**Automation Coding Standards/Best Practices Documentation - Selenium**

August 22nd 2017

Version: 1.0

Table of Contents

[Testing/Framework Definitions 3](#_Toc491179448)

[Declarations 4](#_Toc491179449)

[Code Organization 5](#_Toc491179450)

[Code Formatting 7](#_Toc491179451)

[Creation of variables 7](#_Toc491179452)

[Indentation 7](#_Toc491179453)

[Indentation (Script Level) 8](#_Toc491179454)

[Indentation (Function Level) 8](#_Toc491179455)

[Spacing 9](#_Toc491179456)

[Commenting Standards 10](#_Toc491179457)

[Script Header 10](#_Toc491179458)

[Creating Comments 11](#_Toc491179459)

[Single Line Comments 11](#_Toc491179460)

[Multiple Line Comments 11](#_Toc491179461)

[Trailing Comments 11](#_Toc491179462)

[Deleting Comments 12](#_Toc491179463)

[Indicating Flow of Execution Comments 12](#_Toc491179464)

[Naming Conventions 13](#_Toc491179465)

[Naming Scripts 13](#_Toc491179466)

[Variables & Parameters 14](#_Toc491179467)

[Constant Names 15](#_Toc491179468)

[Object Repository Standards 17](#_Toc491179469)

[Miscellaneous 18](#_Toc491179470)

[Cucumber Gherkin Coding Standards 19](#_Toc491179471)

[Function Library Standards 21](#_Toc491179472)

[Framework Asset Scope 22](#_Toc491179473)

[Framework Test Architecture 22](#_Toc491179474)

[Change Control Standards 26](#_Toc491179475)

[Exception handling Standards 27](#_Toc491179476)

[Function/Subroutine level 27](#_Toc491179477)

[Script Level 27](#_Toc491179478)

[Results Reporting Standards 28](#_Toc491179479)

[Script Level 28](#_Toc491179480)

[Project Level 28](#_Toc491179481)

[Script Execution Standards 29](#_Toc491179482)

[Peer Review Standards 29](#_Toc491179483)

[Document History 31](#_Toc491179484)

[Reviewers and Approvers 31](#_Toc491179485)

# Testing/Framework Definitions

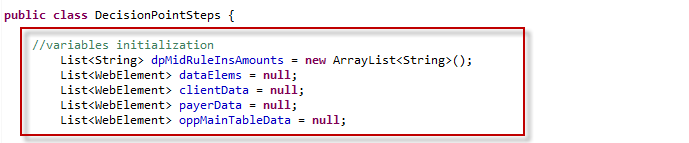
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* A Framework is a standardized, reusable test script architecture that provided specific functional testing capabilities.
* A Framework can consist of varying levels of scope and assets, however the overall structure is considered the actual Framework.
* A test case is a series of manual steps with specific instructions for a process within an application and that process expected results
* A script is the file that contains the code for automation
* A reusable function is a script that contains concise reusable elements of the application’s functionality. It is in most cases used in concert with other actions to form an end-to-end test
* A end-to-end test is a series of actions that reflect a business process
* A component is a script that contains concise reusable elements of the application’s functionality. It is in most cases used in concert with other components to form a business process test scenario.
* A business process test scenario is a series of components that reflect a test case and resides only in Quality Center.
* A parameter is a placeholder for data that is used to drive a script utilizing various sets of information
* A function is a set of code that performs a small, unchanging and highly reusable functionality within a test case. It may or may not return a value
* Constants are a storage places for values that never change.
* The Object Repository (OR) is a folder within the framework that stores the elements of an application for reference at run – or execution – time. These elements include objects taken from the GUI such as buttons, fields and tables.

