# Green Lantern Automation

**Sridevi Pinjala**
http://www.linkedin.com/in/SrideviPinjala

## Abstract

Once upon a time, the average life span for automation scripts that were recorded and played back was until the new user interface replaced the old one or until the HTML ID's were changed during a subtle application face-lift. This kind of change is expected; seldom, once every 5 or 7 years, or frequently with every build.

Major functionality additions, minor bug fixes, elusive changes to improve the response time, etc. all require the testers to spend a lot of time performing regression testing. Build verification and predictable functional tests can rely heavily on automated testing to complete the task. Automated test scripts type in fields, click on buttons, click and wait on links, select from drop down lists, verify text, logos, images present, and compare strings of text or results.

There is a simple way to create a UI/GUI repository in selenium IDE. There is a way all the UI/GUI elements can be named in English (my favorite language). Any changes to the IDs can be tracked and updated and the scripts do not have to be modified or tweaked individually ever again.

Typically, tools that have a repository map such as TestComplete, Quick Test Professional, Rational Functional Tester, etc. collect and store all the GUI elements in the object repository map. When the DOM (document object model) or the HTML ID name or other properties of an element change in an application, this change could be managed / updated via the object repository map.

Automation tools such as Selenium IDE rely on the HTML ID attribute of the User Interface (UI) and GUI (graphical User Interface) to recognize an element. They do not have object repository maps to store all the UI and GUI elements. If the HTML ID changed for any reason, the script death is inevitable. To keep-up with the changes and to avoid script death, each and every script has to be changed.

This paper introduces a concept I call "Green Lantern Automation". I chose the super hero Green Lantern because – Each Green Lantern possess a power ring and power lantern that gives the user great control over the physical world as long as the wielder has sufficient will power and strength to wield it. (Green Lantern) This concept retains the script when minor and major changes occur to the application. We automation test suite developers can control the longevity of our scripts by creating them with appropriate standards. Our standards and conventions are only as good as our ideas. Just like the Green Lantern creates objects with his will power.

## Biography

*I am a Sr. SQA Engineer at IBM. In 2011 Cynthia Gen (my reviewer) asked me to come up with a similar concept as Green Lantern Automation Framework with open source tools like Selenium and luckily, there is a way to implement the concept with Selenium IDE and also to take it to the next level. More ideas at –* http://SrideviPinjala.blogspot.com/

# Introduction

This presentation will give step-by-step instructions for creating a Test suite utilizing the Green Lantern concept. A tester can take this concept and utilize it for creating test suites for one's applications.

I will discuss the possibilities and show how a script written for one UI (Gmail) can be used for a different UI (Netflix). I will discuss the recommended naming conventions, standards and methods to be followed while creating a project suite utilizing Selenium IDE. I will also explain the advantages of following these standards and naming conventions.

First step is to specify a location where the test result pictures will be stored. I log messages with pictures rather than go through the log file and find the errors, warning or information messages.

Second step is to identify the elements (text field, link, button, page, text, check box, etc.). The developers usually assign a "logical name" to these elements. I rename the "logical names" with a naming convention.

Third step is to create a data sheet where the data is saved in HTML format. This data will be used by the test scripts as an alternative to hard coded test data.

Fourth step will invoke the URL intended for testing. I like to use one selenium test case solely to invoke the URL that can be replaced and used with other functionality.

Last but not least, the ingredient that adds value to this concept is being able to post a screen shot at every passed or failed step with a meaningful message as its name.

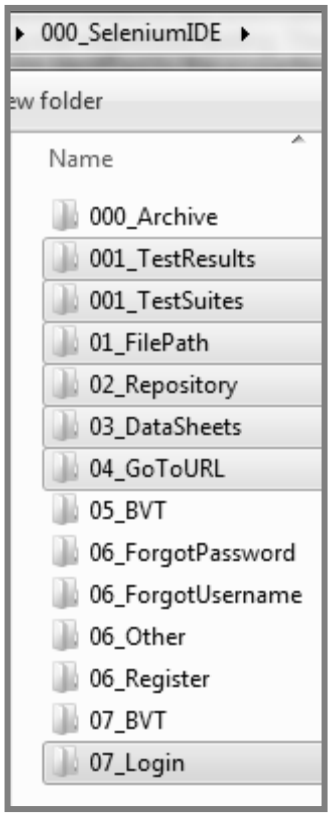
01_filePath, 02_repository, 03_dataSheet (if data is required), 04_GoToURL are the required test cases for this concept. In order to make it easy to remember and add the required test cases, I number the test cases and save them in the same numbered folder. I further numbered the functional pieces of the application too to make it easy for me to recognize them.

