****

**What is Maven?**

Maven is a software project management and build tool primarily used with Java-based projects but that can also be used to manage projects in other programming languages like C# and Ruby. Maven helps manage builds, documentation, reporting, dependencies, software configuration management (SCM), releases and distribution.

It simplifies the build process like ANT. But it is too much advanced than ANT.

# Alternative build tools for maven projects?

ANT

Gradle

# Understand the problem without maven?

There are many problems that we face during the project development. They are..

1. **Adding set of Jars in each project**: In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.
2. **Creating the right project structure**: We must create the right project structure in servlet, struts ,spring ..etc otherwise it will not be executed.
3. **Building and Deploying the project:** We must have to build and deploy the project so that it may work.

# What Build tools will do for us?

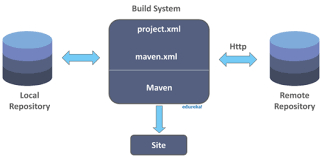
Maven simplifies the above mentioned problems. It does mainly following tasks.

* + It makes a project easy to build
  + It provides uniform build process (maven project can be shared by all the maven projects)
  + It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
  + It is easy to migrate for new features of Maven Apache Maven helps to manage
  + Builds
  + Documentation
  + Reporting
  + SCMs
  + Releases
  + Distribution

# what is difference between ANT and MAVEN?

|  |  |
| --- | --- |
| **ANT** | **Maven** |
| 1. Ant doesn't has formal conventions, so we need to provide information of the project structure in build.xml file. 2. Ant is procedural, you need to provide information about what to do and when to do through code. You need to provide order. 3. There is no life cycle in Ant. 4. It is a tool box. 5. It is mainly a build tool. 6. The ant scripts are not reusable. 7. It is less preferred than Maven. | 1. Maven has a convention to place source code, compiled code etc. So we don't need to provide information about the project structure in pom.xml file. 2. Maven is declarative, everything you define in the pom.xml file. 3. There is life cycle in Maven. 4. It is a framework. 5. It is mainly a project management tool. 6. The maven plug-in are reusable. 7. It is more preferred than Ant. |

**Maven Architecture**

****

A maven repository is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

Local repository then Remote repository then Central repository.

If dependency is not found in these repositories, maven stops processing and throws an error.

# Local Repository:

Maven local repository is located in your local system. It is created by the maven when you run any maven command.

By default, maven local repository is %USER\_HOME%/.m2 directory. For example:

C:\Users\MADHU\.m2

if you want to change local repository we can change using setting.xml (it is available in C:\Users\<USER\_NAME>\.m2 directory ) file under **<localRepository>** tag.

1. **Remote repository:**

Maven Remote repository available in your network. we will palace all needed libraries in one places we can access.

