

JMETER AND SELENIUM INTEGRATION



1286463 oNTARIO INC.

2016

The objectives 2

Motives 2

Test tool characteristics 2

Existing solutions 2

JBehave 2

FitNesse 2

JMeter 3

Proposed solution 4

Difference between the proposed approach and other tools. 4

Implementation 5

Solution 1 5

Implementation of the Solution. 6

AbstractExtention class 6

AbstractExtentionWebDriver class 6

AbstractExtentionKeyBoard class 7

Page interface 7

AbstractExtentionPage 8

AbstractExtentionReadFromPage 8

Concrete classes to be used from Java Sampler. 9

OpenWebPage 9

HitEnter 9

HitTab 10

TypeTextIntoPage 10

SendSpecialKeys 10

ReadTextFromPage 11

Solution 1 obstacles 12

Solution 2 12

PageNavigator 12

WebPageNavigator 13

Navigation String 13

Solution 2 for testing 14

Solution 2 obstacles 14

Solution 3 14

Separate a test from JMeter script 15

PageNavigateAndCheck 15

What is next? 17

## The objectives

The objectives of this project are to integrate several open source libraries and tools to provide an ability to create, execute and modify automated tests for The Web- and Mobile- based application to Software Quality Assurance personnel who has none or limited software development skills.

## Motives

## Currently the majority of the existing automated testing tools, frameworks and techniques require software development skills from Software Quality Assurance staff in order to automate software testing.

## For example, the Selenium framework (http://www.seleniumhq.org/) or JUnit library (http://junit.org/junit4/) provide only Application Programming Interface (API). Those tools do not have friendly Graphical User Interface (GUI) that easy to use by manual testers.

## In order to create a test using an API, a tester should write own software. Those tests become unusable for testers without development experience.

## Test tool characteristics

A good automated testing tool should provide the following capabilities to testers:

1. A testing tool should accept a test scenario, test data, and test configuration as parameters. A scenario, data, and configuration should be independent (decoupled) from each other.
2. The testing tool should be able to construct complex tests scenarios from smaller, independent units (modules).
3. The testing tool should be able to create a test execution report.
4. The testing tool should be extensible.
5. The testing tool should be easy to learn.
6. The testing tool should be affordable.

Many commercial tools and several open source projects exist to address some of those characteristics, but there is no a single solution to satisfy them all.

## Existing solutions

### JBehave

JBehave – is a framework for Behaviour-Driven Development and testing. A test or a story can be written in plain text using any text editor. The steps for the story should be implemented as Java code. The JBehave runs the story and produces a simple report about passed and failed tests.

The approach is very close to unit testing except for the test and behavioral scenarios can be produced by people with no development skills. The problem persists here – in order to create and execute a test, a tester should write Java code.

### FitNesse

Another framework is worth to mention is FitNesse. FitNesse is an acceptance testing framework. It works similar to the JBehave – an acceptance test is created as a wiki page using FinNesse wiki server. For each test, a user should add a so-called test fixture – a piece of code that will be executed by the FitNesse engine. The framework has a rich library of fixtures to re-use but they are very generic and in most cases additional development is required.

### JMeter

One more tool is necessary to mention here is JMeter. It is an open source performance testing tool designed and created by developers for a developer. The same time it is very good as a workflow engine that supports different built-in controllers. Combinations of that controller help to build complex test scenarios that can be executed in a single- or multi-threaded mode. JMeter can be used not only for performance but for functional, integration and regression testing.

The main advantage of the JMeter is – a user does not need to write a line of code to create and execute a wide range of tests.

For many years I am using JMeter (http://jmeter.apache.org/) to test Web-based applications. The tool combines many very useful features such as:

* Ability to send HTTP/HTTPS request and receive a response;
* Ability to send SQL queries to many different databases engines;
* Ability to assert response data;
* Ability to extract information from a response and inject the extraction into a next request.

One of the most important features of the JMeter – it is extensibility. In order to add a feature to the JMeter a user can make his or her own controller or create a Java Sampler.

### Selenium support

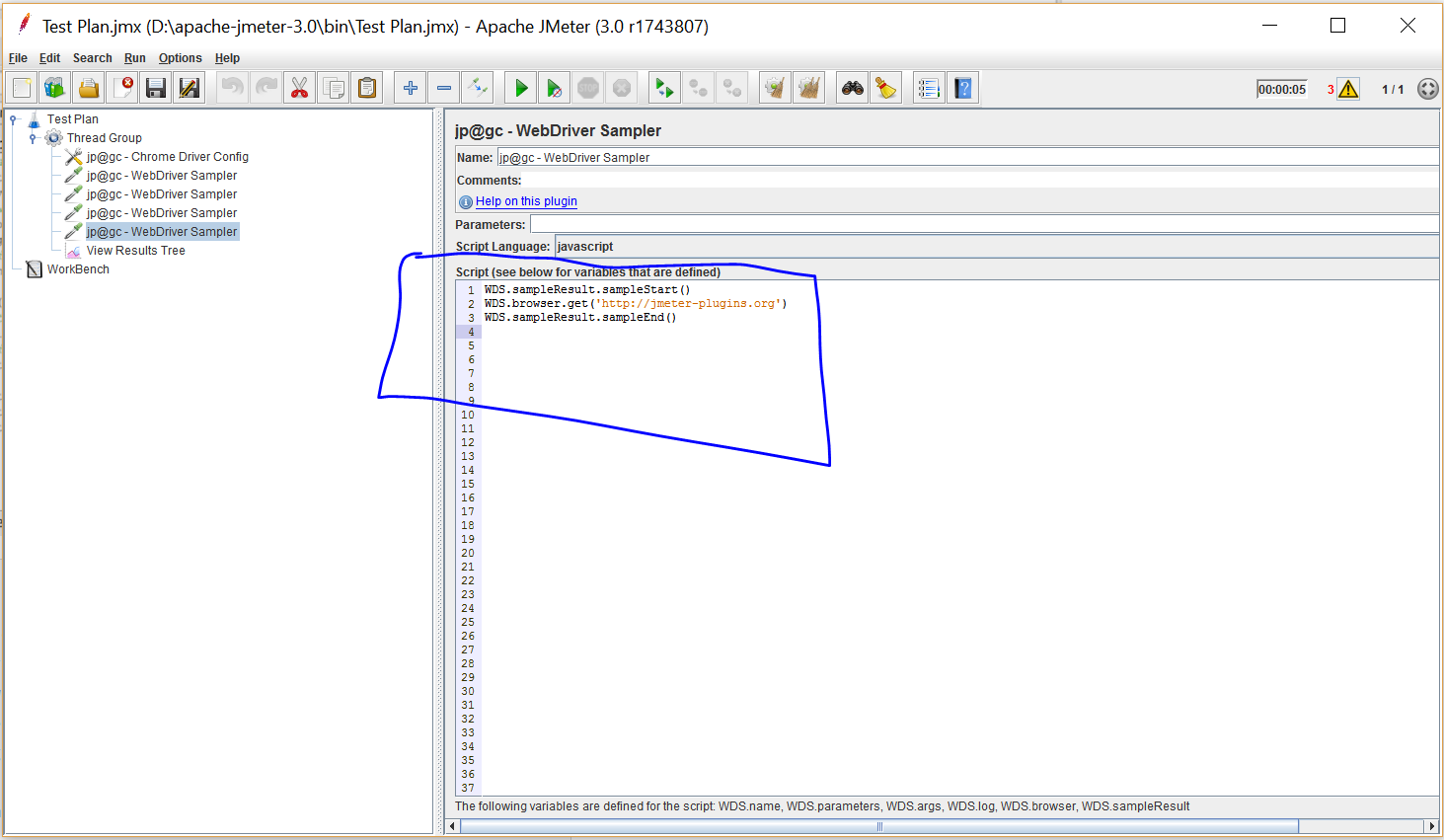
There is no a tool that has built-in support for the Selenium framework and JMeter seems a good candidate to attempt to fix this issue. Using its extensibility features, a goal to integrate Selenium and JMeter seems achievable.