# 1. Installations / Software needed

Download and install –
We will utilize the Selenium IDE, Mozilla Firefox (any version as long as each is compatible with the other)
Mozilla Firefox - http://www.mozilla.org/en-US/firefox/new/
Selenium IDE - http://seleniumhq.org/download/ (download Selenium IDE latest version and any add-ons listed in the same page (optional))
FireBug for Firefox – http://getfirebug.com/downloads

# 2. Create folders –

Select a location on the local Drive and create the following folders in the order specified. The folder names that are highlighted (In **Bold**) only will be used for this tutorial.




**000_SeleniumIDE**
- 000_Archive
- **001_TestResults**
  - **07_Login_Gmail**
  - **07_Login_Netflix**
  - **07_Login_Yahoo**
- **001_TestSuites**
- **01_FilePath**
- **02_Repository**
- **03_DataSheets**
- **04_GoToURL**
- 05_BVT
- 06_Register
- 06_ForgotPassword
- 06_ForgotUsername
- 06_Other
- **07_Login**

*Figure 1 – Create Folders*

Invoke Mozilla Firefox. Go to menu – Tools > Selenium IDE. Selenium IDE comes up in record mode. Stop recording (click on the “pink” button on the top right side. Delete any lines of code that may have been recorded.

# 3. 01_FilePath

File path specifies the location for the Captured Screen Shots to be saved. Each set of screen shots will be saved to a specific location with a specific naming convention.

## 3.1 File Path for Gmail

Click in the Source area to highlight the first line. In the below fields, type in –

Command = store
Target = C:\000_SeleniumIDE\001_TestResults\
Value = first

| Command | Target | Value |
|---|---|---|
| Store | C:\000_SeleniumIDE\001_TestResults | First |
| Store | 07_Login_Gmail\ | Second |
| Store | pGmail_ | Third |
| Store | ${first}${second}${third} | FilePath |

Save the test case as "01_FilePath_Login_Gmail" at "C:\000_SeleniumIDE\01_FilePath"

## 3.2 File Path for Netflix

Copy all the lines of code from file "01_FilePath_Login_Gmail" and paste it into a new file (In Selenium IDE section Test Case, Right click and select "New Test Case".) Change the code as such –

| Command | Target | Value |
|---|---|---|
| Store | C:\000_SeleniumIDE\001_TestResults | First |
| Store | 07_Login_ Netflix \ | Second |
| Store | pNetflix _ | Third |
| Store | ${first}${second}${third} | FilePath |

Save the test case as "01_FilePath_Login_Netflix" at "C:\000_SeleniumIDE\01_FilePath"

## 3.3 File Path for Yahoo

Copy all the lines of code from file "01_FilePath_Login_Gmail" and paste it into a new file (In Selenium IDE section Test Case, Right click and select "New Test Case".) Change the code as such –

| Command | Target | Value |
|---|---|---|
| Store | C:\000_SeleniumIDE\001_TestResults | First |
| Store | 07_Login_ Yahoo \ | Second |
| Store | pYahoo _ | Third |
| Store | ${first}${second}${third} | FilePath |

Save the test case as "01_FilePath_Login_Yahoo" at "C:\000_SeleniumIDE\01_FilePath"

# 4. 02_Repository

Even though Selenium IDE does not offer an object repository (like in QTP) or object map (like in RFT) to store all the elements with their desired properties / attributes, we can "store" the elements by their HTML ID and / or Document Object Models with specific names (variables) that are easy to remember and relate to, when writing code in several test cases.

## 4.1 Store elements for Gmail

I will now capture the elements that are necessary to verify a successful or failed login for page Gmail Login.

In Selenium IDE section - Test Case, right click and select "New Test Case".
In Mozilla Firefox, navigate to Gmail login page (Gmail).