we need to configure remote repository url in pom.xml

1. **Central Repository:**

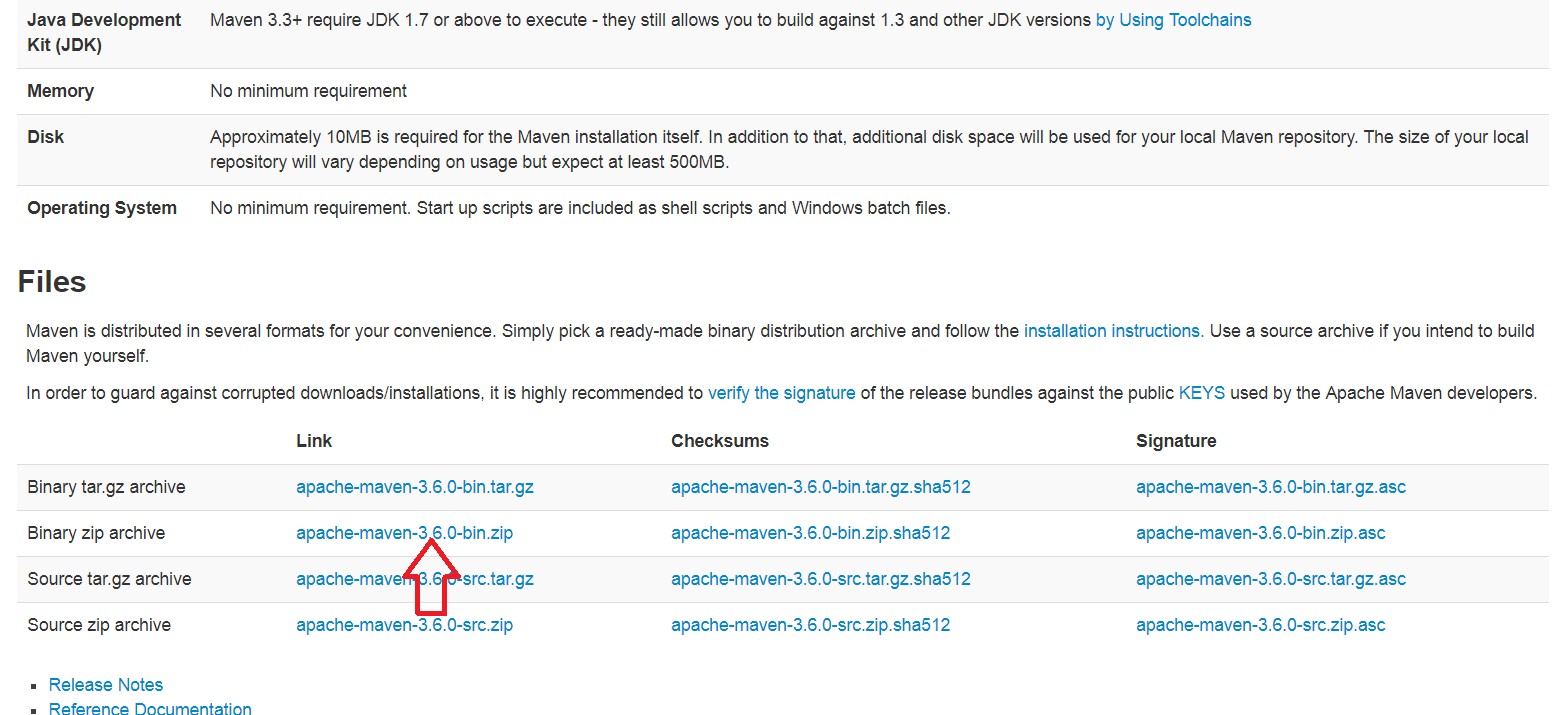
Maven central repository is located on the web. It has been created by the apache maven community itself.

The path of central repository is: <http://repo1.maven.org/maven2/>.

The central repository contains a lot of common libraries that can be viewed by this url [http://search.maven.org/#browse](http://search.maven.org/%23browse).

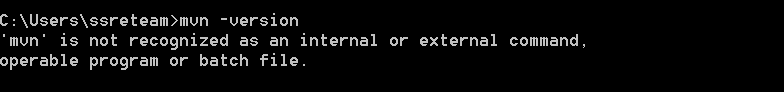
The default repository in central repository.

**Maven Installation in windows** download maven from below link. <https://maven.apache.org/download.cgi>



Download the zip file and extract. Now we need to set the JAVA\_HOME and MAVEN\_HOME and path of maven.