There are several plugins available for the JMeter to extend its functionality.

One of them - Selenium / WebDriver plugin for JMeter (<https://jmeter-plugins.org/wiki/WebDriverTutorial/>; <http://www.ubik-ingenierie.com/blog/leverage-your-load-testing-using-jmeter-and-selenium-webdriver/>;)

This plugin makes possible to instantiate a WebDriver object using WDS - JMeter Web Driver Sampler (WDS). Then the WDS delegates control of the WebDriver to a tester via API.

A downside of it is that a scenario should be, again, a software code in plugin’s script editor field.



Finally, the main reason why WDS is not the right solution for automation is that WDS makes a test scenario and test execution code tightly coupled.

Here is an example of a WDS test from the [www.ubik-ingeniere.com](http://www.ubik-ingeniere.com):

|  |
| --- |
| var pkg = JavaImporter(org.openqa.selenium)  var support\_ui = JavaImporter(org.openqa.selenium.support.ui.WebDriverWait)  var wait = new support\_ui.WebDriverWait(WDS.browser, 5000)  var conditions = org.openqa.selenium.support.ui.ExpectedConditions  // We find the search icon by its class name "icon-search-1"  var searchIcon = WDS.browser.findElement(pkg.By.className('icon-search-1'))  // We click it  searchIcon.click()  // We ensure headersearchform element is clickable  wait.until(conditions.elementToBeClickable(pkg.By.id('headersearchform')))  // We find the search field by its id "s"  var searchField = WDS.browser.findElement(pkg.By.id('s'))  // We enter the parameter in WebDriver Sampler ${searchWord}  searchField.sendKeys([WDS.parameters])  // We find the headersearchsubmit button  var button = WDS.browser.findElement(pkg.By.id('headersearchsubmit'))  // We only start sampling here  WDS.sampleResult.sampleStart()  // We click on search  button.click()  WDS.sampleResult.sampleEnd() |

## Proposed solution

Design and build JMeter Java Sampler controllers that encapsulate complexity of different libraries including Selenuim and provides simple interfaces to design a test for a web application.

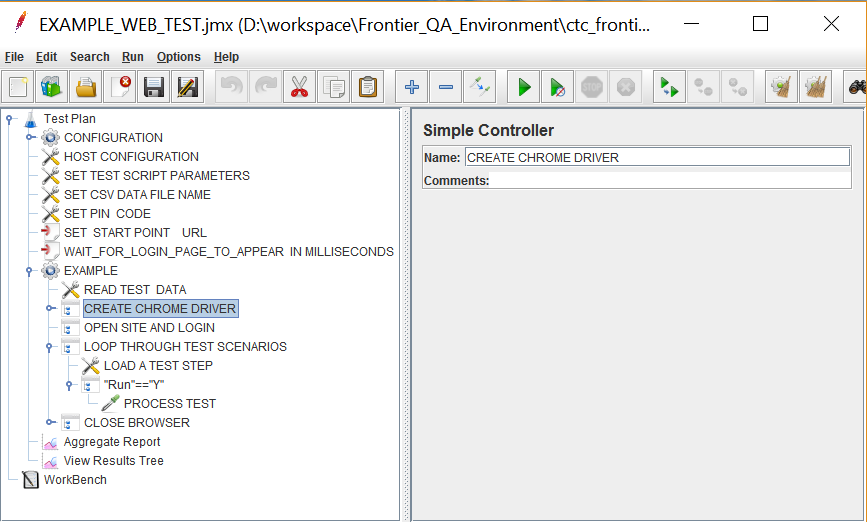
The solution should let a user to create and maintain a test or a suite of tests without any tool. A tool is required to execute those tests, for example open a web page, perform specific actions on the page, verify results. The user should be able to automate tests with no programming.

The focus of a user should be on a scenario and test data instead of wring code to verify another code.

### The Difference between the proposed approach and other tools.

The main difference between this approach and other tools is that a test scenario, test data, and configuration will be separated from each other.

The picture below illustrates how a JMeter test script should look like when scenario, data, and configuration are separated



A test scenario and test data should be CSV (Comma Separated Values) file that are processed by JMeter CSV Data Set Config controller.

### Implementation

The goal of the implementation is to hide all complexities controllin a Web Browser by Selenium from a user and the same time to release a simple and user friendly interface to create web browser based tests.

The JMeter is able to instantiate an external class if one is implementing the JavaSamplerClient interface or extends AbstractJavaSamplerClient class. That gives any external class access to the JMeter run-time environment (context) including system variables and user variables. Also, it gives control over a test execution flow.

For example, *“CreateChromeBrowser”* class provides a user easy and quick way to instantiate Chrome browser by calling it from Java Request controller. *“CreateChromeBrowser”*  class hides java code that communicates with the Selenium API to create an instance of Chrome. The same time it asks a user to specify several parameters that can be provided as JMeter variables or can be given by value in the corresponding sampler column.