Right click on the field Username and select option "`assertText id=Email`"
Right click on the field Password and select option "`assertText id=Passwd`"
Right click on the checkbox Stay Signed in and select option "assertText id=PersistentCookie"
Right click on the button SignIn and select option "`assertText id=signIn`"
Right click on the link Cant Access Your Account and select option "`asertText id=link-forgot-passwd`"

Log in to Gmail with valid credentials.
Right click the email ID in the Gmail Logged in page and select option "`assertText id=gbgs4dn`"
Right click on the button Log out and select option "`assertText id=gb_71`"

Log in to Gmail with invalid credentials.
Right click on the error messages in red and select option "`assertText`"

| Command | Target | Value |
|---|---|---|
| assertText | id=Email | |
| assertText | id=Passwd | |
| assertText | id=signIn | |
| assertText | id=PersistentCookie | |
| assertText | id=link-forgot-passwd | |
| assertText | id=gbgs4dn | |
| assertText | id=gb_71 | |
| assertText | Enter your email address. | |
| assertText | Enter your password. | |
| assertText | exact:The username or passwor... | |
| assertText | link=exact:? | |

*Figure 2 – 02_repGmail file before converting to a repository file*

Save the test case as "02_repGmail" at "C:\000_SeleniumIDE\02_Repository".

### 4.1.1 Store

In Windows Explorer and go to location – "C:\000_SeleniumIDE\02_Repository". Open file "02_repGmail" in Notepad (or WordPad also suffices). Replace text "`assertText`" and "`assertTitle`" with "`store`" via find and replace (Ctrl + h). Replace anything in column – Command to "`store`".

### 4.1.2 English names

Open the file again via Selenium IDE. To the HTML ID of field username (`id=Email`), specify Value as "`fUsername`", meaning – f = field, Username = username. I find it easy to remember the elements I use for coding when I name them in English as they would appear on the webpage along with subtly specifying the element type.

For more operational naming, I usually specify the page name too like – `pGmailLogin_fUsername`, meaning – p = `page`, `GmailLogin` or `GL` (to keep the names short while making sense) = Gmail Login, f = field, Username = username. There is no right or wrong way to name the elements as long as they make sense to the person coding the test cases.

### 4.1.3 Naming convention

```
p = Page,                          logo = Logo
f = Field                          img = Image
b = Button                         m = Menu
ddl = Drop Down List               sm = Sub Menu
cb = Check Box                     sec = Section of a page
rb = Radio Button                  err = Error
l = Link                           tab = Tab
t = text                           cap = Caption
d = Dialog Box
```

These are the most commonly used shortcuts by me to specify the element type. I habitually abbreviate the page names but only occasionally I abbreviate the element names.

The names are short or long based on the hierarchy. Example for short names: "`pGL`" is short and meaningful enough for our test cases. If we also were dealing with page General Ledger, I would name the page – `pGenLed` to be able to differentiate easily. Example for long names: "`pGL_dLoginFailure_bOK`." Or "`pGenLed_dLogFail_errUsernameAndPasswordNoMatch`"

Name the rest of the elements / attributes as discussed.

### 4.1.4 Sort Order

Sort the column = Value alphabetically. This test case file will acquire many additional elements each day. It would be difficult to keep track of the elements when there is sudden need to change or add or even look for, if particular elements already exist. So, sort all the lines in alphabetical order in the Value column to look like the below picture. A little effort will save a lot of time and sanity in the long run.

| Command | Target | Value |
|---|---|---|
| store | id=signIn | bSignIn |
| store | id=gb_71 | bSignOut |
| store | id=PersistentCookie | cbStaySignedIn |
| store | id=Passwd | fPassword |
| store | id=Email | fUsername |
| store | link=exact:? | imgWhat |
| store | id=link-forgot-passwd | ICantAccessAccount |
| store | id=gbgs4dn | tUsername |
| store | Enter your email address. | tUsername_Error |
| store | Enter your password. | tPassword_Error |
| store | id=errormsg_0_Passwd | tUsernamePassword_NoMatch |

