# **QA AUTOMATION STANDARDS**

Contents

[**QA AUTOMATION STANDARDS** 1](#_Toc17809805)

[**Introduction** 3](#_Toc17809806)

[**Purpose** 3](#_Toc17809807)

[**Authors** 3](#_Toc17809808)

[**Source code management** 3](#_Toc17809809)

[**EGit Plugin for eclipse** 3](#_Toc17809810)

[**Branch strategy** 3](#_Toc17809811)

[**Feature branches** 4](#_Toc17809812)

[**Pull requests** 4](#_Toc17809813)

[**Approvals (TBD)** 4](#_Toc17809814)

[**Code standards** 4](#_Toc17809815)

[**Overview** 4](#_Toc17809816)

[**Standards for Variables** 5](#_Toc17809817)

[Standards for Parameters (Arguments) to Member Functions 5](#_Toc17809818)

[Naming Classes 5](#_Toc17809819)

[Naming Interfaces 6](#_Toc17809820)

[Naming Methods 6](#_Toc17809821)

[Naming Constants 6](#_Toc17809822)

[**Comment Standards** 6](#_Toc17809823)

[**Functions/Methods** 7](#_Toc17809824)

[Function Name 7](#_Toc17809825)

[Naming Accessor Member Functions 7](#_Toc17809826)

[Getters: 7](#_Toc17809827)

[Setters: 7](#_Toc17809828)

[Constructors: 7](#_Toc17809829)

[Function Header 7](#_Toc17809830)

[Internal Documentation 8](#_Toc17809831)

[Function Complexity 8](#_Toc17809832)

[**Automation Scripts** 8](#_Toc17809833)

[main source folder 8](#_Toc17809834)

[test source folder 9](#_Toc17809835)

[resource source folder 9](#_Toc17809836)

[**Indentation** 9](#_Toc17809837)

[**General Guidelines** 9](#_Toc17809838)

[**Framework Design** 10](#_Toc17809839)

[**Framework used** 10](#_Toc17809840)

[**Framework Architecture** 10](#_Toc17809841)

[**Framework Components** 11](#_Toc17809842)

[**Selenium WebDriver API** 11](#_Toc17809843)

[**TestNg** 12](#_Toc17809844)

[**Page Object Model** 12](#_Toc17809845)

[**Build Management using Maven** 12](#_Toc17809846)

[**Build Life Cycle:** 12](#_Toc17809847)

[**Maven Repository** 13](#_Toc17809848)

[**Log4j** 13](#_Toc17809849)

[**Apache POI API** 14](#_Toc17809850)

[**Extent Report for Reporting** 14](#_Toc17809851)

[**Continuous Integration and continues deployment (TBD)** 14](#_Toc17809852)

# **Introduction**

## **Purpose**

The purpose of this document is to outline the workflow and standards that

must be adhered in the development of automation scripts using Selenium and JAVA.

Also provides conventions and guidelines for designing and developing framework/scripts for java code in selenium automation. This document will help to ensure consistency across the code, resulting in increased usability and maintainability of the developed test scripts.

It provides brief description of each component used in our automation framework.

## **Authors**

This document was created by XXXX Author Name XXXX and maintained by QA TEAM

# **Source code management**

Git is the preferred system for version control and management of source code.

## **EGit Plugin for eclipse**

Source code Commit/Push/Pull/Rebase are done through EGIT plugin integrated with eclipse IDE.

EGit is an Eclipse Team provider for the Git version control system. Git is a distributed SCM, which means every tester has a full copy of all history of every revision of the code, making queries against the history very fast and versatile.

## **Branch strategy**

Branching in each repository will follow a trunk-based model. This model involves

one eternal “trunk” called master from which all other branches originate. All

development work will take place on these branches and will be committed to

master via pull requests. The goal of this approach is

to have a highly stable codebase that is releasable on demand at all times.

The additional branches will be created for each automation tester who works on that repository. Each

of these branches will have specific purposes and follow strict rules as to how

they should be used. At a high level, the branching rules are described as follows:

Branch Naming Convention Example ( TBD)

Currently we name the feature branch with the name of the tester who checkouts the code.

## **Feature branches**

The Feature Branch Workflow assumes a central repository, and master represents the official project history. Instead of committing directly on their local master branch, developers create a new branch every time they start work on a new feature.

In addition, feature branches can (and should) be pushed to the central repository. This makes it possible to share a feature with other developers without touching any official code. Since master is the only “special” branch, storing several feature branches on the central repository doesn’t pose any problems. Of course, this is also a convenient way to back up everybody’s local commits. The following is a walk-through of the life-cycle of a feature branch.