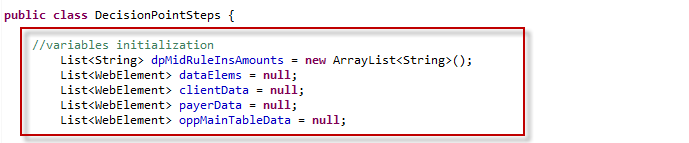
# Declarations

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* Declare all variables used throughout the script near the top of the file in the ‘Variable declaration’ section (see below example)

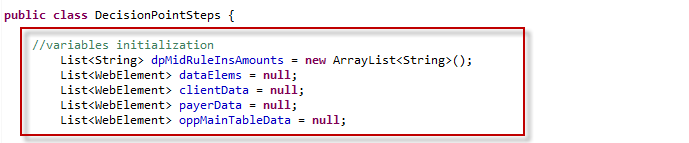
Rationale: To prevent type mismatch errors at compile time, and execute more quickly at run time through early binding. Allows for easy code readability 

* Assign all variables used throughout the script to parameters near the top of the file in the ‘Variable Assignment’ section

Rationale: Allows for a central location for parameter value assignment facilitating better code readability 

* Declare variable and assign them each on one line – no multiple line declaration statements

Rationale: Ease of maintenance and readability



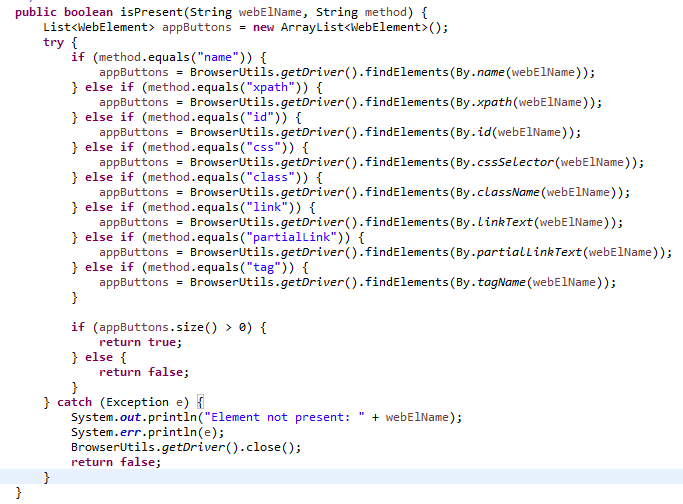
# Code Organization

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

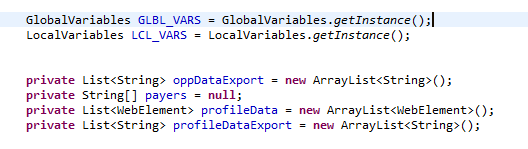
* Keep scripts to a manageable size. Each script should be less than 500 lines. If a script is becoming larger than 500 lines, consider splitting it based on some functional criteria

Rationale: Small, cohesive scripts are more manageable and easier to understand and any script that is longer than 500 lines must be reviewed for action and component reusable appropriateness.

* Keep functions to a manageable size. Each function should be concise enough that its entire purpose can be easily expressed and understood. One rule of thumb is that the well-commented function should fit on a printed page

Rationale: Small functions are easier to understand and verify 

* All related functions should be contained in an external file and be associated with its intended-use scripts(s) 
* Variable declaration and assignment sections must be in alphabetical order

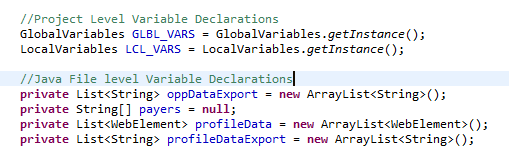
Rationale: Ease of maintenance and readability 

# Code Formatting

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Code Formatting Standards includes guidelines for creation of variables, indentation, spacing, etc.

## Creation of variables

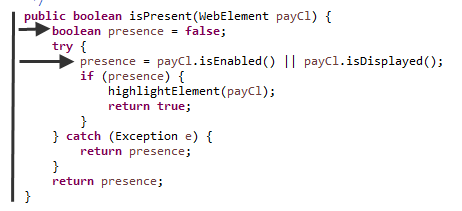


## Indentation

* Use the standard indentation provided by Eclipse in case of Java automation projects and Visual Studio in case of C# automation projects. The relevant IDEs provides the language specific indentations.