*Figure 3 – 02_repGmail file after sorting the given names*

## 4.2 Store elements for Netflix

In Selenium IDE section Test Case, Right click and select “New Test Case”.
In Mozilla Firefox, navigate to Netflix login page (Netflix).
Capture all the elements on the page. Log in with valid credentials and add the page title and link / button Logout.
Save the test case as “02_rep_Netflix_HomePage” at “C:\000_SeleniumIDE\02_Repository”. Following the steps discussed in ‘Store elements for Gmail’, name the attributes and sort.

| Command | Target | Value |
|---|---|---|
| store | id=login-form-contBtn | bSignIn |
| store | link=Sign Out | bSignOut |
| store | //form[@id='login-form']/di... | cbStaySignedIn |
| store | id=password | fPassword |
| store | id=email | fUsername |
| store | xpath=(//a[contains(text(),'Cl... | lCantAccessAccount |
| store | link=Click here. | lRegister |
| store | //form[@id='login-form']/di... | lWhatsThis |
| store | link=Sridevi Balla | tUsername |
| store | css=#aerrors > ul | tUsernamePassword_NoMatch |

*Figure 4 – 02_repNetflix file after sorting the given names*

## 4.3 Store elements for Yahoo

In Selenium IDE section Test Case, Right click and select “New Test Case”.
In Mozilla Firefox, navigate to Yahoo login page (Yahoo).
Capture all the elements on the page. Log in with valid credentials and add the page title and link / button Logout.

Save the test case as “02_rep_Netflix_HomePage” at “C:\000_SeleniumIDE\02_Repository”. Following the steps discussed in ‘Store elements for Gmail’, name the attributes and sort.

| Command | Target | Value |
|---|---|---|
| store | id=signUpBtn | bCreateAccount |
| store | id=.save | bSignIn |
| store | id=yui_3_2_0_7_134025709554... | bSignOut |
| store | id=persistent | cbStaySignedIn |
| store | id=passwd | fPassword |
| store | id=username | fUsername |
| store | id=.save | lCantAccessAccount |
| store | css=span.yuhead-name | tUsername |
| store | css=div.yreqertxt | tUsernamePassword_NoMatch |

*Figure 5 – 02_repYahoo file after sorting the given names*

**Note**: I did not specify the '`pGmail_`' or '`pYahoo_`' or '`pNextflix_`' for any of the attributes because, I am demonstrating the reuse of scripts with multiple yet similar user interfaces. My objective here is to show how script death can be minimized when the user interface changes.

# 5. Documentation (optional)

Take some time to document the elements in any kind of file that you would like. I prefer to document the naming convention of each application and its pages in Excel spreadsheets. Specify values to the rest of the attributes too using the suggestions discussed. The end result would look somewhat like the picture below.

