# Apache ANT – a Tool for Automating Software Build Processes and its Importance in Testing

In the last module, we tried to make you acquainted with the[concept of generics and common methods](http://www.softwaretestinghelp.com/creating-generics-and-testsuites-selenium-tutorial-22/). We also discussed the benefits we get out of generics like reusability. We also shared the practical approaches towards creation of generics and their accessibility.

In the current module in this [**Selenium automation**](http://www.softwaretestinghelp.com/selenium-tutorial-1/) series, we would shed light on **a build tool named as “Apache Ant”**. We would broadly discuss its applicability and importance besides the practical approach.

*Take a note that the module is limited to testing aspects of using Apache Ant.*

Apache Ant is a very popular and conventional build tool of our times. Ant is an open source java based build tool provided by Apache Software Foundation freely distributed under GNU license. Apache Ant plays a significant role in developer’s as well as Tester’s day to day work schedule. The tool has immense power to build the development code into deployment utilities.

**Ant is a tool that automates the software building process.** **Ant is not just limited to compilation of code, rather packaging, testing and a lot more can be achieved in some simple steps.**

The tool works on the principle of targets and dependencies defined in the XML files. Ant libraries are used to build the applications. The libraries have a set of defined tasks to archive, compile, execute, document, deploy, and test and many more targets. Moreover, Ant allows the user to create his/her own tasks by implementing their own libraries.

Ant is primarily used with Java Applications but it can still be used for applications built on other languages depending on the extended support.

The most important aspect of using Ant is that it doesn’t demands another set of code to be written in order to build the application, rather the entire process is defined by targets which are none other than XML elements.

**Apache Ant Benefits**

* **Ease of Use** – The tool provides a wide range of tasks that almost fulfills all the build requirements of the user.
* **Platform Independent** – Ant is written in Java thus is a platform independent build tool. The only requirement for the tool is JDK.
* **Extensibility** – As the tool is written in Java and the source code is freely available, user is leveraged with the benefit to extend the tool’s capabilities by writing java code for adding task in Ant Libs.

**Apache Ant Features**

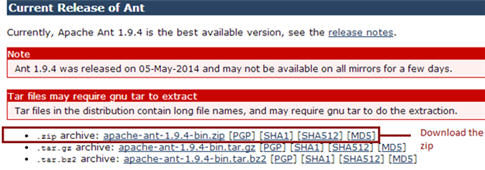
* Can compile java based applications
* Can create Java Doc
* Can create war, jar, zip, tar files
* Can copy files to at different locations
* Can delete or move files
* Can send Emails to the stakeholders
* Supports Junit 3, Junit 4, TestNG etc.
* Can convert XML based test reports to HTML reports
* Can make directories
* Can check out the code from version control system (SVN, GIT, CVS etc).
* Can execute test scripts and test suites

**Environment Setup**

Let us demonstrate the entire setup process step by step.

**Step 1: Apache Ant Download**

The first and the foremost step is to download the zipped folder of Apache Ant latest version from the repository. The distribution is available at “<http://ant.apache.org/bindownload.cgi>”.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Apache-ANT-1.jpg)

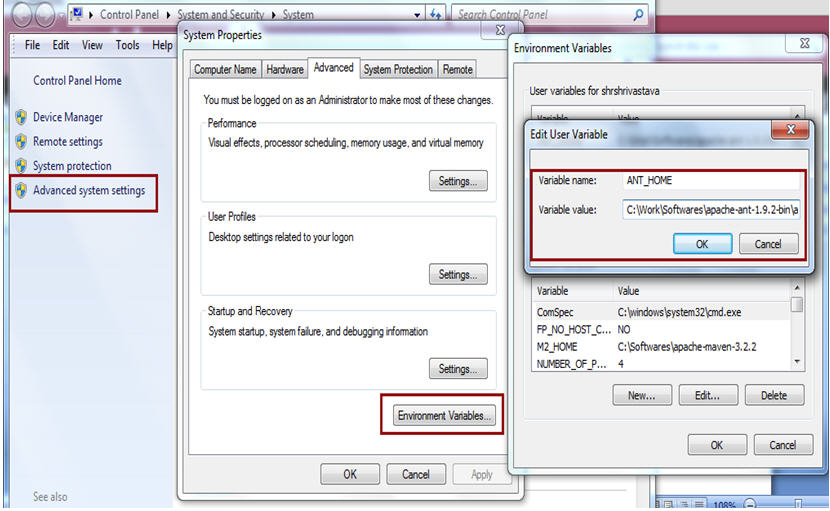
**Step 2: Extract folder and Set Environment Variables**