Rationale: Indentation improves readability. Using the standard tab-stop indentation improves maintainability, allowing the indent level of blocks of code to be easily changed

* Use the indent feature to notate when subsequent lines of code are related to the first, non-indented line such as with IF statements ad looping elements

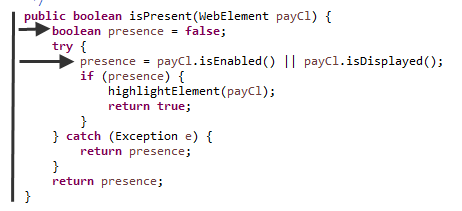
Rationale: Indentation improves readability 

* Before checking in the code to the repository, make sure that the code is properly indented, by selecting the complete java file contents and clicking on CTRL-SHIFT-F to format the complete java file to be checked in.

## Indentation (Script Level)



## Indentation (Function Level)



## Spacing

* Apply spaces liberally

Rationale: Spaces break up the code, making it easier to read

* Make use of spacing between related lines of code so that similar functionality in the script is its own block of code

Rationale: Use of code blacks or segments improves readability

* One blank line should precede any line comment

Exception: Within the header section



# Commenting Standards

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Commenting Standards includes guidelines for creating comments, deleting comments, script headers and templates

Make use of comments throughout your script. A script must include comments when the following situations are encountered – more comments are always better than fewer though so use when needed at any time

* Scripting lines of code are vague or complex
* Code that is used in formatting strings or performing calculations
* Changes of application process flow occur
* An exception to a standard is present ( after team lead approval)

## Script Header

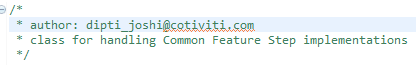
Write a header block at the top of each script. The header block will include the name of the script, a description of its purpose, prerequisites, its authors, and its revision history. It should also contain version information if appropriate. Each entry in the revision history will include the date that the change was made, the author who made the change and the reason that the change was made

Rationale: Headers help future programmers to understand and respect the purpose of the script and to consider the potential impacts before making revisions.

The following detailed header is to be used for reusable functions

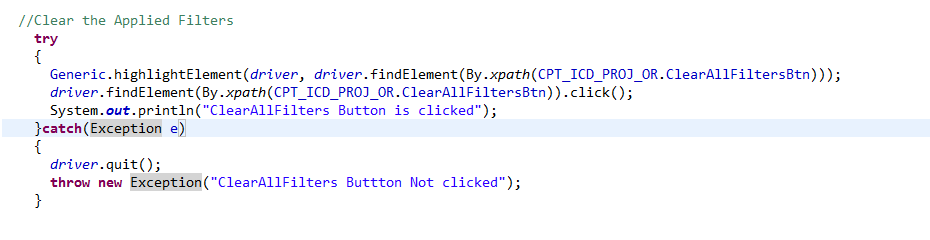
*'####################################################################################*  
*'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  
*'  Test Case ID         : Client\_Pofile\_Smoke\_Test\_CI*  
*'  Created Date         : 16-April-2015*  
*'  Created By           : Rajendra Pradeep Nethala*  
*'  Modified Date         :*  
*'  Libraries             : AppUtils.vbs, GenericUtils.vbs, GlobalVar.vbs and LocalVar.vbs*  
*'  Object Repository     : Client Profile Project.tsr*  
*'  Objective             : To Validate whether Client Profile Project is Suitable for further testing and making sure that deployment of build is successful*  
*' -----------------------------------------------------------------------------------------------------------------------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  
*'####################################################################################*

The following header is to be used for stepdef classes.