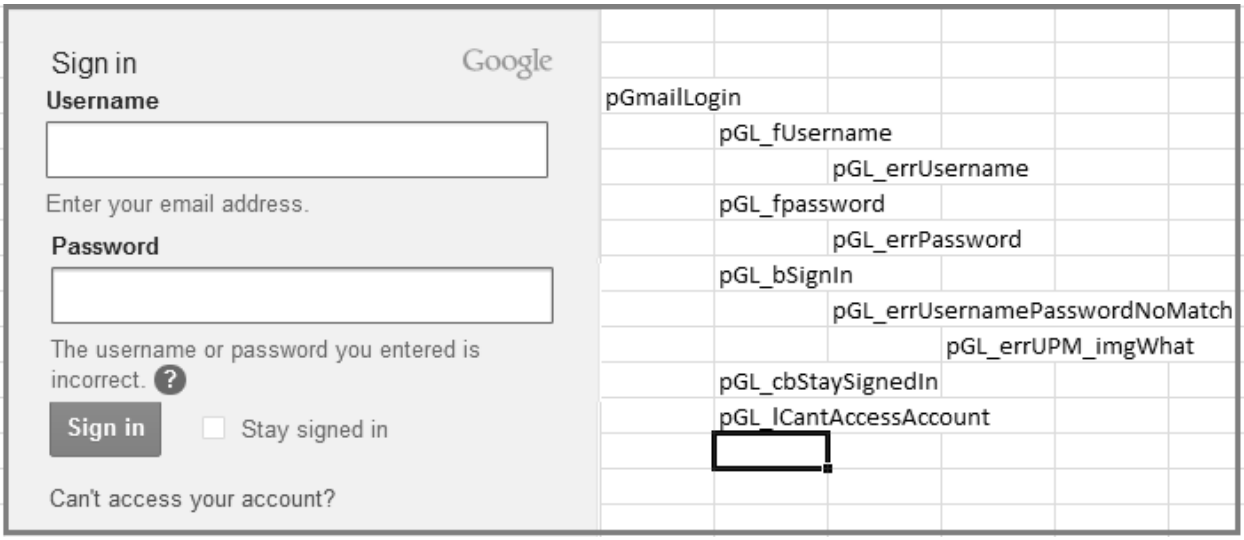



*Figure 6 – 02_repGmail file after converting to a repository file*

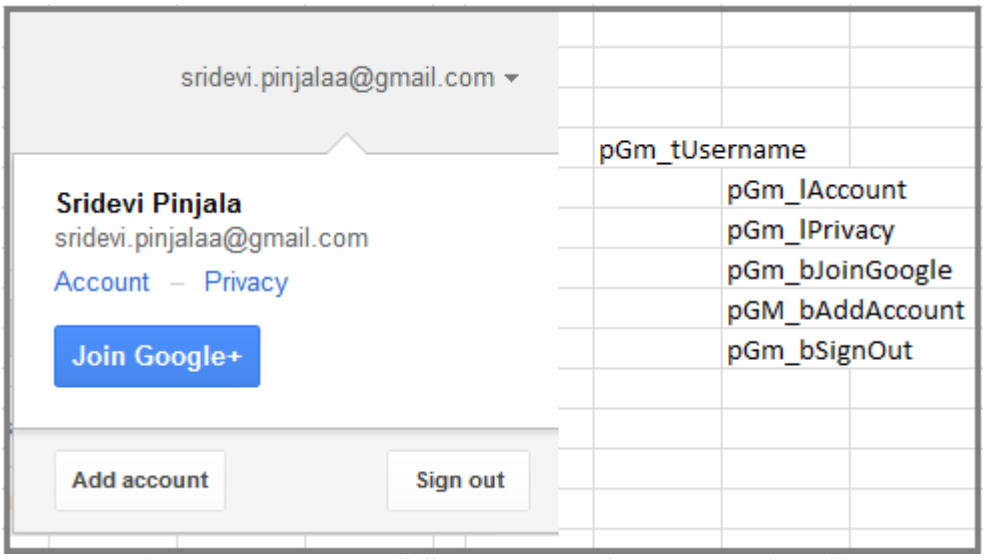



*Figure 7 – 02_repGmail file after converting to a repository file*

The span of the name does not matter, its uniqueness matters. The purpose for naming is to recall the element and utilize them promptly. Meaningful names will also eliminate the possibility of naming two attributes the same. Maintaining the given names in the hierarchical order in the Excel spreadsheet, gives me clarity about the requisite elements needed to be tested before utilizing / testing an element. (Example: Unless the page Gmail is invoked, the login test case cannot be executed)

## 5.1 More about 02_Repository

### 5.1.1 02_rep(page)

Create a repository file (page) to store all the elements per page. Some pages have just 2 elements specific to the page. Some pages have 200 elements specific to the page. I store the page specific elements for each page in an application in an individual repository file. This way when the User Interface or Document Object Model or HTML IDs of the page change, it would be easy to locate the particular page and update the values quickly.

### 5.1.2 02_repHomeElements

Create a repository file home to store all the home elements. I call the elements (links, logos, version, header, footer, menus etc.) that are common on all the pages of a web application as the Home elements. This practice eliminates the need to maintain the standard elements in all the individual page repository files and also eliminates the need to update numerous files.

### 5.1.3 02_repPages

Create a repository file Pages to store all the page titles. Most pages may not need individual repository files, but it would be necessary still test their existence. Though the elements of each page are

maintained in individual files, it is a good practice to store all the page titles in a singular file. It would be easier to update the Titles in one quick editing session.