## **Pull requests**

The merging of supporting branches (feature) into the master branch

will always require a pull request.

Pull requests should include all related task description in the commit comment,

using the following convention (where “Added new page under page package and utility function for Date format” is the task description and <comments> is placeholder for any optional comments.

## **Approvals (TBD)**

Best Practice is :

Completion of pull requests on master will be restricted to designated approvers.

The job of the approvers is to diligently review all code being merged to master

to ensure the utmost stability of that branch. Their aim should be to maintain

a release-ready master branch at all times.

Currently collaborator will create pull request and merge pull request by themselves

# **Code standards**

 These are the collection of standards, conventions and guidelines for designing and developing framework/scripts for java code in selenium automation.

## **Overview**

* Naming standards
* Comment standards
* Functions/Methods standards
* Automation Scripts
* Indentations
* General guidelines

## **Standards for Variables**

1. Variable names should be defined with data type abbreviation followed by actual variable name (English descriptors that accurately describe the variable/field/class/interface)

Example:

* + - 1. *int intCount*
      2. *String strMethodName*
      3. *float ftvariableName*
      4. *decimal decVariableName*
      5. *object objObjectName*

1. Domain specific terminologies should be used.
2. Mixed case should be used to make names readable.
3. Abbreviations should be used sparingly, but if it is used then it should be used intelligently and should be documented

For example, to use a short form for the word “number”, choose one of nbr, no or num.

1. Long names (<15 characters is a good tradeoff) should be avoided.
2. Names that are similar or differ only in case should be avoided.

#### **Naming Exception Objects**

The letter ‘e’ should be used for a generic exception object name.

#### **Declaring and Documenting Local variables**

1. One local variable per line of code should be used.
2. Local variable should be declared with an end line comment.
3. Declare all local variables before the functional block or in the beginning of the script.
4. Whenever a local variable is used for more than one reason, it effectively decreases its cohesion, making it difficult to understand. It also increases the chances of introducing bugs into the code from unexpected side effects of previous values of a local variable from earlier in the code.

### Standards for Parameters (Arguments) to Member Functions

Function parameters should be named following the exact same conventions as for local variable.

Example:

If Account has an attribute called balance and you needed to pass a parameter representing a new value for it, the parameter would be called accountBalance.

### Naming Classes

Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words -avoid acronyms and abbreviations.

Example: class ReadExcelFile { ..}

### Naming Interfaces

Interface names should be capitalized like class names.

Example: interface RegesterDelegate;

interface Storing;

### Naming Methods

Methods should be verbs, in mixed case with the first letter lowercase, with the first letter

of each internal word capitalized.

Example:

ResultSet getData(String query)

void initiateExcelConnection(String fileName)

### Naming Constants

The names of the variables constants should be all uppercase with words separated by underscores (“\_”).

Example:

*int MIN\_WIDTH = 4;*

*int MAX\_WIDTH = 5;*

## **Comment Standards**

Every change to the framework/scripts should be documented in modification history. A modification history should contain the following:

• Name of the person who changed the code:

• Date of change:

• Version:

• Changed function/event:

• Change description:

*Example:*

*/\*\**

*\* Project Name : Your Company Automation Framework*

*\* Author : Your Company QA*

*\* Version : V1.0*

*\* Reviewed By : Manager 1*

*\* Date of Creation : April 13, 2013*

*\* Modification History :*

*\* Date of change : 13-Sep-09*

*\* Version : V1.1*

*\* changed function : def func1*

*\* change description :*

*\* Modified By : Tester 1*

*\*/*

## **Functions/Methods**

### Function Name

Member functions should be named using a full English description, using mixed case with the first letter of any non-initial word capitalized. The first word of the member function should be a verb.

Examples:

*openAccount()*

*printMailingList()*

*save()*

*delete()*

### Naming Accessor Member Functions

### Getters:

Getters refer to member functions that return the value of a field/attribute/property of an object.

* Use prefix “get” to the name of the field/attribute/property if the field in not Boolean.
* Use prefix “is” to the name of the field / attribute / property if the field is Boolean.
* A viable alternative is to use the prefix ‘has’ or ‘can’ instead of ‘is’ for boolean getters.

Examples:

*getFirstName()*

*isPersistent()*

### Setters:

Setters refer to member functions that modify the values of a field. Use prefix ‘set’ to the name of the field.

Examples:

*setFirstName()*

### Constructors:

In Java, constructor is a member function that performs any necessary initialization when an object is created. Constructors are always given the same name as their class name.

