## Methods

All methods are created as follows:

Must be in a class which inherits from the “Runner” class (C:\Projects\testSeleniumFramework\src\main\java\com \automatedTest\engine\Runner.java). The method itself is the “run” method. This must have the parameter interface:

public abstract Result run(WebDriver d, HashMap<String,String> params) throws Exception;

Methods will be either “utility” methods – see below section – or application methods – which basically perform steps in a test which cannot be easily implemented as an html test script.

On the whole “application” methods should only be created as a last resort – or to make a test more efficient.

### How to call a method from within an html test script:

All methods are listed in the RunnerList class. So look in C:\Projects\testSeleniumFramework\src\main\java\com \automatedTest\engine\RunnerList.java for a complete list. To call a method, simply use the action “callmethod” and pass in the name as defined by the name given on the left side of the listed item.

E.g:

runners.put("turn\_iip\_on", new TurnIIPOn());

runners.put("turn\_iip\_off", new TurnIIPOff());

To call the run method of the “TurnIIPOn” class, pass in “turn\_iip\_on” as the input value of your step where you are using the “callmethod” action:

action=callmethod

intput value=turn\_iip\_on

Any parameters required for a method must be set up in the preceding steps of you test using the “setparameter” action.

## Utility Methods

These are methods located at:

C:\Projects\testSeleniumFramework\src\main\java\com \automatedTest\engine\Utilities

These are created as methods rather than being a separate “action” for the action column, because they have more than one parameter. Please document all utility functions here:

|  |  |
| --- | --- |
| Utility | Details |
| executesql | Will execute a sql statement. This can be performing an update/insert/delete \*\*\*which should be used with caution\*\*\*) or return data using a select a sql statement.  **Parameters:**  **sql**-the sql statement you want to perform  **commit**- Y or N. Optional parameter which defaults to N. If Y will issue a commit sql statement after the sql statement you passed in. To be used if you are updating the database in anyway and these need to be committed. If N will not issue a commit.  **connectionstring**-the connection string to the database. E.g. jdbc:oracle:thin:@a-testdb3.dev.pretend.com:1521:dbname. If this is not passed in and the brand from the properties file is “psi” will take the connection string from the properties file value of “psiconnectionstring”.  **user**-the database username. If not passed in and the brand from the properties file is “psi” will take this from the properties file value of “psiconnectionuser”.  **password**-the database password. If not passed in and the brand from the properties file is “psi” will take this from the properties value of “psiconnectionpassword”.  **rowseparator** - optional parameter which defaults to blank (“”). If the sql statement is a “select” statement – this will be the separator that is used to separate rows of data which are returned.  **columnseparator** - optional parameter which defaults to blank (“”). If the sql statement is a “select” statement – this will be the separator that is used to separate columns of data which are returned.  If the sql statement is a select statement, this executesql method will return as output the data which is returned from the select statement. If you are expecting multiple fields and/or multiple rows of data to come back you may want to use the rowseparator and columnseparator parameters to format the data in a particular way.  E.g. if your select statement is as follows:  select product, producttype from userproducts where useremail = ‘test@test.com’ and the data returned is:  product producttype  row1: TAX SUBS  row2: ENV SUBS  with blank column and row separators the following will be output:  TAXSUBSENVSUBS.  If you pass in rowseparator of || and columnseparator of |, the following will be returned instead:  TAX|SUBS|||ENV|SUBS|||  Example script:  [..\src\test\specs\com \automatedTest\demo\kirstysql.html](../src/test/specs/com/thomsonreuters/selenium/automatedTest/demo/kirstysql.html) |
|  |  |

## Application methods

These should only be created as a last resort of for the purpose of making a commonly used script for efficient. Currently at time of writing there are no application methods.

**All application methods should be documented here. Should describe what the method does and any parameters which need to be passed in.**

|  |  |
| --- | --- |
| Method | Details |
|  |  |

## Writing new methods.

All methods needs to be implemented as the “run” method of a new class you create which inherits from the Runner class.

Utility methods should be created under testSeleniumFramework\src\main\java\com \automatedTest\engine\Utilities.

Look at existing methods for idea on how to create these.

Application methods should be created under

testSeleniumFramework\src\main\java\com \automatedTest\scriptFunctions

Look at existing methods for idea on how to create these. There is a template test which can be copied as a starting point:

testSeleniumFramework\src\main\java\com \automatedTest\scriptFunctions\ TemplateScriptFunctionToCopy.java

This template test has further info in it. Basically a “Result” object is returned which returns any Output, the Outcome (PASS or FAIL) and the ErrorMessage. Ideally the ErrorMessage returned should be an accumulation of all failures so that when the test runs, as much information about any failures is returned to the results (the ErrorMessage returned is returned to the Error Message column in the results.)

Once a new method has been written, you must add it to the RunnerList class – so update:

C:\Projects\testSeleniumFramework\src\main\java\com \automatedTest\engine\RunnerList.java