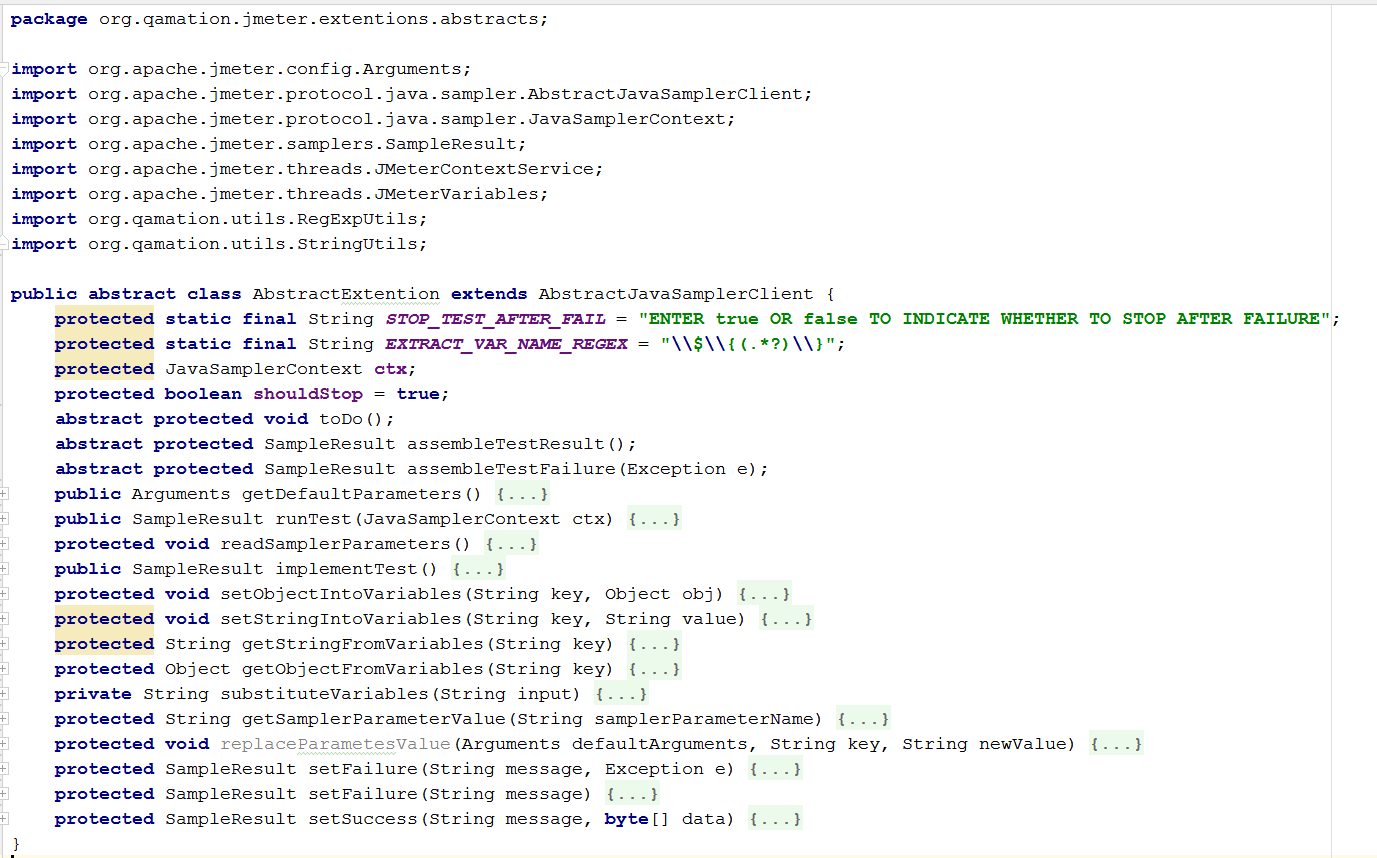
#### Solution 1

To create a dedicated class for every operation in Selenium that a user may need to use.

Several abstract classes are built to provide a user with easy access to Selenium framework features through JMeter. That abstraction has a set of methods to be used from concrete classes – it is a concrete class that will be called from JMeter Java Sampler.

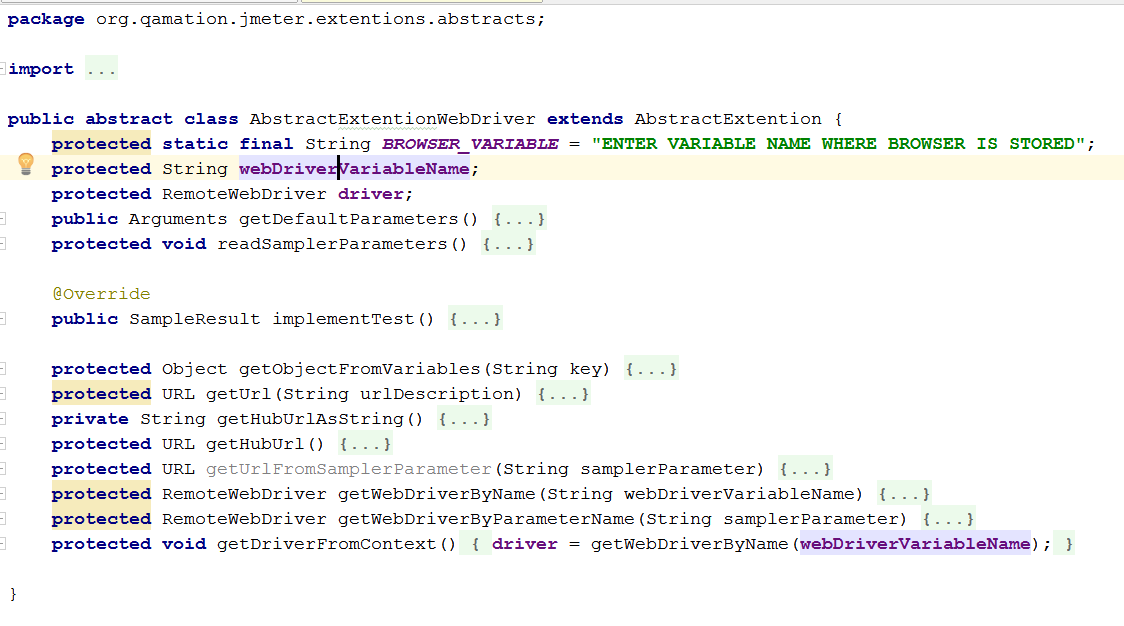
##### AbstractExtention class

The main purpose of the class is to isolate communication with JMeter run-time environment from all other functionality. AbstractExtention class extends “*AbstractJavaSamplerClient”* that implements *JavaSamplerClient* interface from JMeter API.



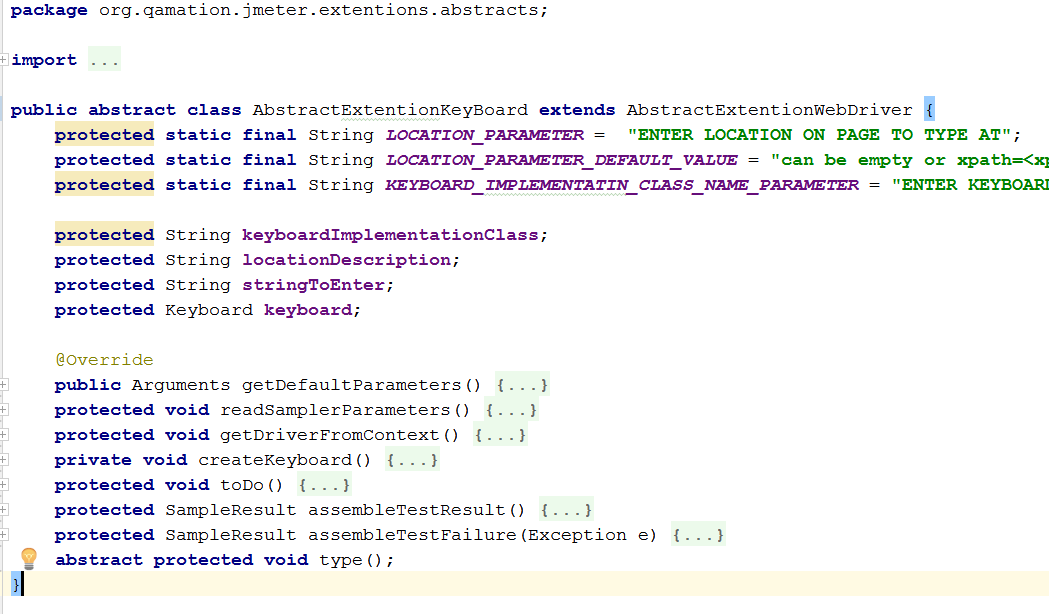
##### AbstractExtentionWebDriver class

*AbstractExtentionWebDriver* class adds another layer of abstraction to support a WebDriver instance within JMeter run-time environment.