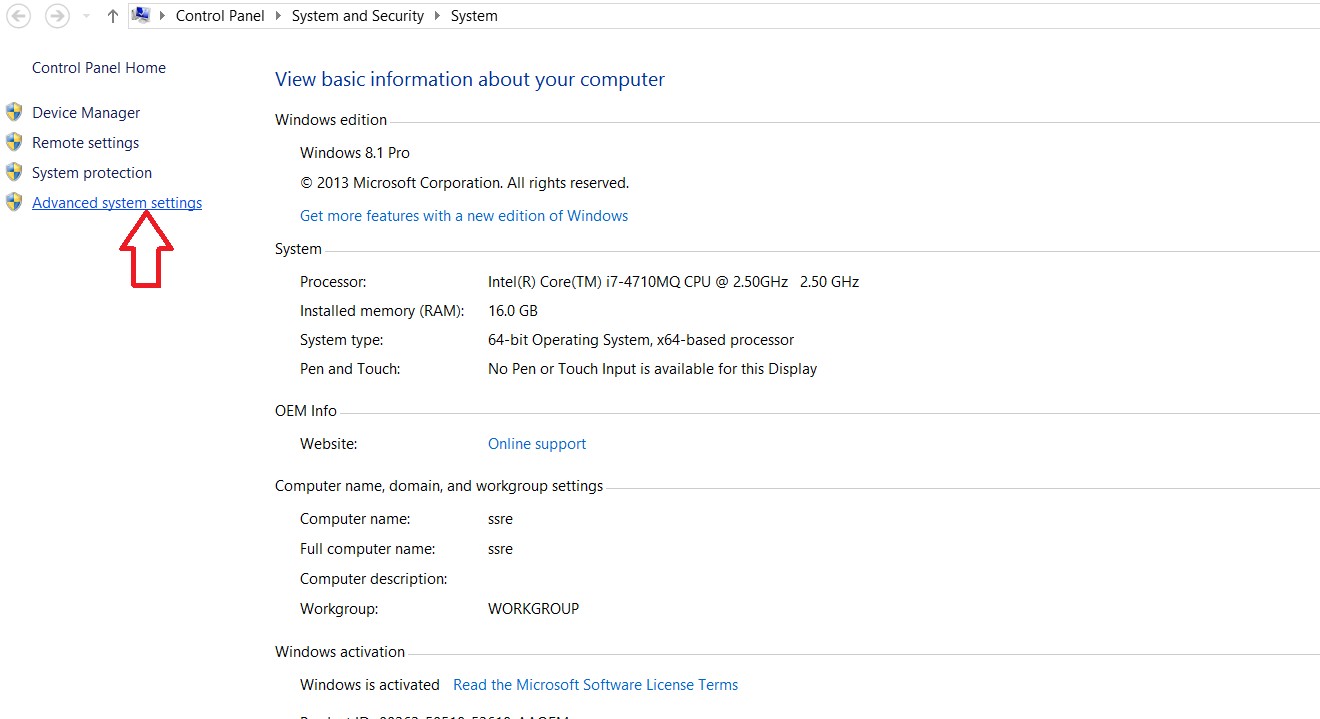
If not set the path if you try to run any mvn commands will give mvn command is not found like below.

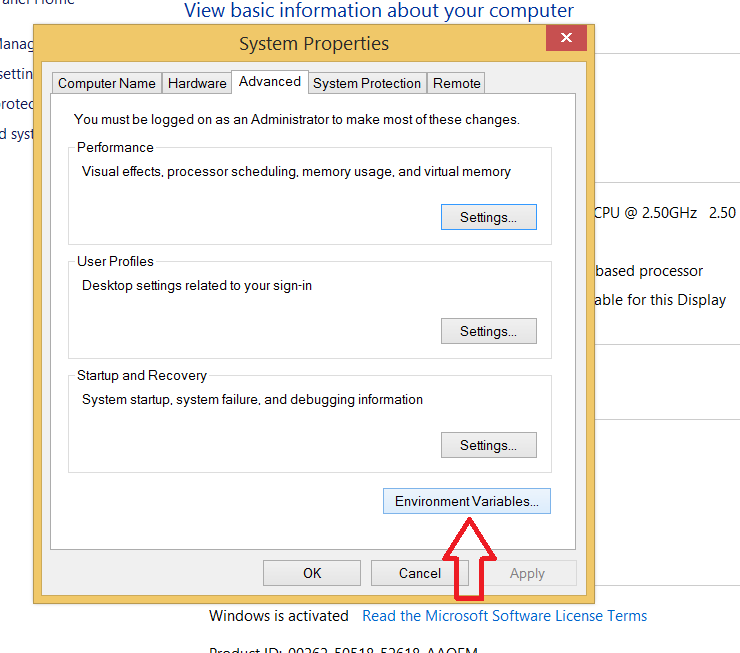


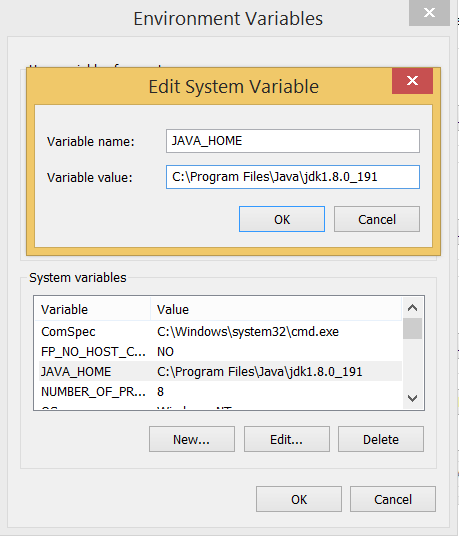
set the JAVA\_HOME path if not set already.