*Examples:*

*Customer()*

*SavingsAccount()*

### Function Header

Function header should contain following detail as mentioned in the below example:

Example:

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*\* @Function Name : getValuesFromExcel()*

*\* @Description : This function is used to fetch data from excel sheet.*

*\* @param : fileName - Name of the workbook from which the test data*

*\* needs to be fetched.*

*\* @param : sheetName - Name of the workbook from which the test data*

*\* needs to be fetched.*

*\* @Return : Resultset*

*\* @Date : April, 13, 2013*

*\* @Author :  Tester Name*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

### Internal Documentation

### Function Complexity

Framework code should be designed and developed with minimal possible loops and conditions for reduced complexity and enhanced maintainability.

Each function should contain max 30 lines of code. If it crosses more than 30 lines, function should be broken in sub modules.

## **Automation Scripts**

Folder/package structure should be created based on the functionality/purpose of the scripts/files present in the respective folder.

Example:

### main source folder

Src/main/java

* com.Projectname.qa.base - This package should consists of base class which is inherited across packages
* com.Projectname.qa.listeners - This package should consists of listeners classes
* com.Projectname.qa.pages - This package should consists of all java files specific to the web element’s Ids present in the web page and domain specific functions.
* com.Projectname.qa.util - This package should contain all java files related to global functionalities

### test source folder

Src/test/java

* com.Projectname.qa.testcases - This package consists of the test scripts for all module and setup script for the test execution. Testscripts names should be decided based on the functionality and the testscript should contain group of test cases related to that functionality.

### resource source folder

Src/resource/java

* excel – This folder has testdata excel files
* executables – This folder has driver exe (Chrome, IE, edge etc)
* logs - This folder has log configurations
* properties - This folder has config.properties file where all the global/constant variables are declared.
* runner- This folder has TestNg suite xml files.

## **Indentation**

* Break after a comma.
* Break before an operator.
* Prefer higher-level breaks to lower-level breaks.
* Align the new line with the beginning of the expression at the same level on the previous line.
* If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

Here are some examples of breaking method calls:

*function(longExpression1, longExpression2, longExpression3,*

*longExpression4, longExpression5);*

*var = function1(longExpression1,*

*function2 (longExpression2,*

*longExpression3));*

## **General Guidelines**

* An ampersand (&) for concatenating strings should be used instead of ‘+’ symbol.
* Objects should be set to nothing for cleaning the memory.
* Only one variable should be declared in a line and all variable should be initialized as null/0/’ ’ while declaring them.
* There should not be more than 80 characters per line.
* The code should be properly indented.
* Variables should be declared using appropriate data types.
* Success/ failure can be logged inside sub-methods, instead of re-writing in all called places.
* Finite number of loops should be defined when we use “While” loops.
* If the first line of the method is failed, then control should move to catch block. It should not try executing second line of the method.
* Try-Catch-Finally blocks should be used for all methods.
* If similar kind of logic is used in more than one place, then reusable components should be used. Redundancy should be avoided.
* Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

# **Framework Design**

## **Framework used**

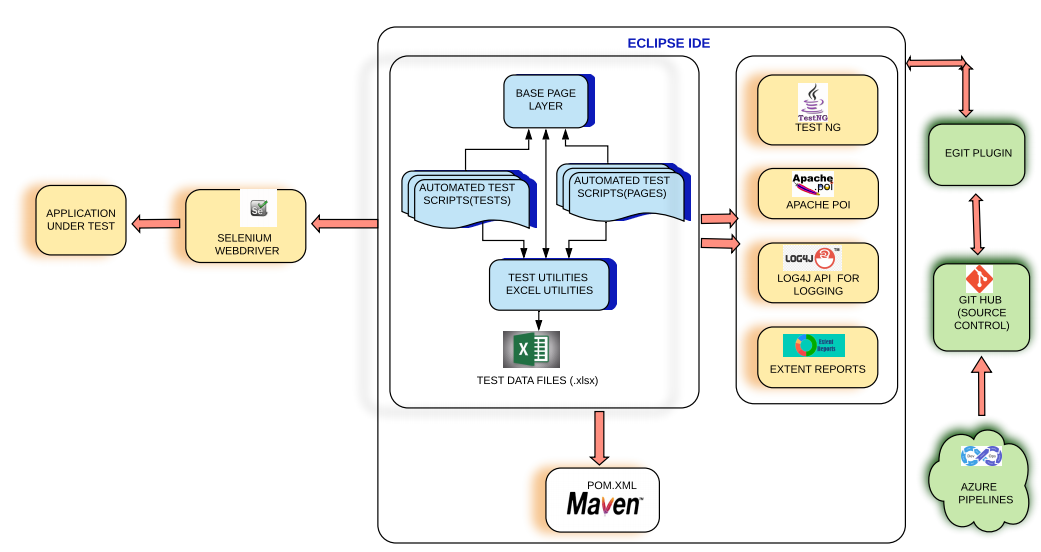
We have designed a HYBRID framework which combines the following