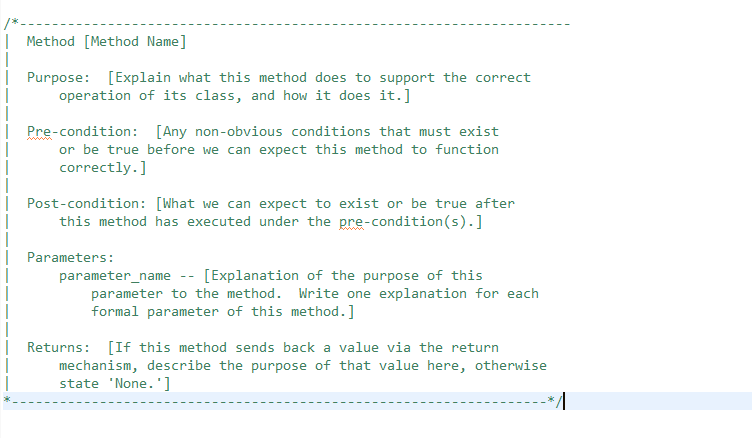


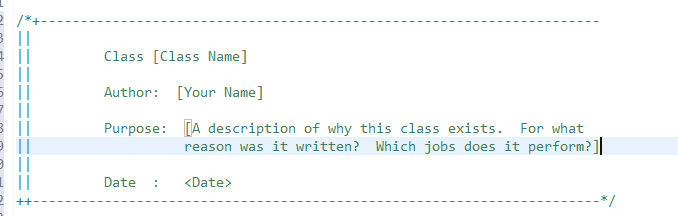
## Creating Comments

### Single Line Comments



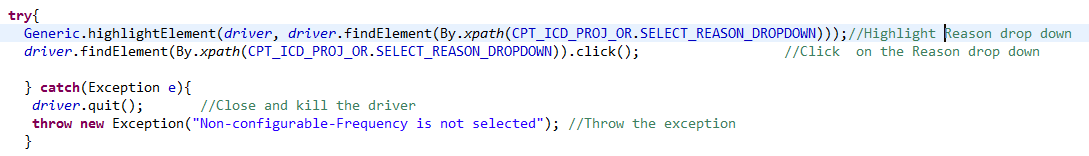
### Multiple Line Comments





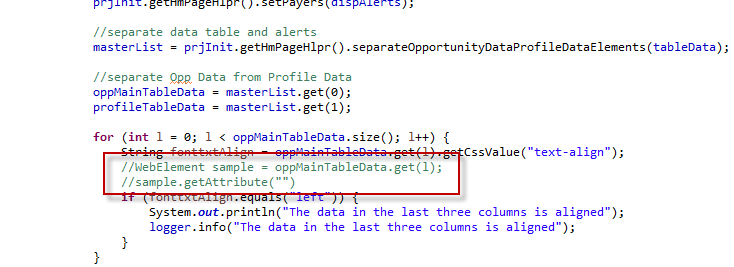
### Trailing Comments

Include short comments at the same line of the code wherever required. Keep the comments 1 or 2 tab spaces away for easy reading.



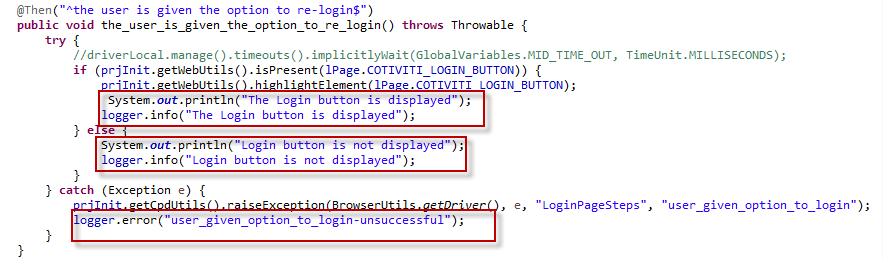
## Deleting Comments

For scripts that have been approved: when lines of code are obsolete for any reason they are to be commented out only. Do not delete lines of code that have been present for at least one round of execution, instead comment these lines out

Exception: Speak to your team lead if there is any reason why you need to delete commented lines of code 

## Indicating Flow of Execution Comments

Key points in execution need to be tracked (by putting in proper comments)



# Naming Conventions

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

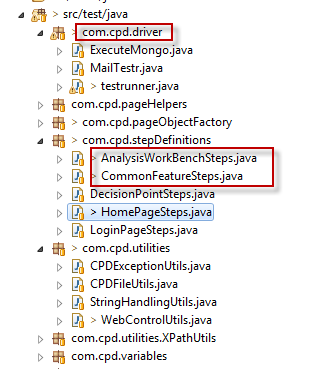
Naming Convention Standards includes guidelines for naming scripts, variables, parameters, constants and they are needed to ensure all developed assets are uniform and easy to understand in purpose.

