SYNTAX
TECHNOLOGIES

Framework

Agenda

What is Framework

What is Maven and why do we need it?

What is Maven Repository? Types of Maven Repo.

What is pom.xml file? Adding dependencies in pom.xml

What is Framework?

A testing framework is a set of guidelines or rules used for creating and designing test cases. These guidelines could include coding standards, test-data handling methods, object repositories, processes for storing test results, or information on how to access external resources.

Framework is a collection of concepts you know in an organised manner with some proper rules, validation, proper exception and error handling which can be easily reuse, manage and scale.

In a framework we create set of multiple reusable components like methods to launch browser, methods for clicking, sending keys, selecting dropdowns, switching between frames etc.

We can create methods for reading writing from external data sources, validation, reporting etc and can place it in such a way that you and other projects can also use it. It saves efforts, time and of course money of company.

Why Framework?

Framework defines the organization's way of doing things – a 'Single Standard'. Following this standard would result in the project team achieving:

Script-less representation of Automated tests: The testing framework should offer point-and-click interface for accessing and interacting with the application components under test—as opposed to presenting line after line of scripting.

Testers should be able to visualize each step of the business scenario, view and edit test cases intuitively. This will shorten the learning curve for testers and help QA teams meet deadlines.

Data Driven tests: Excel A key benefit of automating functional testing is the ability to test large volumes of data on the system quickly. But you must be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations with minimal effort.

Concise Reporting: The framework must automatically generate reports of the test run and show the results in an easy-to-read format. The reports should provide specifics about where application failures occurred and what test data was used.

Why Framework?

- Reports must present application screenshots for every step to highlight any discrepancies and provide detailed explanations of each checkpoint pass and failure. Reports must also be easily shared across the entire QA and development teams.

- **Standard Scripting and Team Consistency:** Modular Scripting standard should be maintained across the framework library creation, which includes business components, system communications, data check points, loggers, reporters etc. Project team should follow the defined scripting standards. Published standards across the project team pre-empt the effort involved in duplicate coding, which prevent individuals from following their own coding standards.

- **Implement and Maximize Re-Usability:** Documents Establish the developed libraries across the organization/project team/product team, i.e. publish the library and provide access rights. Utilities/components shared across the team. Usage of available libraries. Minimized effort for repeated regression cycle.

Characteristics of a framework

- Framework should be developed in such a way that any body can plug and use it. They should be able to start scripting with minimal or no setup.
- It must not be project dependent.
- It should be reusable and easy to manage and maintain.
- Flows should be clear.
- It should be simple so that an average automation tester could understand and use it.
- It should not contain hardcoded values.
- It should be easily extendable i.e. enhancing.
- It should be platform independent.

Benefits of Test Automation Framework

Utilizing a framework for automated testing will increase a team's test speed and efficiency, improve test accuracy, and will reduce test maintenance costs as well as lower risks. They are essential to an efficient automated testing process for a few key reasons:

- Improved test efficiency
- Lower maintenance costs
- Minimal manual intervention
- Maximum test coverage
- Reusability of code

Building Test Automation Framework

In order to build test automation framework we will be combining use of multiple tools

What is Maven?

- Maven is build automation and management tool developed by Apache Software Foundation. It was initially released on 13 July 2004.
- It allows the developer to create projects using **Project Object Model and plugins.**
- It helps to build **projects, dependency, and documentation**
- Maven allows the developer to automate the process of the creation of the initial folder structure, performing the compilation and testing and the packaging and deployment of the final product. It cuts down the good number of steps in build process and it makes it one step process to do a build which makes programmer’s life easier

Why To Use Maven In Selenium Project?

- Maven uses **POM.xml** configuration file which kept all project configuration related Information.
- For selenium, we need to provide selenium webdriver version related information in POM.xml file and then it will download all required jar files automatically and store it **in local repository called m2.**
- Later on if we want to change version of selenium webdriver then we **need to modify version** in POM.xml file only.Then maven will download new version jar files automatically and store in local repository.
- If we upgrade any dependencies version in POM.xml file, first maven will check if that version of jar files are available or not in local repository. If available then fine else It will **download them from maven central repository.**

Main objectives of Maven

- It makes project build process easy.
- It provides easy and uniform build system.
- It manages project dependencies by downloading required dependencies jar files automatically from Maven central repositories which also allows users to reuse same jars across multiple projects.
- Provides guidelines for better project management practices.
- It allows to build project using project object model (POM).
- It provides quality project document information.
- Allows execution of Tests from Command Line

What is a Maven Repository?