##### AbstractExtentionKeyBoard class

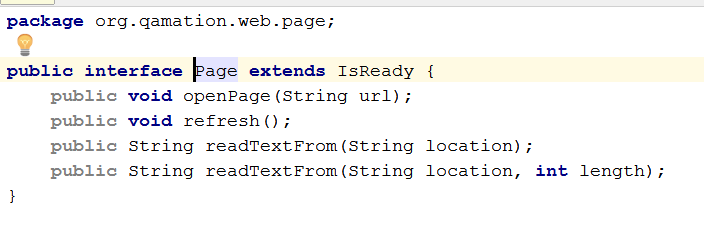
This class gives access to keyboard operations available through the Selenium WebDriver API.



Where Keyboard is an extensions from the Selenium org.openqa.selenium.interactions.Keyboard interface.

##### Page interface

Before continue exploring other abstract classes, it is important to review the Page interface.



The main purpose of this interface is to avoid the use of the Page Object Design Pattern that shifts focus from test automation to developing web page models.

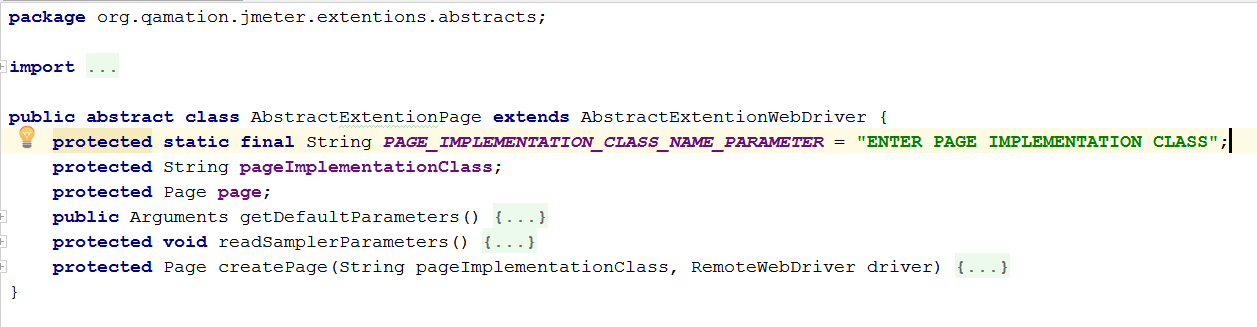
For example, an average web application may contain a lot of different pages with different content. Even grouping the pages by similar functionality will make developing and maintenance of page models a difficult task.

Instead, a web page should be described as a java interface and a user to should be able to inject page implementation into JMeter during test execution. In this solution, a web page is considered to be not more than just a source of information.

The same time in order to change the content of the page a user should do something. For example fill a form, click a mouse or press Enter key. Those actions should never be a part of a page itself.

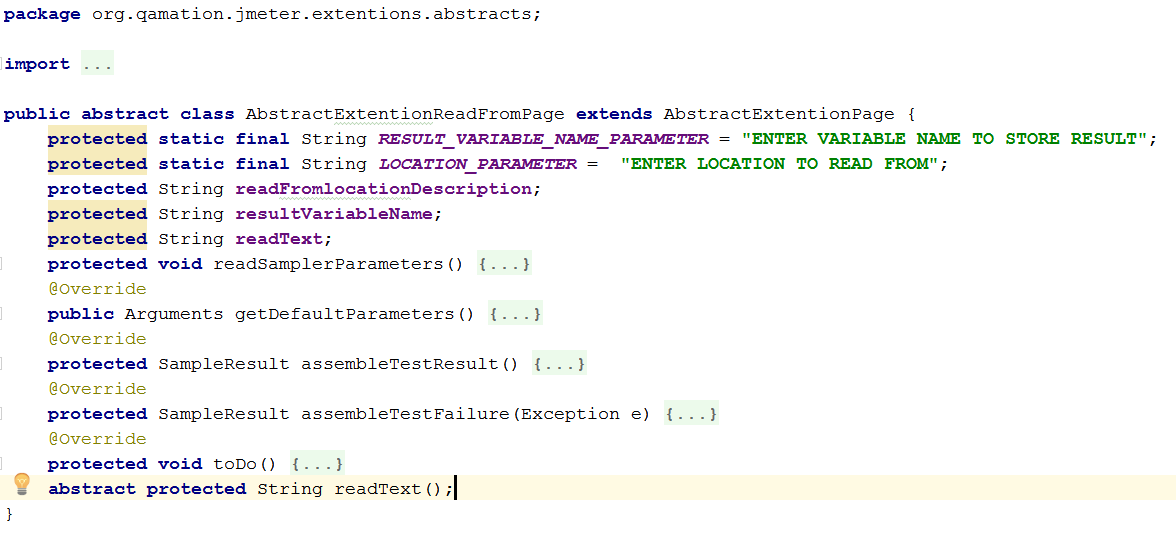
##### AbstractExtentionPage

This abstract class adds the ability to JMeter to work with a Page interface implementation.



##### AbstractExtentionReadFromPage

This abstract class adds support to read and store text information from a Page.