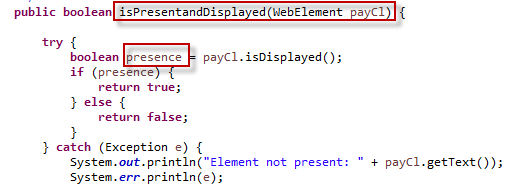
## Naming Scripts

Script and test case names should follow the structure of TC\_ [module] \_ [function performed] \_ [variation]

Rationale: This standard ensures that scripts are organized into similar functional aspects

|  |  |
| --- | --- |
| **Name** | **Convention** |
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | Should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |



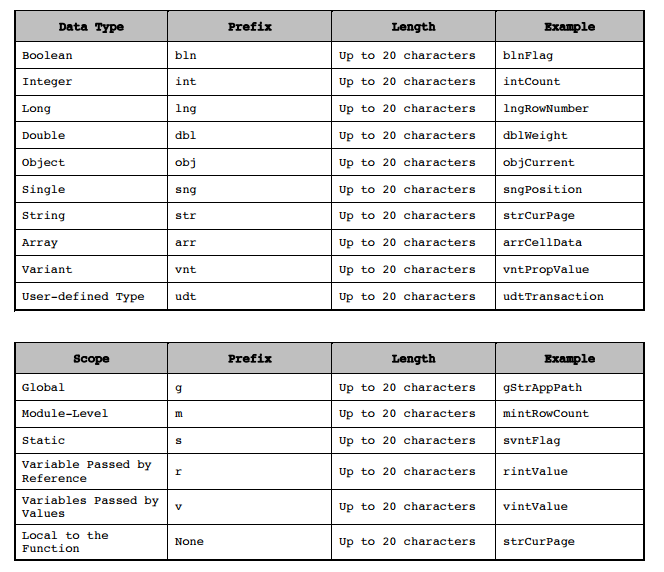


## Variables & Parameters

* Variable and parameter names should be meaningful noun or noun phrases, without spaces and must be camel notation. The names of variables and parameters (when they are to be assigned to each other) must be identical.
* No use of generic or vague variables such as “x” or “counter”

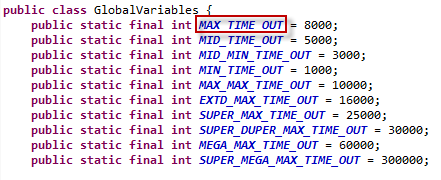
Exception: Looping statements

* All assigned values to Boolean variables must be either “TRUE” or “FALSE”
  + boolean result; ------should never be declared like this
  + boolean result = false;
* Variables and parameters used to store output parameters should have “\_Out” at the end of their names
* All array variable names must be prefixed with an “a”
* All object variable names must be prefixed with an “obj”
* All Global variable names must be prefixed with “gb”
* All Local variable names must be prefixed with “lo”
* All Private variable names must be prefixed with “pv”
* All String Parameters must be prefixed with “str” or “s”
* All Integer Parameters must be prefixed with “int” or “I”
* Use plural for collections (arrays) and singular for individual objects.



## Constant Names

The names of constants should be meaningful noun or noun phrases with all letters capitalized and underscores between words.

Rationale: Consistency with standard Visual Basic and Windows API naming conventions. This convention emphasizes and differentiates constants from variables. 

# Object Repository Standards

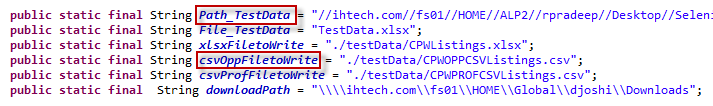
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Any object placed into the Object Repository should be named according to the field name on a screen. As web GUI entries will be captured as the html programming name, rename this to an understandable name such as GenerateRecommendationsEditBox or RecommendationsEditBox. This will then mean the GUI entry is shown in functions and test scripts as something understandable. A GUI entry in the format F032fwfn is not acceptable. Any GUI entries that are non-web based must also have an understandable name.

* In a rare case when identical objects are created in the pageObjects, add in the reference to the newly created object. For example: An object with its locator and usable methods, called Client is present in a page object, however, another similar object has to be created for some environment, say UAT then it will be wise to name the newly created object as Client\_UAT.