Extract the zipped folder at any desired location onto local file system.

Prior to setting up environment for Ant, it is required to install and set JDK on to your system. I am assuming that the JDK is already set and installed, Thus moving forward with the Ant Setup.

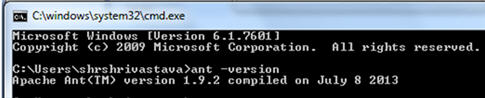
Create an environment variable for “ANT\_HOME” and set the variable’s value to the location of Ant folder. Refer the following screenshot for the same.

***(Click to enlarge image)***

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Apache-ANT-2.jpg)

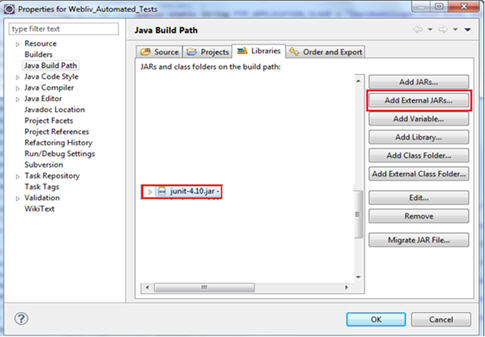
Edit the Path variable to append the location of the bin folder i.e. compiler location.

User can also verify for the successful Ant installation by typing in the “ant -version” command in the command prompt. The user would be able to see the following screen for the successful installation.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Apache-ANT-3.jpg)

**Step 3: Download and Extract Junit Jar**

Download the latest version of Junit jar from “[https://github.com/junit-team/junit/wiki/Download-and-Install](http://github.com/junit-team/junit/wiki/Download-and-Install)” and configure the project’s build path in eclipse and add the jar as external library. Refer the following illustration.

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Apache-ANT-4.jpg)

Thus, no other installation is required to use Apache Ant in collaboration with Junit and Selenium WebDriver to build, execute and report the test scripts.

***Note***: Take a note to necessarily add “ant-junit4.jar” jar file that can be found within the library folder of the Ant’s software distribution.

**Sample Build.xml**

The next step is to create the project’s build file. Build file is nothing but a collection of xml elements. Worth mentioning that one build file can relate to one and only one project i.e. one build file per project or vice versa. Build file is customarily located at the project’s root/base folder but the user is leveraged to select the build’s location driven by his/her wish. Moreover the user is free to rename the build file if he/she desires.

Each of the build file must have one project and at least one target element. Refer the sample build.xml

------------

|  |  |
| --- | --- |
| 1 | <?xml version="1.0" encoding="UTF-8"?> |
| 2 | <project name="Learning\_Selenium" default="junitReport"basedir="."> | |

|  |  |
| --- | --- |
| 3 | <property name="src" value="./src" /> |
| 4 | <property name="lib" value="./lib" /> |

|  |  |
| --- | --- |
| 5 | <property name="bin" value="./bin" /> |
| 6 | <property name="report" value="./report" /> | |

|  |  |  |
| --- | --- | --- |
| 7 | <property name="test.dir" value="./src/com/tests"/> | |
| 8 | <path id="Learning\_Selenium.classpath"> |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | <pathelement location="${bin}" /> | | |
| 10 | | <fileset dir="${lib}"> |

|  |  |  |
| --- | --- | --- |
| 11 | <include name="\*\*/\*.jar" /> | |
| 12 | </fileset> |