#### 5.1.4 Create 02_repPages

Go to home pages for Gmail, Netflix login and Yahoo mail. Right click anywhere on the page. Select option "`assertTitle`". Add the variable name in the column Value as such - for Gmail add `pGmailLogin`, for Netflix login add `pNetflixLogin`, for Yahoo Mail add `pYahooLogin`.

**Note:** We have successfully created the pages repository file necessary for our test today. In real time, the pages of a particular application only are to be added to this file.

# 6. 03_DataSheets

Selenium IDE does not support data driven testing. However, we can create files specifically to store data. Several of these files can be created ahead of time and saved for iteration. This kind of data files may also be used for setting up data, to keep track of the data already used and so on. Instead of in an Excel sheet, the data is maintained in the Selenium test case itself.

Each data using test case does not need an individual data file. For example: the file used only once for Registration can also be used for several Logins and profile changes, etc. (For now, only data for Username and Password will be discussed.)

## 6.1 Data Sheets for Gmail

In Selenium IDE section Test Case, Right click and select "New Test Case".
Click in the Source area to highlight the first line. In the below fields, type in –

| **Command** | **Target** | **Value** |
|---|---|---|
| Store | GmailUsername | data_Username |
| Store | Abc123^^ | data_Password |

Save the test case as "03_data_Gmail_Register" at "C:\000_SeleniumIDE\03_DataSheets".

## 6.2 Data Sheets for Netflix

In Selenium IDE section Test Case, Right click and select "New Test Case".
Copy all the lines of code from file "01_data _Gmail_Register". Modify data as necessary.

| **Command** | **Target** | **Value** |
|---|---|---|
| Store | NetflixUsername | data_Username |
| Store | Abc123^^ | data_Password |

Save the test case as "03_data_Netflix_Register" at "C:\000_SeleniumIDE\03_DataSheets".

## 6.3 Data Sheets for Yahoo

In Selenium IDE section Test Case, Right click and select “New Test Case”.
Copy all the lines of code from file “01_data _Gmail_Register”. Modify data as necessary.

| **Command** | **Target** | **Value** |
|---|---|---|
| Store | YahooUsername | data_Username |
| Store | Abc123^^ | data_Password |

Save the test case as “03_data_Yahoo_Register” at “C:\000_SeleniumIDE\03_DataSheets”.

### 6.3.1 About data sheets

I like to name the data variables as “data_(name)”. This way the data variables won’t interfere with repository variables. I would also highly recommend maintaining a main data file for each set of functionalities. This main data sheet could be cloned and reused with a new name and updated data. Uniquely naming the data files will help identify the data we seek for a particular test results.

# 7. 04_GoToURL

Selenium typically records the default URL each test case is supposed to invoke before executing the steps in it. There are plenty of occasions where we would like to use the steps created on one environment in another (example: login script created in environment QA1 is to be used on the build installed in environment QA2). There are plenty of occasions where we would like to continue running various test case steps on a particular environment too. For this purpose, I would highly recommend maintaining a file specifically for invoking the URL alone.

## 7.1 04_GoToURL for Gmail

In Selenium IDE section Test Case, Right click and select “New Test Case”.
Start recording (click on the red button. The red button turns pink.)
In Mozilla Firefox, navigate to Gmail login page.
Stop recording.
Add (write) steps to verify title and to capture a screen shot of the page for reference.

| **Command** | **Target** | **Value** |
|---|---|---|
| *Open* | \ | |
| assertTitle | ${pGmailLogin} | |
| captureEntirePageScreenshot | ${FilePath}.png | |

Save the test case as “04_GoTo_Gmail” at “C:\000_SeleniumIDE\04_GoToURL”.

## 7.2 04_GoToURL for Netflix

In Selenium IDE section Test Case, Right click and select “New Test Case”.
Copy all the steps from test case “04_GoTo_Gmail” and paste them in the new Test Case.

Update "`${pGmailLogin}`" to "`${pNetflixLogin}`"
Save the test case as "04_GoTo_Netflix" at "C:\000_SeleniumIDE\04_GoToURL".
Via Windows Explorer navigate to the location "C:\000_SeleniumIDE\04_GoToURL".
Open the file "04_GoTo_Netflix"
Replace the URL Gmail login page with Netflix login page.
Save the file and close it.