Maven Repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

There are three types of maven repository:

- **local** - is a folder location on your machine. It gets created when we run any maven command for the first time. Maven local repository keeps project's all dependencies (library jars, plugin jars etc.). Maven local repository by default get created by Maven in %USER_HOME% directory.

- **central** - is repository provided by Maven community. It contains a large number of commonly used libraries. This server downloads all the latests jar files from the remote maven repositories.

- **remote -** is a repository that manages by the companies who own the software, but they will expose the maven software to the maven.
  - Selenium jars are available at openqa servers
  - TestNG, POI jars are available at apache server

How Maven works?

- Maven is dependency management tool i.e to execute the frameworks, we need the jar files. Maven helps user to keep the up to date jar file on the framework.

- During runtime maven will check the version of the jar files present in the local system and compares with pom.xml file. If the latest version is available online then it downloads them automatically and replaces the local version and then executes the framework.

- If there is no internet connection or if the latest version is not available, then maven executes the framework with the jar files available in the local system.

- Maven can helps us to **minimize** project and build management **time and efforts**.

Create maven project in Eclipse

MAVEN helps us in creating the project structure, managing and downloading the dependencies.

We need to define the required dependencies in pom.xml. By default maven adds few dependencies specific to project structures and that will be based on the archetype that we choose.

After creating Maven project if we observe, we will see pom.xml file created.

Now we can start adding the dependencies which ever we require for our project like Selenium jars, Testng jars etc. in **pom.xml** file

What is pom.xml

POM is an acronym for **Project Object Model**.

pom.xml which is the core of any project. This is the configuration file where all required information are kept.

The pom.xml file contains **information of project and configuration information for the maven** to build the project such as **dependencies, build directory, source directory, test source directory, plugin, goals etc.**

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.syntax</groupId>
  <artifactId>HRMS</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

How to add dependency in Maven using Eclipse

In order to add the dependent jar for our project using Maven, we can add it through **<dependencies></dependencies>**tag in our **POM.xml** like below.

```
<dependencies>
		<dependency>

	<groupId>org.seleniumhq.selenium</groupId>
			<artifactId>selenium-java</artifactId>
			<version>3.141.5</version>
		</dependency>

		<dependency>
			<groupId>org.testng</groupId>
			<artifactId>testng</artifactId>
			<version>6.14.3</version>
			<scope>test</scope>
		</dependency>
</dependencies>
```

Elements of maven pom.xml file

Element	Description
project	It is the root element of pom.xml file.
model	It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0.
groupId	It is the sub element of project. It specifies the id for the project group. Generally groupId refers to domain id. For best practices company name is used as groupId. It identifies the project uniquely
artifactId	It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: **JARs, source and binary distributions, and WARs.**
version	It is the sub element of project. It specifies the version of the artifact under given group.
dependencies	defines dependencies for this project.

Maven Surefire Plugin

Maven surefire plugin is used to run the project tests.
It also allows us to configure which XML suites or Test Runners to execute when we build our project.
Below configuration will tell maven surefire plugin to execute only **specific Test Runner Class**

```
<build>
	<plugins>
		<plugin>

	<groupId>org.apache.maven.plugins</groupId>
					<artifactId>maven-surefire-
plugin</artifactId>
					<version>3.0.0-M3</version>
					<configuration>
						<includes>

	<include>**/*SmokeRunner.java</include>
						</includes>

	<testFailureIgnore>true</testFailureIgnore>
					</configuration>
		</plugin>
	</plugins>
</build>
```

Maven Lifecycles

Maven has the following three standard lifecycles:

- clean Lifecycle - involves cleaning of the project. Deletes all artifacts and targets which are created already. It handles everything related to removing temporary files from the output directory, including generated source files, compiled classes, previous JAR files etc. It has 3 phases: pre-clean, clean, post-clean.