##### Concrete classes to be used from Java Sampler.

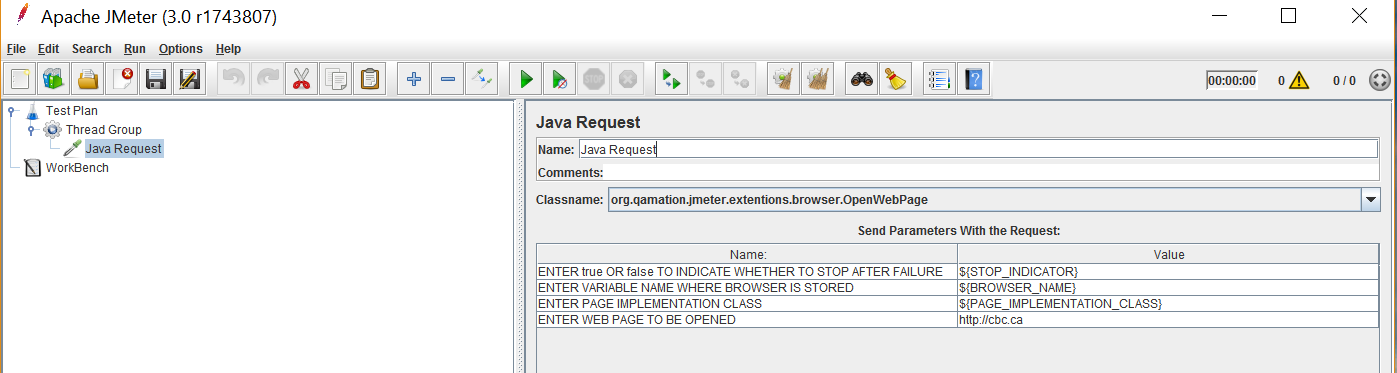
Now it is easy to implement a set of commands required for a simple test.

Here are several examples.

###### OpenWebPage

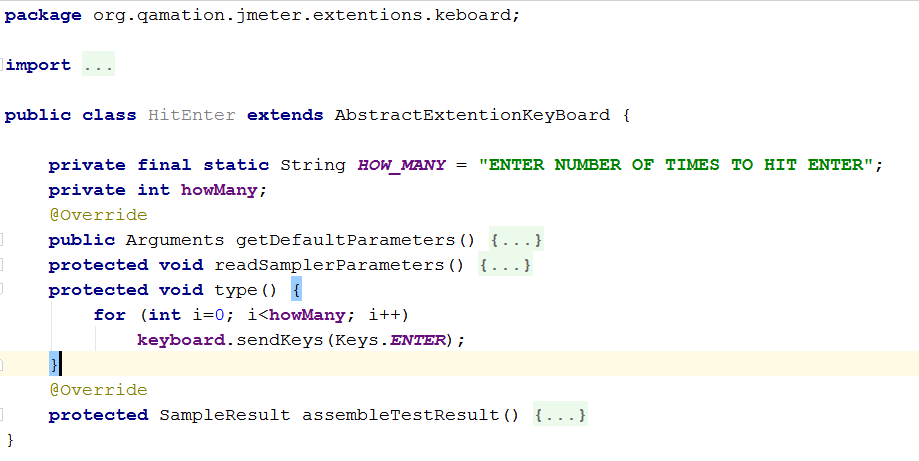
Its name describes exactly what the class is doing when called from the JMeter.

As a result:



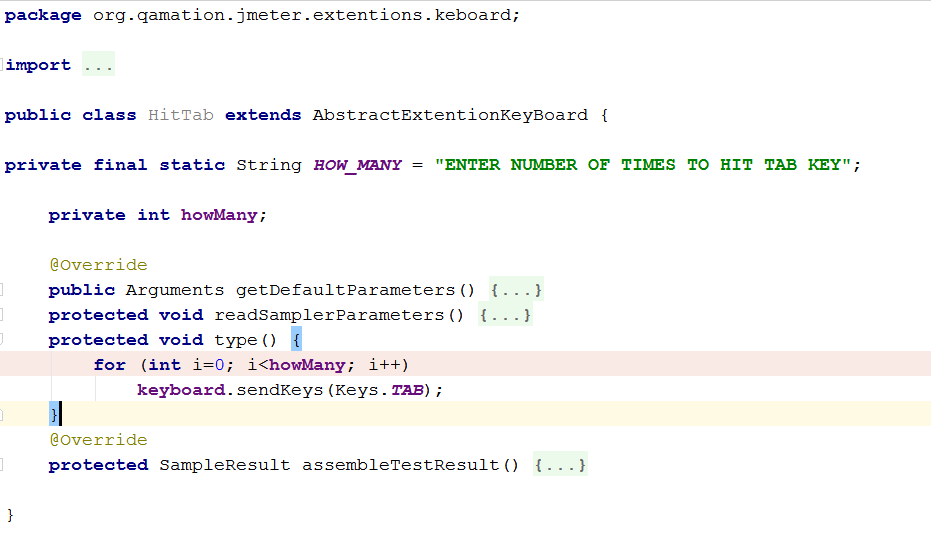
###### HitEnter

A call to HitEnter class from JMeter Java Sampler triggers keyboard Enter event. This class derives from *AbstractExtentionKeyBoard* class.



###### HitTab

Similar to HitEnter class, HitTab class triggers Tab keyboard event when called from JMeter.