### 7.3 04_GoToURL for Yahoo

In Selenium IDE section Test Case, Right click and select "New Test Case".
Copy all the steps from test case "04_GoTo_Gmail" and paste them in the new Test Case.
Update "`${pGmailLogin}`" to "`${pYahooLogin}`"
Save the test case as "04_GoTo_Yahoo" at "C:\000_SeleniumIDE\04_GoToURL".
Via Windows Explorer navigate to the location "C:\000_SeleniumIDE\04_GoToURL".
Open the file "04_GoTo_Yahoo"
Replace the URL Gmail login page with Yahoo login page.
Save the file and close it.

## 8. 07_Login scripts for Gmail, Netflix, Yahoo

We are now going to create selenium test cases for Login Steps, Login Pass, Login Fail and Log Out. These test cases will be hand written (keyboard typed), not recorded using the variables created in the previous files. These test cases can be used for all the three user interfaces (Gmail, Netflix, and Yahoo).

In Selenium IDE section Test Case, Right click and select "New Test Case".
Manually type the following in Command, Target, and Value.

### 8.1 Login Steps

| **Command** | **Target** | **Value** |
|---|---|---|
| Type | ${fUsername} | ${data_Username} |
| type | ${fPassword} | ${data_Password} |
| clickAndWait | ${bSignIn} | |

Save the test case as "07_LoginSteps" at "C:\000_SeleniumIDE\07_Login".

#### 8.1.1 Purpose of Login Steps

Type in field Username the data specified for the Username;
Type in field Password the data specified for the Password;
Click button Sign In and wait for the page to load.
**Note:** This test case only carries out the steps for a test. It does not verify if the test had passed or failed.

## 8.2 Login Pass

| Command | Target | Value |
|---|---|---|
| assertElementPresent | ${tUsername} | |
| captureEntirePageScreenshot | ${FilePath}${Username}_PASS.png | |

Save the test case as "07_LoginPASS" at "C:\000_SeleniumIDE\07_Login".

### 8.2.1 Purpose of Login Pass

Only if text Username is present execute the rest of the steps;
Capture and save a screen shot at the specified location as PASS.
**Note:** This test case executes and logs a picture as PASS, if the desired outcome (text Username) exists.

## 8.3 Login Fail

| Command | Target | Value |
|---|---|---|
| assertElementNotPresent | ${tUsername} | |
| assertElementPresent | ${fUsername} | |
| captureEntirePageScreenshot | ${FilePath}${Username}_FAIL.png | |

Save the test case as "07_LoginFAIL" at "C:\000_SeleniumIDE\07_Login".

### 8.3.1 Purpose of Login Fail

Only if text Username is NOT present execute the rest of the steps;
Only if field Username is present execute the rest of the steps;
Capture and save a screen shot at the specified location as FAIL.
**Note:** This test case executes and logs a picture as FAIL, if the desired outcomes (text Username does not and field Username) exist.

## 8.4 Log Out

| Command | Target | Value |
|---|---|---|
| assertElementPresent | ${tUsername} | |
| Click | ${tUsername} | |
| clickAndWait | ${bSignOut} | |
| captureEntirePageScreenshot | ${FilePath}${Username}_LogOut.png | |

Save the test case as "07_LogOut" at "C:\000_SeleniumIDE\07_Login".

### 8.4.1 Purpose of Log Out

Only if text Username is present execute the rest of the steps;
Click the text Username;
Click the button Sign Out and wait for the page to load.
Capture and save a screen shot at the specified location as LogOut.
**Note:** This test case executes and logs a picture as LogOut only if the desired outcome (text Username) exists.

# 9. Base Script

For effortless and quick test case creation, I would recommend creating and maintaining a base script. The base script will consist of all the elements needed for writing a test case for most actions, verifications and assertions.

| **Command** | **Target** | **Value** |
|---|---|---|
| assertTitle | ${p} | |
| assertElementPresent | ${} | |
| assertElementNOTPresent | ${} | |
| click | ${rb} | |
| click | ${cb} | |
| click | ${l} | |
| click | ${b} | |
| type | ${f} | ${data_} |
| select | ${ddl} | ${data_} |
| captureEntirePageScreenshot | ${FilePath}${}_.png | |

# 10. Test set arrangement and execution order

We have created test cases in modular fashion. It is time to put them together sensibly. As I had already mentioned,
01_FilePath (files – 01_FilePath_First, 01_FilePath_Second, 01_FilePath_Third),
02_Repository (files – 02_repPages, 02_repHomeElements files; If specific page or pages are involved in the test, 02_rep (page(s)) will also be necessary in the test suit),
03_DataSheet (optional, datasheets may be added as necessary or may be skipped for some test sets),
04_GoToURL (one file suffices)
The test cases should be added in ascending order till 04_GoToURL.