Rationale: These objects are redundant and cause clutter in the Page Objects

The Name of the object should be unique which can represent the object on the GUI



# Miscellaneous

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* Do not use abbreviations or acronyms unless they are extremely popular abbreviations in the project

Rationale: Abbreviations and acronyms make the code harder to understand

Exception: When needed, provide a list of acronyms to the team lead.

* Eliminate unwanted code and libraries – All the warnings for the project in Eclipse are to be handled.
* In the Java class files keep only the “**import**” package and class statements which are applicable for the objects and methods used in the particular class.
* Remove the code which is not related to the functionality being handled or logic in the Class/ Interface/Method.
* Avoid Hard Coding in the following areas as much as possible so that maintenance will be easy.
* Test Data
* Thread Sleep Times
* Popup Validation Messages
* DB Connection User Names & URLs
* File Names and Paths
* Code Performance (does it add delay far more that the application requires?)

Do not use waits/synchronization unnecessarily in the code blocks.

Ex: If there are 4 Input fields in a Web Page then as per the functionality if we need to enter the value in the first Input filed then include the Thread.Sleep or any synchronization statement to wait till the webpage loaded initially.

Then don’t include the synchronization statement before entering value into the other 3 fields. It will add unnecessary delay for the script execution.

* Code Security – No passwords should be used without encryption



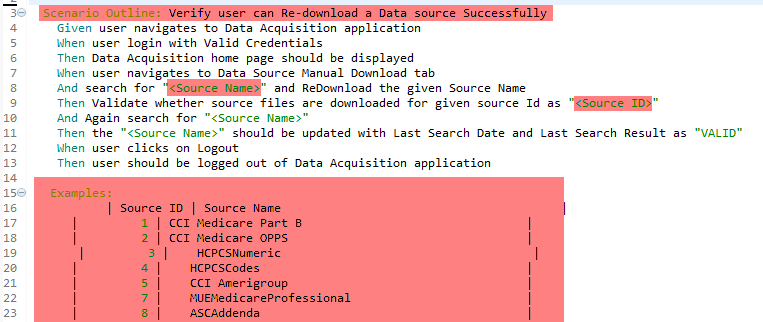
# Cucumber Gherkin Coding Standards

Code Review Instructions for Cucumber Gherkin scenarios

* **Feature**: Every .feature file conventionally consists of a single feature. A line starting with the keyword **Feature** followed by free indented text starts a feature and you can write Description you want up until the first scenario.



* Scenario Outline: Scenario outline allow us to more concisely express these examples through the use of a template with placeholders, Placeholders must be contained within < > in the Scenario Outline's steps



* + **Given**: The purpose of “Given” is to **put the system in a known state** before the user starts interacting with the system



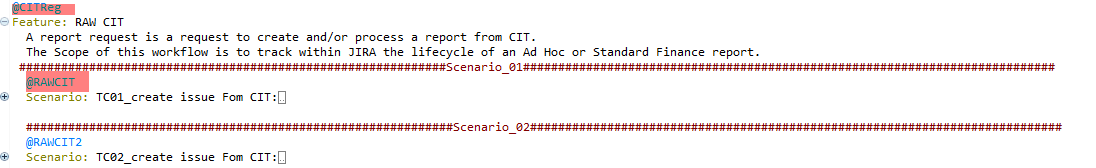
* + **When**: The purpose of “When” steps is to **describe the key action** the user performs



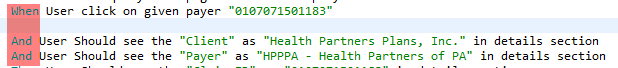
* + **Then**: The purpose of “Then” steps is to **observe outcomes**.



* + @**Tag**: Tagging can be done at Feature & scenario level and naming should justify Scenario/Feature Number /Functionality it drives



* + **And, But:** If you have several Given’s/ when’s or Then’s, then you can write And/But, To Cucumber steps beginning with And or But are exactly the same kind of steps as all the others.



* + **Background:** A Background is pretty much like scenario and the difference is when it is run. The background is run before each of your scenarios but after any of your Before [Hooks](https://github.com/cucumber/cucumber/wiki/Hooks).