Selenium framework

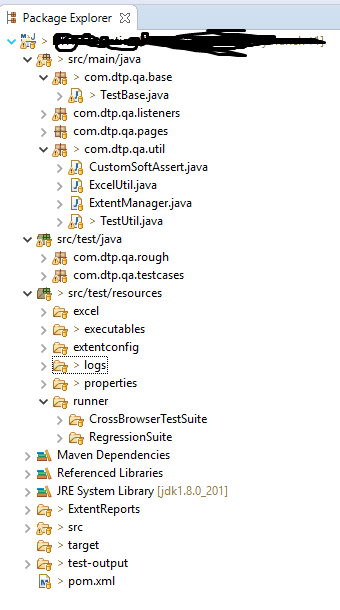
TestNG with data driven framework

POM Design Pattern

## **Framework Architecture**



**Project Folder Structure:**



# **Framework Components**

## **Selenium WebDriver API**

We are using Selenium interfaces and classes to develop our automation scripts using Java as programming language.

Selenium is software testing framework for web applications. It gives you flexibility to write code in various languages such as Java, Perl, Python, Ruby , Groovy and Scala. It’s open source and can be run on Windows, Linux or Mac machines and with different browsers like Chrome, Internet Explorer, Mozilla firefox etc. We can combine Selenium with the framework like TestNG which provides flexibility to run test cases in parallel and reduces execution time for test suite

## **TestNg**

We use TestNg Framework for our test automation.

It is an open source automated testing framework; where NG of TestNG means Next Generation. TestNG is similar to JUnit but it is much more powerful than JUnit but still it’s inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes

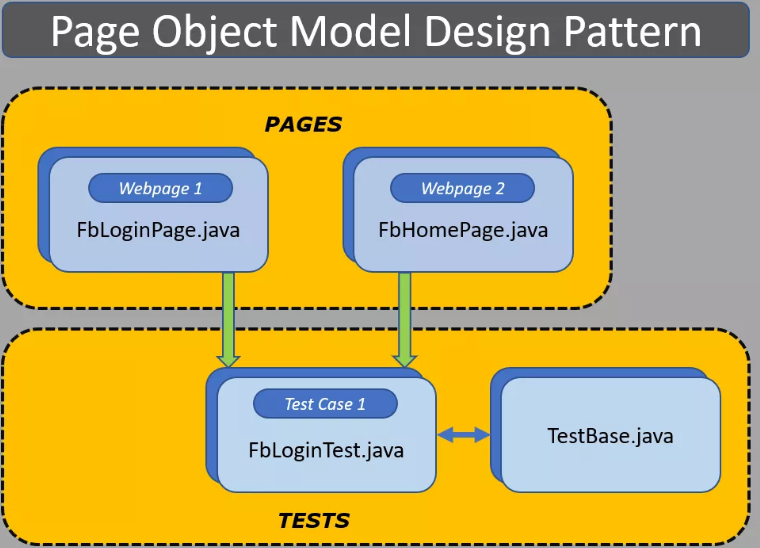
TestNG eliminates most of the limitations of the older framework and gives the test developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parametrizing.

## **Page Object Model**

We use POM design patterns for the following reasons,

Page Object Model is a design pattern which has become popular in test automation for enhancing test maintenance and reducing code duplication. A page object is an object-oriented class that serves as an interface to a page of your [AUT](https://en.wikipedia.org/wiki/System_under_test). The tests then use the methods of this page object class whenever they need to interact with the UI of that page, the benefit is that if the UI changes for the page, the tests themselves don’t need to be changed, only the code within the page object needs to change. Subsequently all changes to support that new UI are located in one place.

Increasing automation test coverage can result in unmaintainable project structure, if [locators](https://www.protechtraining.com/content/selenium_tutorial-locators) are not managed in right way. This can happen due to duplication of code or mainly due to duplicated usage of locators.



## **Build Management using Maven**

We use Maven as build management tool for our automation projects.

The build tool is used to set up everything which is required to run your java code independently. This can be applied to your entire java project. It generates source code, compiling code, packaging code to a jar etc. Maven provides a common platform to perform these activities which makes programmer’s life easier while handling the huge project.