My computer --> right click Properties --> Advance System Settings --> Environment variables

--> system variables --> new... --> enter JAVA\_HOME is key and value is Home directory of java.



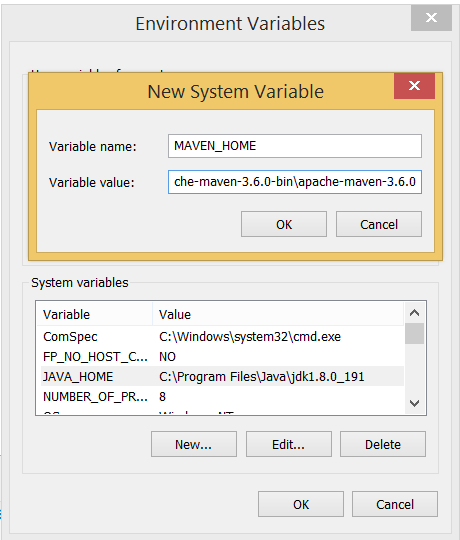




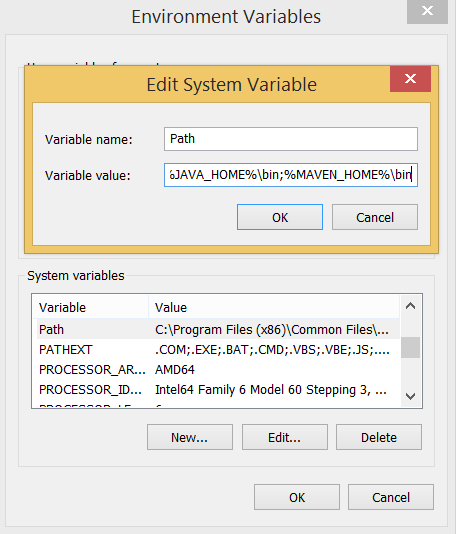
Now add the MAVEN\_HOME too..

My computer --> right click Properties --> Advance System Settings --> Environment variables

--> system variables --> new... --> enter MAVEN\_HOME is key and value is Home directory of maven.



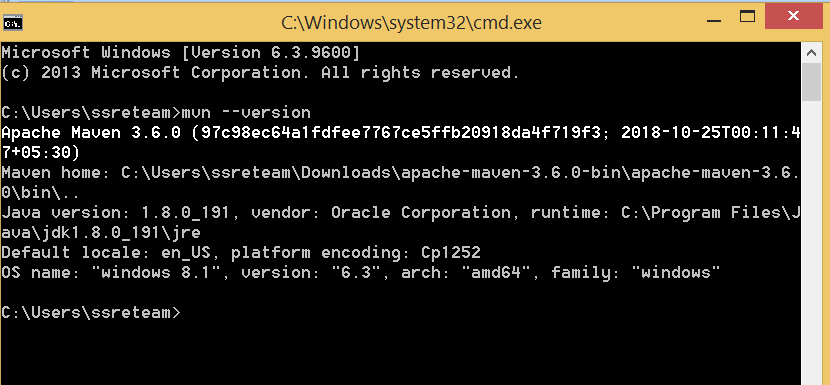
add the path of maven directories. edit the path of system variable..add the up-to bin folder of maven.



Now lets confirm the maven is configured properly or not open the new command prompt

and run below command.

# mvn --version



Done...we are successfully configured maven in our system..let's do further.

**Life cycle of Maven**

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired.

There are three built-in build lifecycles: **default**, **clean** and **site**. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation.

# A Build Lifecycle is Made Up of Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

* **validate** - validate the project is correct and all necessary information is available

* **compile** - compile the source code of the project

* **test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

* **package** - take the compiled code and package it in its distributable format, such as a JAR.