- default Lifecycle (build lifecycle) - handles the complete deployment of the project (has 23 phases)

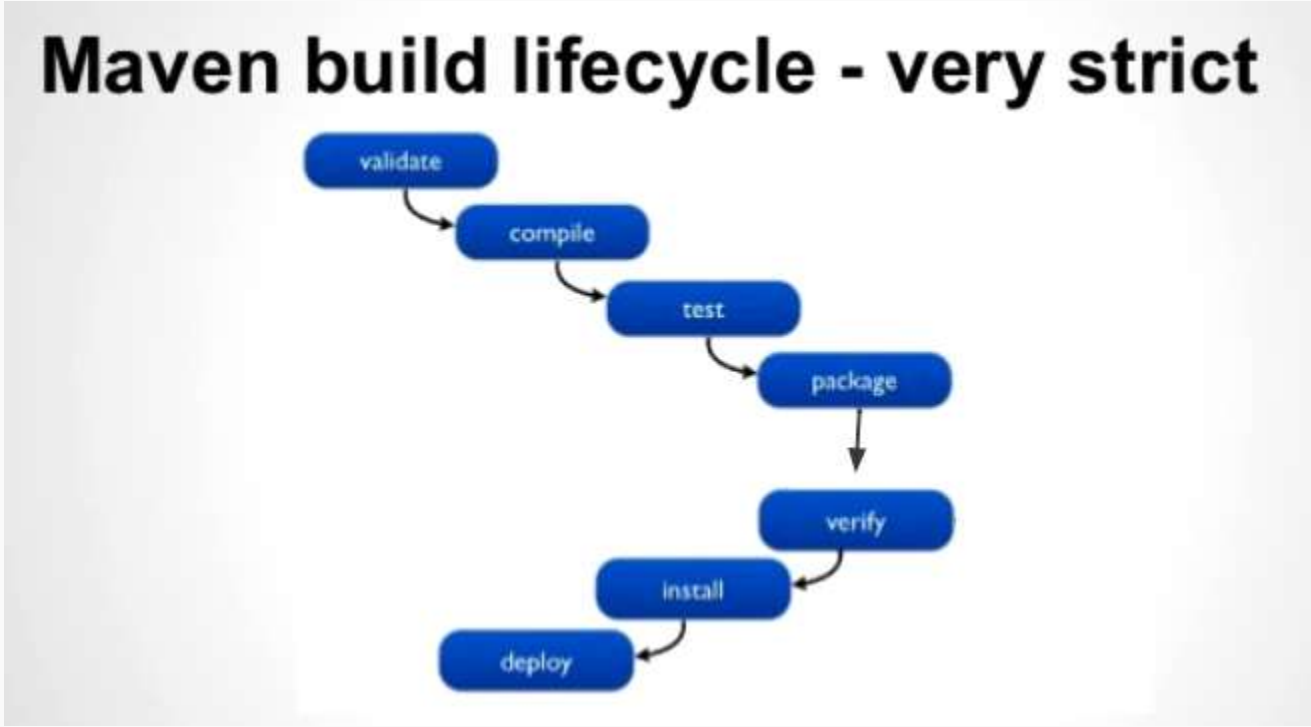
- site Lifecycle - handles generating the java documentation of the project. In fact, the site can generate a complete website with documentation for your project. It has 4 phases: pre-site, site, post-site, site-deploy.

Build Life Cycle

A Build Lifecycle is a sequence of tasks we used to build a software. For example, compile, test, package and publish or deploy are all tasks we need to do to build a software.

A Maven build lifecycle is a sequence of phases we need to go through in order to finishing building the software.

The following table lists some of the build lifecycle.



Build Life Cycle

Basic maven phases:

validate: validate the project is correct and all necessary information is available.

compile: used to compile the source code of the project.

test: test the compiled code

package: package is used to convert your project into a jar or war etc.

verify: run any checks to verify the package is valid and meets quality criteria.

install: install the package into the local repository for use of another project.

deploy: publish to integration or release environment

Maven Commands

Basic maven commands:

mvn clean : it deletes all class files, the java docs, the jars, reports and so on)

mvn test: run all the unit or testNG test classes.