# Function Library Standards

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Function Library Standards will include guidelines on the choice between a Subroutine and a Function, the scope of the procedure, variable passing and Function Library associations.

* Only use a Function when a value is returned otherwise use a method

Exception: A Function can return True or False depending on the scope of the Function

* Global or Generic Functions/Subroutines be prefixed with “gfn\_”

Example: gfn\_<Function Name> or gfn\_ < Subroutine Name>

# Framework Asset Scope

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

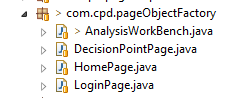
Framework Asset Scope Standards includes guidelines on the size and scope of a Framework function, action, or other piece of code

# Framework Test Architecture

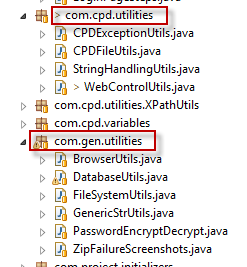
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Framework Test Architecture Standards includes guidelines on how to build a test within the framework, how to run a test, and general information on where Framework assets are located.

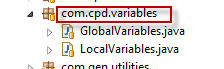
* Object Repositories must be saved to com.project.pageObjects or similar relevant location in the project and script must point to this location



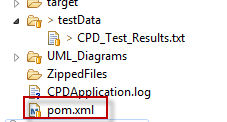
* Generic and Application Specific Functions must be saved to com.project.utilities and script must point to this location



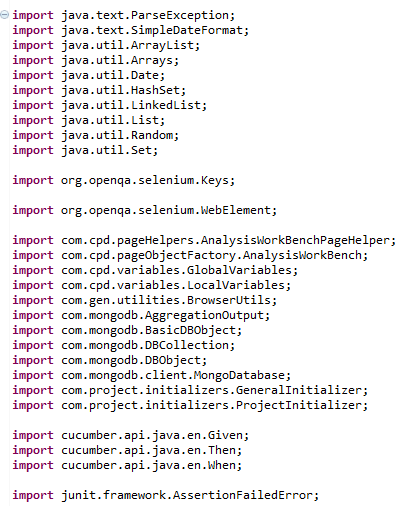
* Global and Local Variables must be saved in the Project Variables class and script must point to this location

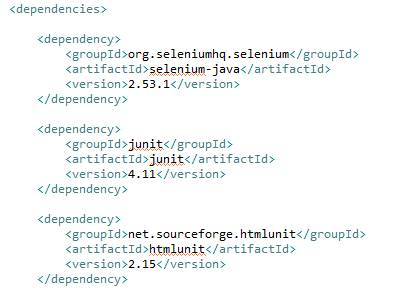


* Scripts must be executed from pom.xml defined for the project



* Library files syntax: Check the library files for syntax before applying them in scripts. Make sure that those relevant libraries are included in the pom.xml as dependencies





* Script execution interruption: The script should complete its run even if any verification/check fails. This has to be taken care through your code and ensure that the application crash or any unforeseen incidences are not breaking the script execution. Consider debugging scenarios while coding itself (Date, time stamps, Level of Pausing etc.)
* Maven settings changes in scripts: If a script needs some settings to be changed, it should be done pro-grammatically so that other scripts are not affected.
* **Approval from Architect:**
  + If there is Need to add new dependencies /Lib required other than what Rolled out
  + Change in any version in POm.xml
  + Proper justification should be provided for any structural change at Framework Level
  + Any changes to Reusable shared in the Framework should require approval

# Change Control Standards

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Change Control Standards includes guidelines on how to save and version tests as well as how to properly update framework components

* When Lines of code are obsolete for any reason they are to be commented out only. Do not delete lines of code that have been present for at least one round of execution, instead comment these lines out

Rationale: This prevents code from being lost unnecessarily

* Update Function Headers when making updates to a procedure

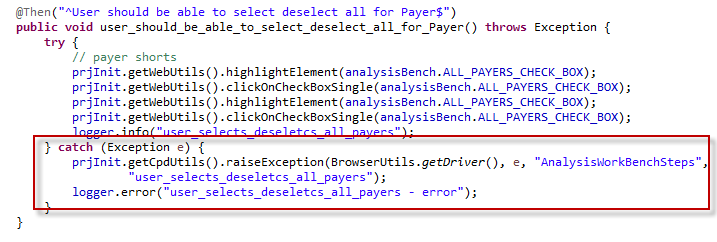
Rationale: This will help keep a change log of how/when procedures are modified ad help track down any downstream issues from the change