* **verify** - run any checks on results of integration tests to ensure quality criteria are met

* **install** - install the package into the local repository, for use as a dependency in other projects locally

* **deploy** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

# Command Line calls:

In a development environment, use the following call to build and install artifacts into the local repository.

# mvn install

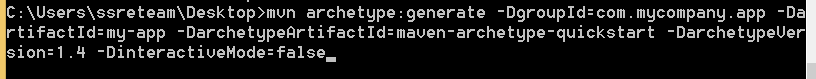
This command executes each default life cycle phase in order (validate, compile,test, package, verify..), before executing install. You only need to call the last build phase to be executed, in this case, install.

# How to create maven project trough command line?

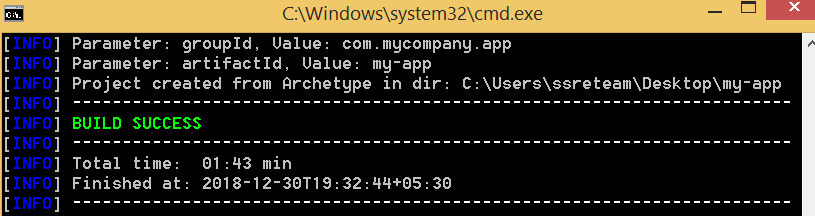
You will need somewhere for your project to reside, create a directory somewhere and start a shell in that directory. On your command line, execute the following Maven goal:

# mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app - DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -

**DinteractiveMode=false**



**the output is :**



If you have just installed Maven, it may take a while on the first run. This is because Maven is downloading the most recent artifacts (plugin jars and other files) into your local repository. You may also need to execute the command a couple of times before it succeeds. This is because the remote server may time out before your downloads are complete. Don't worry, there are ways to fix that.

You will notice that the generate goal created a directory with the same name given as the artifactId. let's see the project structure



In maven 2 files are most important 1. pom.xml 2. setting.xml

# pom.xml file :

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about **the project and configuration details**(dependencies,plugins ..etc ) used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is **target**; the source directory, which is **src/main/java**; the test source directory, which is src/test/java; and so on. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

***Sample pom.xml file***

<?xml version="1.0" encoding="UTF-8"?>

<project [xmlns="http://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0) [xmlns:xsi="http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema-instance)-instance"

[xsi:schemaLocation="http://maven.apache.org/POM/4.0.0](http://maven.apache.org/POM/4.0.0) [http://maven.apache.org/xsd/maven-4.0.0.xsd">](http://maven.apache.org/xsd/maven-4.0.0.xsd)

<modelVersion>4.0.0</modelVersion>

<groupId>com.saucelabs</groupId>

<artifactId>sauce\_appium\_junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>sauce\_appium\_junit</name>

<description>Sample Appium tests using JUnit</description>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.2.4</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>saucelabs-repository</id>

<url>https://repository- saucelabs.forge.cloudbees.com/release</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

</project>

ref : https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

# setting.xml file:

The settings element in the **settings.xml** file contains elements used to define values which configure Maven execution in various ways, like the **pom.xml**, but should not be bundled to any specific project, or distributed to an audience. These include values such as **the local repository location, alternate remote repository servers, and authentication information.**

There are two locations where a settings.xml file may live:

* + The Maven install: ${maven.home}/conf/settings.xml
  + A user’s install: ${user.home}/.m2/settings.xml

The former settings.xml are also called global settings, the latter settings.xml are referred to as user settings. If both files exists, their contents gets merged, with the user-specific settings.xml being dominant.

<settings [xmlns="http://maven.apache.org/SETTINGS/1.0.0"](http://maven.apache.org/SETTINGS/1.0.0) xmln[s:xsi="http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema-instance)-[instance"](http://www.w3.org/2001/XMLSchema-instance) [xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0](http://maven.apache.org/SETTINGS/1.0.0)

https://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository/>

<interactiveMode/>

<offline/>

<pluginGroups/>

<servers/>

<mirrors/>

<proxies/>

<profiles/>

<activeProfiles/>

</settings>

# ref : https://maven.apache.org/settings.html