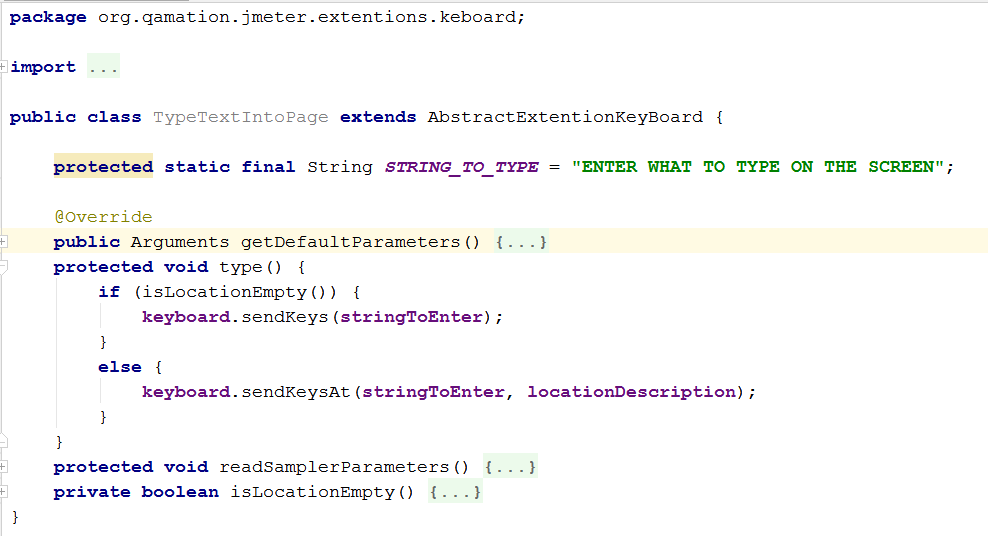
###### TypeTextIntoPage

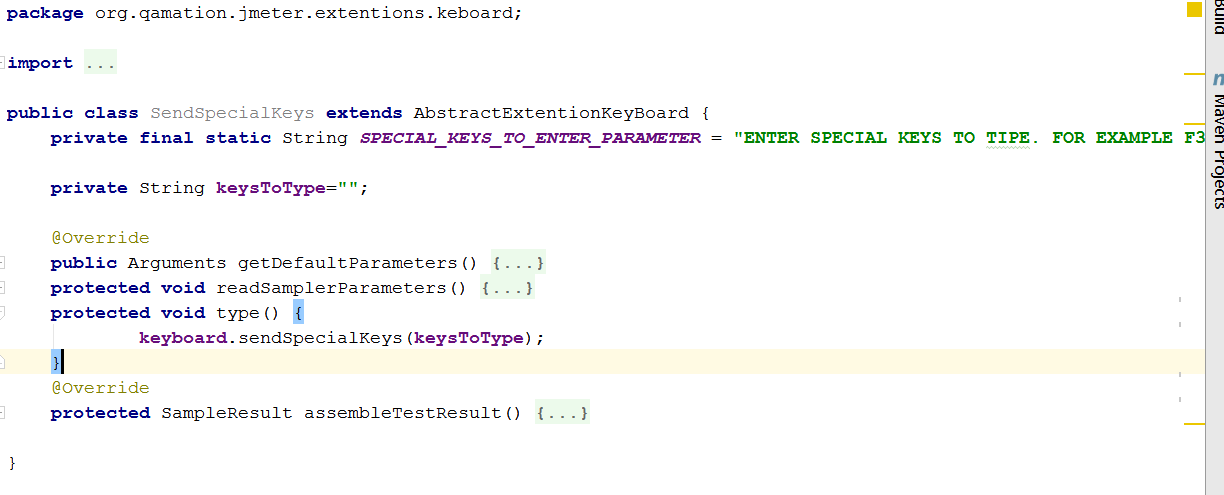
###### SendSpecialKeys

Two more classes: *TypeTextIntoPage* and *SendSpecialKeys* are created to perform more complex keyboard operations.

*TypeTextIntoPage* class provides ability to send several keyboard operations together. For example – type a text.

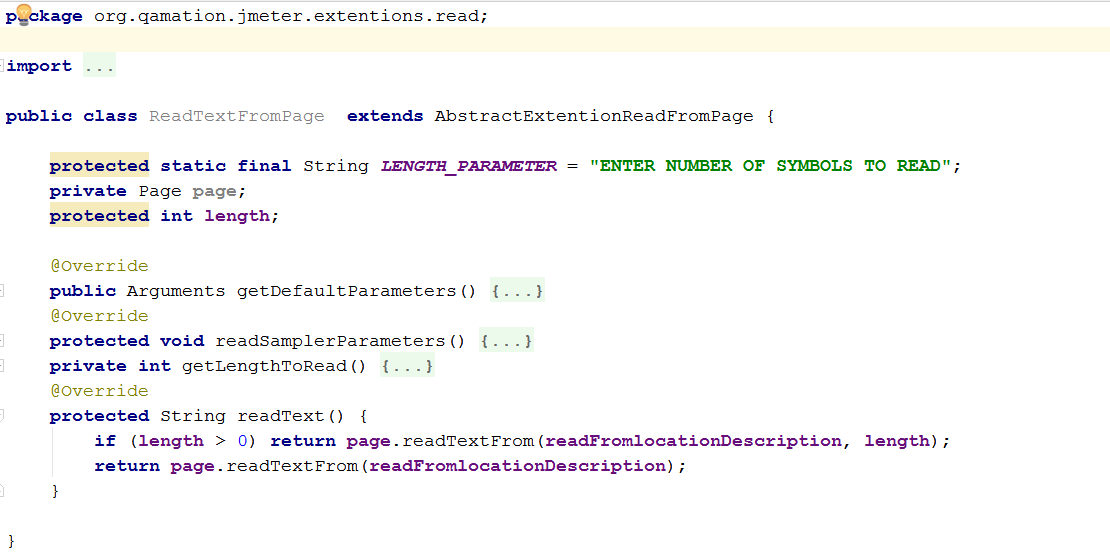
SendSpecialKeys class is similar to the previous one but support a series of “special” keys. For example, F1 or Shift and F1 together (Shift+F1).



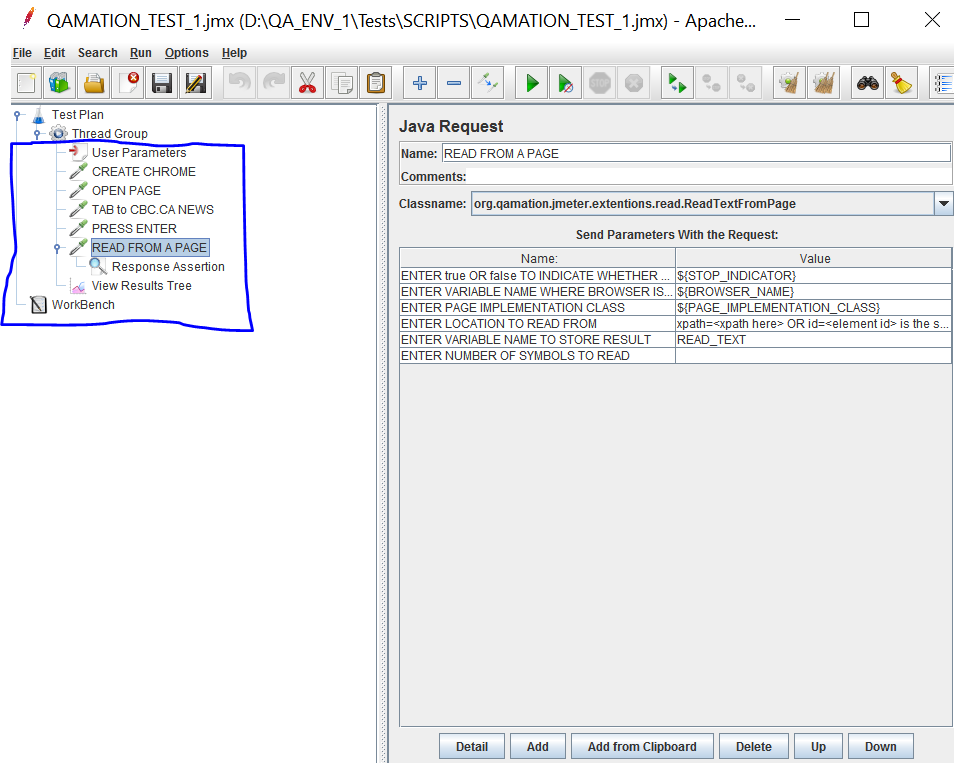


###### ReadTextFromPage

ReadTextFromPage class extends functionality of JMeter with ability to extract a value (read) from a web page.



The scenario below is simple but illustrates how a test can be constructed using JMeter extensions described above.



More complex tests may contain longer combinations of the keyboard, read and assert operations.

#### Solution 1 difficulties

The main problem of this solution is that a test scenario is stored within JMeter. This mix very quickly leads to long and unmaintainable “noodles” of steps.

An attempt to split a long scenario into smaller reusable units leads to another problem – sending test data from calling module to a called unit.

JMeter has *Include* and *Test Fragment* controllers to create modules from which a complex scenario can be formed. But, a test fragment references data by a variable name. It leads to additional value assignment operation for a test fragment calls.

Those assignments make those test units very inconvenient to use.