# Exception handling Standards

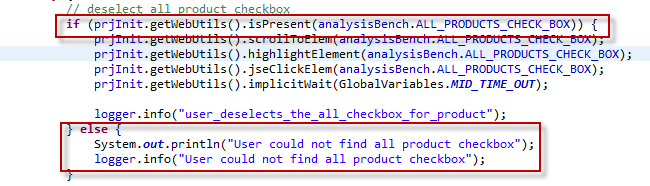
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Exception handling standards includes guidelines on how to include exception at the script, function and Subroutine while creating a Test Scenario/Script.

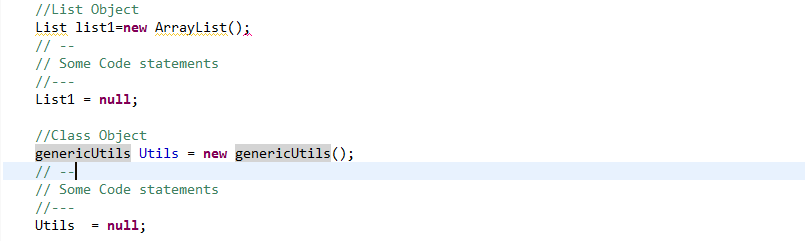
## Function/Subroutine level



## Script Level



Exception handling (Null objects, Empty objects, invalid format values and boundary conditions) need to be handled

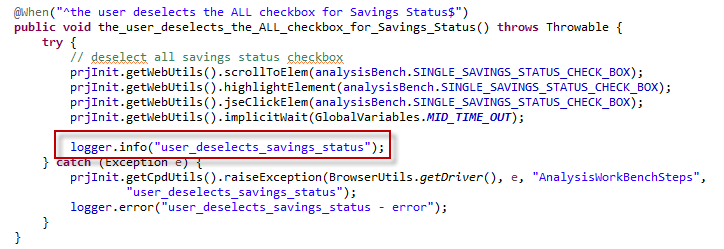


# Results Reporting Standards

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Results Reporting standards includes guidelines on how to include reporting at the script, function and Subroutine while creating a Test Scenario/Script.

## Script Level



## Project Level

In case the project takes care of any reporting functionality for the test results, those particular Java files should be placed in a separate folder like com.project.report.utilities or any other such relevant folder



# Script Execution Standards

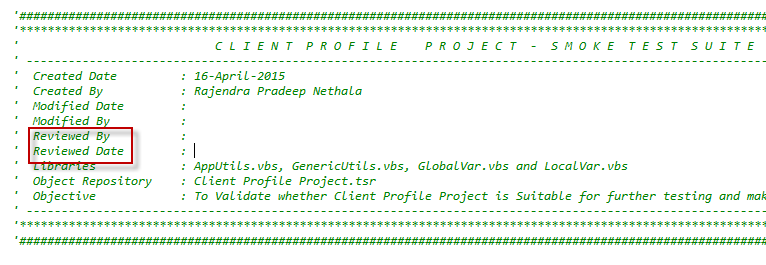
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Peer Review Standards

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Peer Review Standards includes guidelines on how to perform the code review and must to follow below template for scripts, functions and subroutines

* Update the Reviewed by and Reviewed date in the script template



Document History

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

| **Version** | **Date** | **Change** | **Author** |
| --- | --- | --- | --- |
| 1.0 | 8/22/2017 | Document Created. | Dipti Joshi |
| 1.1 | 11/6/2017 | Document Finalized | Dipti Joshi |
|  |  |  |  |

Reviewers and Approvers

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

| Version | Name | (R)eviewer, (A)pprover or (B)oth | Disposition Date |
| --- | --- | --- | --- |
| 1.0 | UdayKiran Lanka | R |  |
| 1.0 | Neelima Katari | R |  |
| 1.0 | Rajendra Pradeep Nethala | B |  |
|  |  |  |  |