**Build Life Cycle:**

Basic maven phases are used as below.

* **clean**: deletes all artifacts and targets which are created already.
* **compile**: used to compile the source code of the project.
* **test**: test the compiled code and these tests do not require to be packaged or deployed.
* **package**: package is used to convert your project into a jar or war etc.
* **install**: install the package into the local repository for use of another project.

## **Maven Repository**

We are not importing any external Jar files instead all the dependencies required for the project are maintained in POM.XML.

How it works:

Maven provides *pom.xml* which is the core of any project. This is the configuration file where all required information’s are kept. Many of the IDEs (Integrated Development Environments) are available which makes it easy to use. IDEs are available for tools like Eclipse, NetBeans, IntelliJ etc.

Maven stores all project jars. Library jar is in a place called repository which could be a central, local or remote repository. Maven downloads the dependency jar from a central repository. Most of the commonly used libraries are available in https://repo.maven.apache.org/maven2/

Downloaded libraries are stored in the local repository called m2. Maven uses the libraries available in an m2 folder and if any new dependency added then maven downloads from the central repository to local repository. If libraries are not available in the central repository then maven looks for the remote repository. The user has to configure the remote repository in *pom.xml* to download from the remote repository.



## **Log4j**

We use log4j as logging framework.

Create an instance for logger class: Logger class is a Java-based utility that has got all the generic methods already implemented to use log4j

Define the Log4j level: Primarily there are five kinds of log levels

* All - This level of logging will log everything ( it turns all the logs on )
* DEBUG – print the debugging information and is helpful in development stage
* INFO – print informational message that highlights the progress of the application
* WARN – print information regarding faulty and unexpected system behavior.
* ERROR – print error message that might allow system to continue
* FATAL – print system critical information which are causing the application to crash
* OFF – No logging

## **Apache POI API**

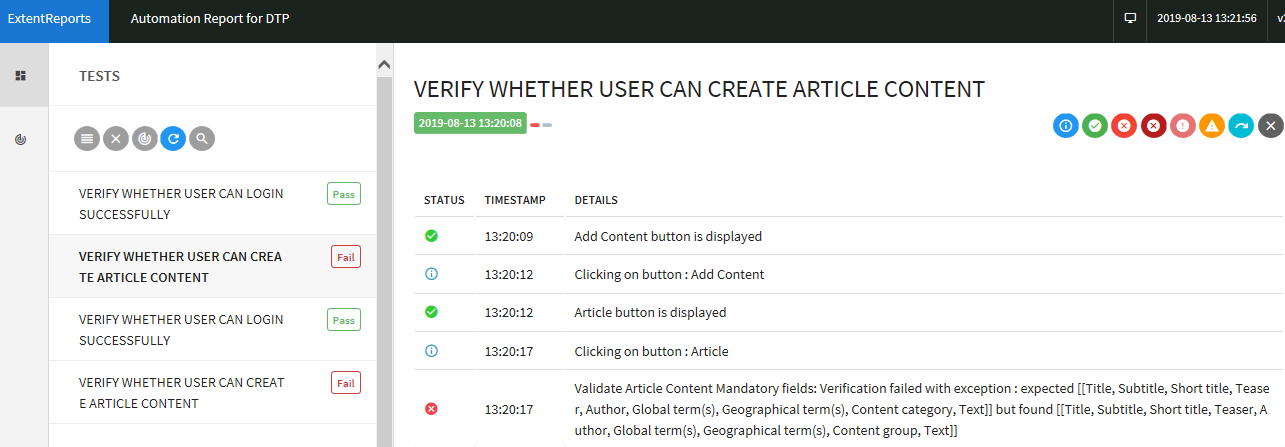
We use XSSF interface of POI

Apache POI is the most commonly used API for Selenium data driven tests. POI is a set of library files that gives an API to manipulate Microsoft documents like .xls and .xlsx. It lets you create, modify, read and write data into Excel. In POI, Workbook is a common interface for HSSF, XSSF and SXSSF

## **Extent Report for Reporting**

We use Extent reports as Test Results reporting tool even though Selenium comes with inbuilt reports using frameworks like JUnit and TestNG for the following features,

* Extent reports are more customizable than others
* Extent API can produce more interactive reports, a dashboard view, graphical view, capture screenshots at every test step, and e-mailable reports
* It can be easily integrated with frameworks like JUnit & TestNG
* It displays the time taken for test case execution



# **Continuous Integration and continues deployment (TBD)**

We will follow Azure devops process.

As part of that process QA Role is to create Azure pipelines build for the automated scripts developed for the current sprint.