#### Solution 2

The second solution groups all keyboard related operations into one JMeter extension class to move browser navigation outside from a test scenario. Now the operations can be sent to execution as a string (Navigation String) for processing by *WebPageNavigator* class. *WebPageNavigator* class can be called from *PageNavigator* class that is the subclass o*f AbstractExtensionPage* described above.

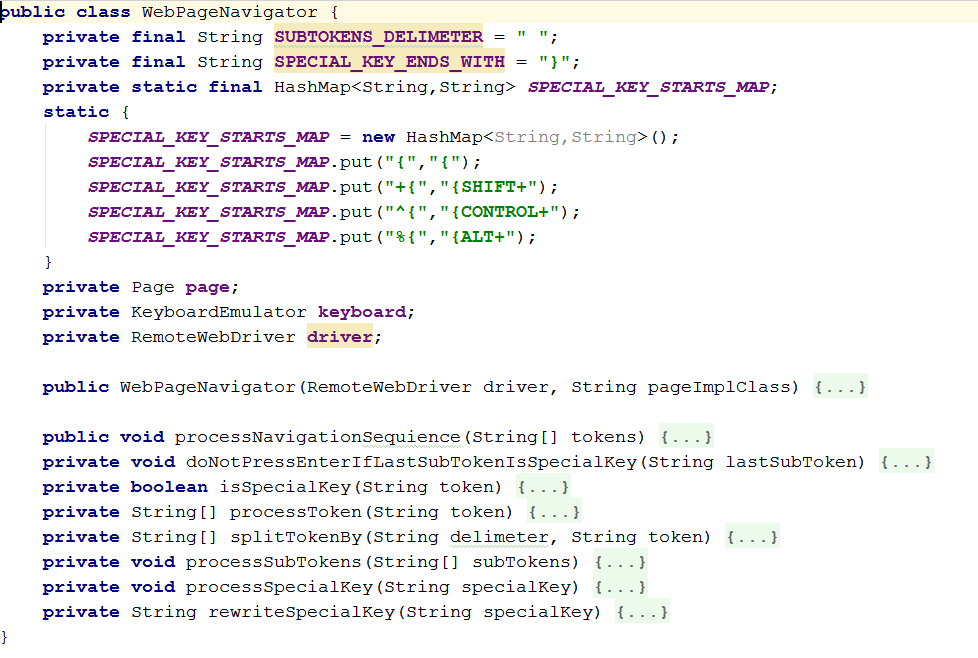
###### PageNavigator

*PageNavigator* class supports integration between JMeter and Selenium keyboard operations by using *WebPageNavigator* class.



###### WebPageNavigator

WebPageNavigator class takes an instance of *NavigationString* class (see below) and makes a browser follow the keyboard commands received with the navigation string.



##### Navigation String

Keyboard operations supported by Selenium Keys class are: (<https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/Keys.html>). They can be combined into a sequence of key names delimited by a configurable symbol, for example “.” (dot).

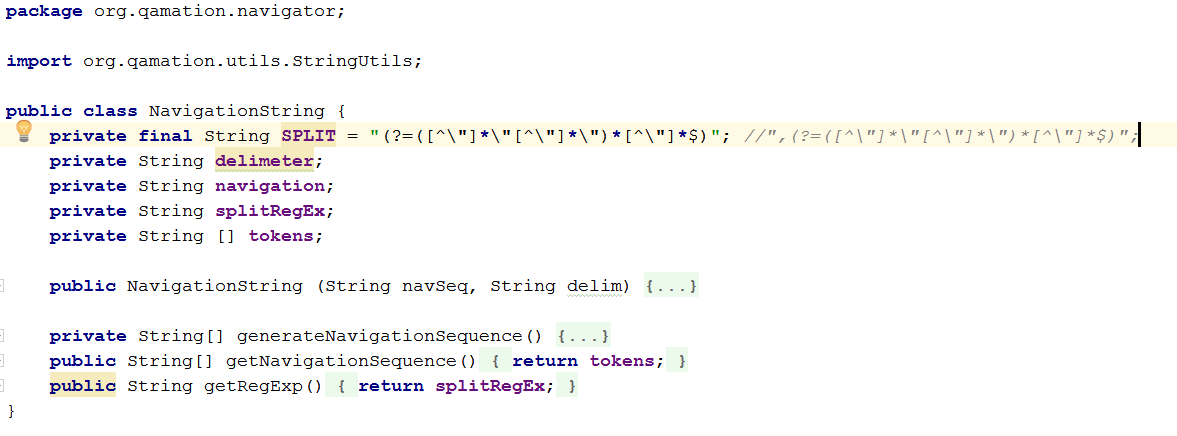
For example, the following string:

01.01.${PRODUCT\_ID} {TAB} ${ENV\_STORE\_NUMBER}

That navigation line contains three navigation tokens: 01; 01 and ${PRODUCT\_ID} {TAB} ${ENV\_STORE\_NUMBER}

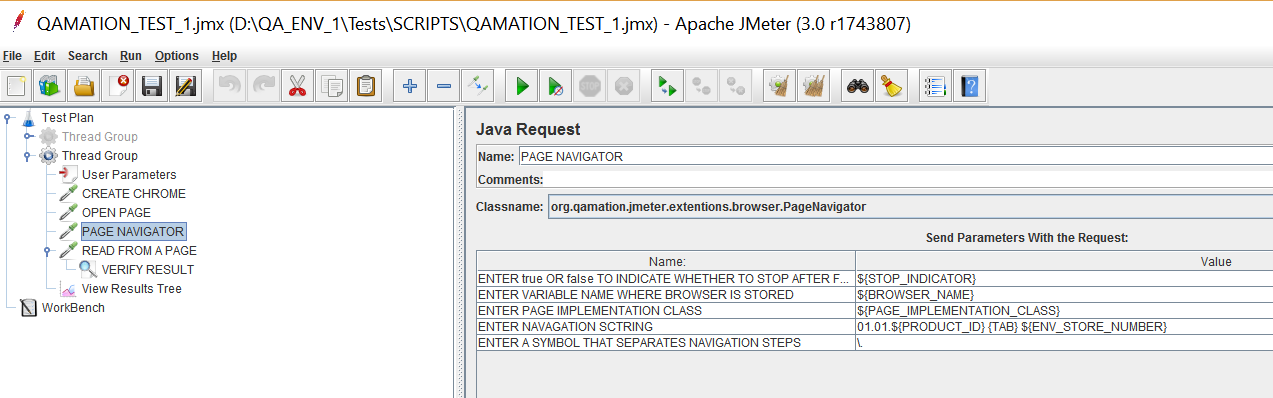
That navigation string makes browser execute the following keyboard actions:

1. type 01 and press Enter key;
2. type 01 and press Enter key;
3. type value of the ${PRODUCT\_ID} variable, press TAB key, type value of the ${ENV\_STORE\_NUMBER} variable and press Enter.



#### Solution 2 for testing

A set of JMeter extensions described above allows to wrap browser navigation portion of a test script into one node by either using *PageNavigator* directly in a scenario or hide one or several navigators into a module using Include and Test Fragment controllers built in JMeter.