|  |  |
| --- | --- |
| 13 | </path> |
| 14 | <echo message="-----------------------------------------------------------------" /> | |

|  |  |
| --- | --- |
| 15 | <echo message="--------------------Selenium Learning Tests----------------------" /> |
| 16 | <echo message="-----------------------------------------------------------------" /> |

|  |  |
| --- | --- |
| 17 | <target name="init" description="Delete the binary folder and create it again"> |
| 18 | <echo message="----------Delete the binary folder and create it again----------" /> |

|  |  |
| --- | --- |
| 19 | <delete dir="${bin}" /> |
| 20 | <!-- Create the time stamp --> | |

|  |  |
| --- | --- |
| 21 | <tstamp> |
| 22 | <format property="lastUpdated"pattern="dd-MM-yyyy HH:mm:ss" /> | |

|  |  |
| --- | --- |
| 23 | </tstamp> |
| 24 | <!-- Create the build directory structure used by compile --> | |

|  |  |  |
| --- | --- | --- |
| 25 | <mkdir dir="${bin}" /> | |
| 26 | </target> |

|  |  |
| --- | --- |
| 27 | <target name="compile" depends="init"description="Compile the source files"> |
| 28 | <echo message="----------Compile the source files----------" /> |

|  |  |  |  |
| --- | --- | --- | --- |
| 29 | <javac source="1.7" srcdir="${src}"fork="true" destdir="${bin}" includeantruntime="false"debug="true" debuglevel="lines,vars,source"> | | |
| 30 | | <classpathrefid="Learning\_Selenium.classpath" /> |

|  |  |  |
| --- | --- | --- |
| 31 | </javac> | |
| 32 | </target> |

|  |  |  |
| --- | --- | --- |
| 33 | <target name="exec" depends="compile"description="Launch the test suite"> | |
| 34 | <echo message="----------Launch the test suite----------" /> |

|  |  |  |
| --- | --- | --- |
| 35 | <delete dir="${report}" /> | |
| 36 | <mkdir dir="${report}" /> |

|  |  |  |
| --- | --- | --- |
| 37 | | <mkdir dir="${report}/xml" /> |
| 38 | <junit fork="yes"printsummary="withOutAndErr" haltonfailure="no"> | | |

|  |  |  |
| --- | --- | --- |
| 39 | <classpathrefid="Learning\_Selenium.classpath" /> | |
| 40 | <formatter type="xml" /> |

|  |  |  |
| --- | --- | --- |
| 41 | <batchtest fork="yes"todir="${report}/xml"> | |
| 42 | | <fileset dir="${src}"includes="\*\*/com/TestSuite.java" /> |

|  |  |  |
| --- | --- | --- |
| 43 | </batchtest> | |
| 44 | </junit> |

|  |  |
| --- | --- |
| 45 | </target> |
| 46 | <target name="junitReport" depends="exec"description="Generate the test report"> | |

|  |  |  |
| --- | --- | --- |
| 47 | <echo message="----------Generate the test report----------" /> | |
| 48 | <junitreport todir="${report}"> |

|  |  |
| --- | --- |
| 49 | <fileset dir="${report}/xml"> |
| 50 | <include name="TEST-\*.xml" /> | |

|  |  |
| --- | --- |
| 51 | </fileset> |
| 52 | <report format="frames"todir="${report}/html"> | |

|  |  |  |  |
| --- | --- | --- | --- |
| 53 | <param name="TITLE"expression="Selenium\_Learning\_Report" /> | | |
| 54 | | </report> |

|  |  |  |
| --- | --- | --- |
| 55 | </junitreport> | |
| 56 | </target> |

|  |  |
| --- | --- |
| 57 | </project> |

**Explanation of Build.xml**

The project element is fundamentally consists of 3 attributes:

*<project* *name=“Learning\_Selenium”* *default=“junitReport”basedir=“.”>*

**Each of the attribute has a “Key-Value pair” structure.**

* **Name** – The value of the name attribute represents the name of the project. Thus in our case, the project’s name is “Learning\_Selenium”.
* **Default** – The value of the default attribute represents the compulsory target for the build.xml. A build.xml file can have any number of targets. Thus this field represents the mandatory target amongst all.
* **Basedir** – Represents the root folder or base directory of the project. Under this directory, there may be several other folders like src, lib, bin etc.

