integration

To integrate Ant build file with the Eclipse IDE, first create a **build.xml** file, right click the project select ***New->Other->XML***, enter the name as ***build.xml*** and click *Finish*.

```
<?xml version="1.0" ?>
<project name="Ant Example" default="execute">

<target name="init" depends="clean">
   <mkdir dir="build/classes" />
</target>

<target name="compile" depends="init">
<javac srcdir="src" destdir="build/classes" />
</target>

<target name="execute" depends="compile">
<java classname="com.durga.ant.HelloWorld"
      classpath="build/classes" />
</target>

<target name="clean">
<delete dir="build" />
</target>

</project>
```

To build the project right click the *build.xml* file and select ***Ant Build***.



The ***HelloWorld.java*** file will be compiled and executed. The following message shows the sequence of events that happen once the build file is executed.

```
01. Buildfile: E:\Eclipse Workspace\AntExample\build.xml
02. clean:
03. [delete] Deleting directory E:\Eclipse
                        Workspace\AntExample\build
04. init:
05.[mkdir] Created dir: E:\Eclipse
        Workspace\AntExample\build\classes
06. compile:
07. [javac] Compiling 1 source file to E:\Eclipse
            Workspace\AntExample\build\classes
08. execute:
09. [java] Hello World!
10. BUILD SUCCESSFUL
11. Total time: 625 milliseconds
```

5. Ant Property Task:

The ***<property>*** task is used to set the Ant properties. The property value is immutable, once the value is set you cannot change it. To set a property to a specific value you use Name/value assignment.

Property		
Attribute	**Description**	**Requirement**
name	name of the property, which is case-sensitive	not necessary
value	name of task attribute, this is done by placing the property name between "${"name }" in the attribute value	necessary
location	it contain property name	not necessary
file	name of the property file	not necessary

```
1. <property name="project.name" value="AntExample2" />
```

To set a property to a location you use Name/location assignment.

```
1. <property name="web.dir" location="WebContent"/>
2. <property name="web.lib.dir" location="${web.dir}/WEB-
        INF/lib"/>
3. <property name="build.classes.dir"
                        location="build/classes"/>
4. <property name="dist.dir" location="dist"/>
```

To use the properties surround them with *${}*.

6. Ant Properties File:

You can also group all the property values in a separate properties file and include it in the ant build file. Here the **build.properties** file contains all the property values. Remember the property value is immutable, so if you set a property value in the properties file you cannot change it in the build file. This gives more control over the build process.

The *build. Properties* file:

```
1. Web.dir=WebContent
2. web.lib.dir=${web.dir}/WEB-INF/lib
3. build.classes.dir=build/classes
4. dist.dir=dist
5. project.name=AntExample3
```

Use the **property** task to include the **properties file** in the **Ant build file**.

```
1.<property file="build.properties" />
```

(Q) How to ear your Java application using Ant?

We use ant taks **<ear>** to create ear of your Java Application

<ear> task syntax:

```
<ear destfile="ear-file-name-and-location"
      appxml="applicaton-config-file ">
    <fileset dir="files-dir"  includes="ListofJarsAndWars"/>
  </ear>
```

Parameters

destFile	The EAR file to create.
appxml	Display name to insert into the application.xml

Example:

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

7. Ant pre-defined Tasks:

The following List represents the available pre-defined task libraries in ant tool.

1. **Archive Tasks**
2. **Audit/Coverage Tasks**
3. **Compile Tasks**
4. **Deployment Tasks**

5. **Documentation Tasks**
6. **EJB Tasks**
7. **Execution Tasks**
8. **File Tasks**
9. **Java2 Extensions Tasks**
10. **Logging Tasks**
11. **Mail Tasks**
12. **Miscellaneous Tasks**
13. **Pre-process Tasks**
14. **Property Tasks**
15. **Remote Tasks**
16. **SCM Tasks**
17. **Testing Task**

Note: if you want to know more about Ant pre-defined tasks, please refer the site **http://ant.apache.org/manual/tasksoverview.html**


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com