#### Solution 2 difficulties

This solution makes a test scenario easy to develop and maintain, but one major problem is still here –maintenance of a test is not possible without the JMeter.

#### Solution 3

Solution 3 comes naturally after solutions 1 and 2 implementations. The first two implementations have built the foundation that helps to separate a test scenario, data, and configuration from JMeter.

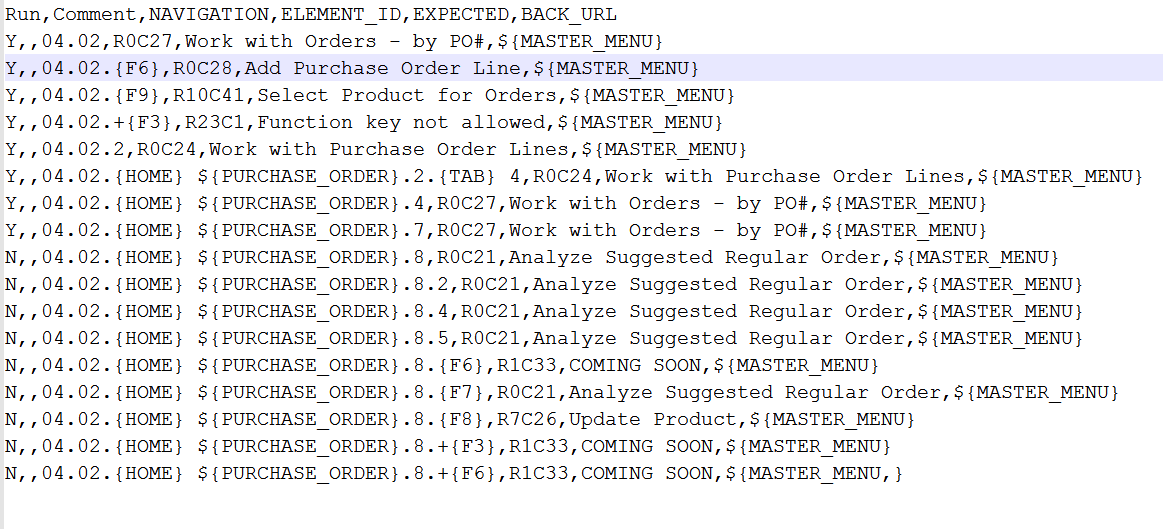
Solution 3 is using JMeter *CSV Data Config* controller and *PageNavigateAndCheck* class to read a test scenario and data from a coma separated values (csv) files. Using csv files a test can have different data sets and a single data set can be re-used by different tests.

#### Solution 3 limitation

This solution assumes that each element of a web page has an id attribute with a value matching to RxxCyy form where xx and yy one or two digit number and each element has an alphanumeric value of length 0 or 1.

##### Separate a test from JMeter script

A test for the current solution is one or several lines in a csv file:



The file should contain the following columns:

* Run: A line will be processed if a value of the column is “Y” or “y”, otherwise the line will be ignored.
* Comment: any notes about a test line.
* NAVIGATION: a navigation string for a test line. This value is optional.
* ELEMENT\_ID: a web page element that indicates a start position to read from a page. A number of symbols to read equals to the length of a string provided in the EXPECTED column. Element\_id value is optional. If the field is empty, a test will not read from the current page and will not compare the result with expected result.
* EXPECTED: a value in this column be compared with a string starting from ELEMENT\_ID. If values are not equal the test line will fail. This value is optional. If missing, no comparison will be made with a value read from a page.
* BACK\_URL: an url that is opened after the test if provided. This value is optional. The browser will stay on the current page if the value is empty.

A skipped value for one or several fields gives an opportunity to group tests related to the same page. For example, if the BACK\_URL field is empty then the next navigation string is processed starting from the current page.

Another example – multiple check points on a page. When the BACK\_URL and NAVIGATION fields are empty then a browser keeps the current page and assertion will continue using ELEMENT\_ID and EXPECTED values from the next line.

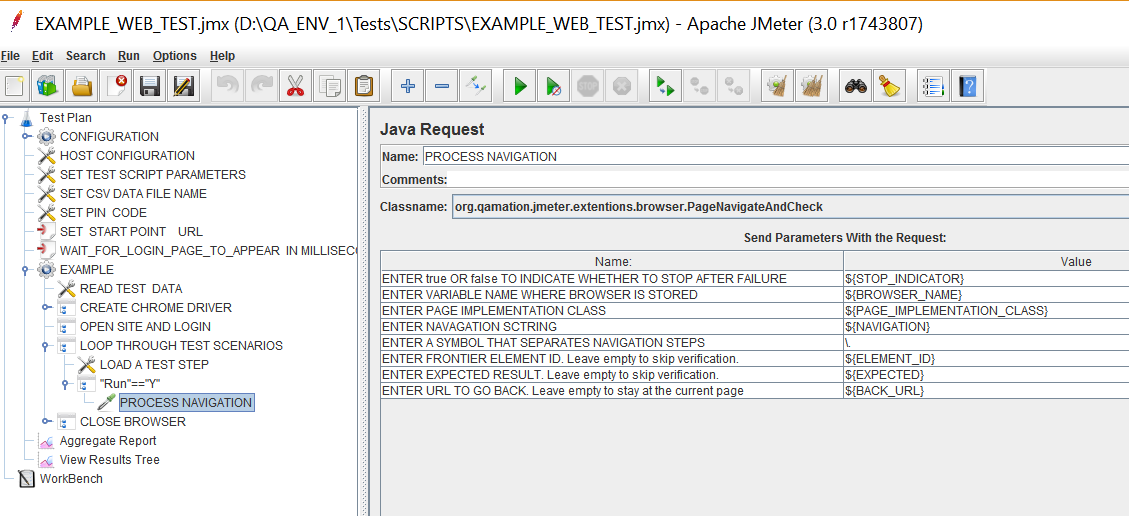
##### PageNavigateAndCheck

For each line in a test csv file the *PageNavigateAndCheck* class:

* expects that a right web page is opened;
* reads a line from the file;
* processes navigation;
* reads from the page at the specific locations;
* asserts value against expected results;
* goes to a provide url;
* can use a comment in a test.



The following picture illustrates the *PageNavigateAndCheck* class usage in the JMeter.



## What is next?

Solution 3 is used for operational validation of an enterprise level application during deployments into QA and the pre-production environments.

More than 200 tests are created by now and this number is growing.

However, this implementation limits its usage by assumptions described above. To remove that limitation, the next solution should be able to use not only a web element id but all locators provided by Selenium framework.

Solution 3 is using Java Request controller as the integration point. It is not very convenient. The better solution should convert *PageNavigateAndCheck* class into a JMeter controller. This is going to be a fun project.

Another limitation is – *CSV Data Config* controller from the JMeter. To bring the power of the Microsoft Excel documents as test data sources for the JMeter should be another interesting project.