*<target* *name=“init”* *description=“Delete the binary folder and create it again”>*

All the tasks in the Ant build file are defined under Target elements. Each Target element corresponds to a particular task or goal. A single target can consists of multiple tasks if needed. Like I mentioned earlier, the user is credited to create more than one target within a particular build file.

**In the above xml code, we have created targets for the following goals:**

1. Deleting and creating directories
2. Compiling the code
3. Executing the test classes
4. Generating the test reports

*<target* *name=“exec”* *depends=“compile”* *description=“Launch the test suite”>*

Sometimes it is required to execute a particular target only when some other target is executed successfully. Take a note that the target are executed sequentially i.e. in order of sequence they are mentioned in the build file. Also I would like to mention that a particular target is executed once and only once for the current build execution. Thus, when the user is required to generate dependency between the target, he/she has to use depends attribute. The value of the “depends” attribute shall be the name of the target on which it depends. A target can depend on more than one target as well.

**Built-in Tasks**

Ant build file provides varieties of tasks. Few of them are discussed below:

**File Tasks** – File task are self explanatory.

1. <copy>
2. <concat>
3. <delete>
4. <get>
5. <mkdir>
6. <move>
7. <replace>

**Compile Tasks**

1. <javac> – Compiles source files within the JVM
2. <jspc> – Runs jsp compiler
3. <rmic> – Runs rmic compiler

**Archive Tasks**

1. <zip>, <unzip> – Creates a zipped folder
2. <jar>, <unjar> – Creates a jar file
3. <war>, <unwar> – Creates a war file for deployment

**Testing Tasks**

1. <junit> – Runs JUnit testing framework
2. <junitreport> – Generates the test report by converting JUnit generated XML test reports.

**Property Tasks**

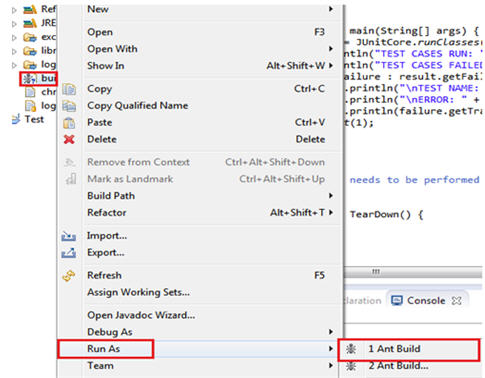
1. <dirname> – Sets the property
2. <loadfile> – Loads a file into property
3. <propertyfile> – Creates a new property file

**Misc. Tasks**

1. <echo> – Echoes the text message to be printed either on the console or written within an external file.
2. <javadoc> – Generates the java based documentation using javadoc tool.
3. <sql> – Establishes a JDBC connection and hits dash of SQL commands.

**Execution**

The easiest section is to execute the test suite with Ant. To execute the test suite with Ant, Right click on “build.xml” and select “Run As -> Ant Build” option. Thus, the option hits the execution. Refer the following figure for the same.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Apache-ANT-5.jpg)

After the entire execution is completed, Ant generates a test execution report for review inside the “Report” folder.

The execution can also be initiated outside the eclipse by hitting the command on the command prompt. User is expected to navigate to the directory where build.xml is kept and type “ant”.

**Conclusion**

In this module, we laid emphasis on useful information related to Ant, its installation and various Ant tasks. Our motive was to at least introduce you with the basic conceptual picture and its importance as a tool all together with respect to testing. Hence, we discussed build.xml in detail describing the various components.

Briefing in the end, Ant is a tool that automates the software building process. Ant is not just limited to compilation of code, rather packaging, testing and a lot more can be achieved in some simple steps.