This is one way to create, store and guesstimate when a test case is ready and if it is set up in order. Now, the actual test cases may be added in the order they would execute manually.

# 11. Test Execution and Results

Typically, all test cases are supposed to turn Green in order to assume that the test set passed. Here, that is not the case. Certain test cases are supposed to pass and certain test cases are supposed to fail. Each test case will only execute if the required conditions are met.

Provide valid credentials to pass a test. Passed tests would have three picture logs, one for navigating to the website, one for successful login, one for log out.

Provide invalid credentials to fail a test. Failed tests would have two picture logs, one for navigating to the website, one for failed login.

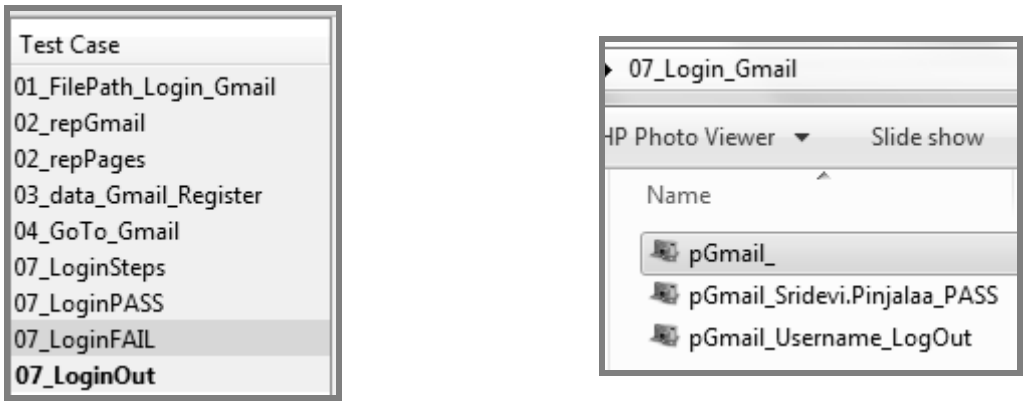



*Figure 8 – Gmail Login – PASS*


*Figure 9 – Gmail Login – FAIL*

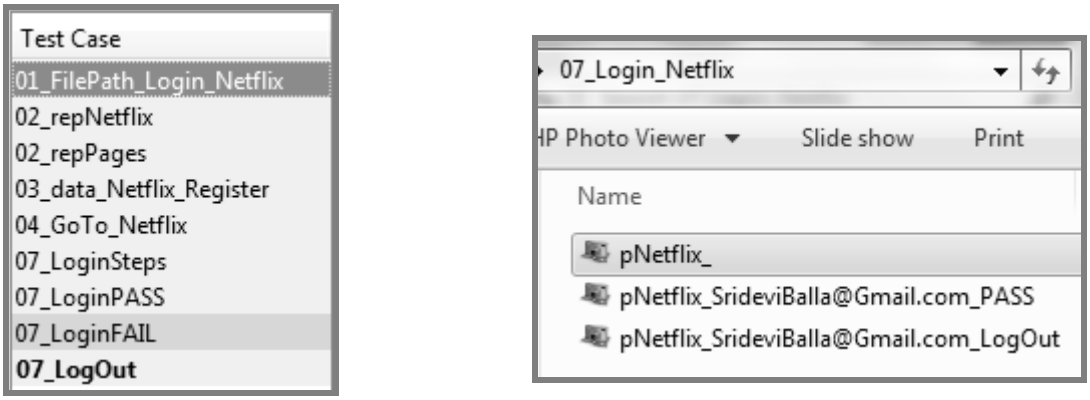



*Figure 10 – Netflix Login – PASS*

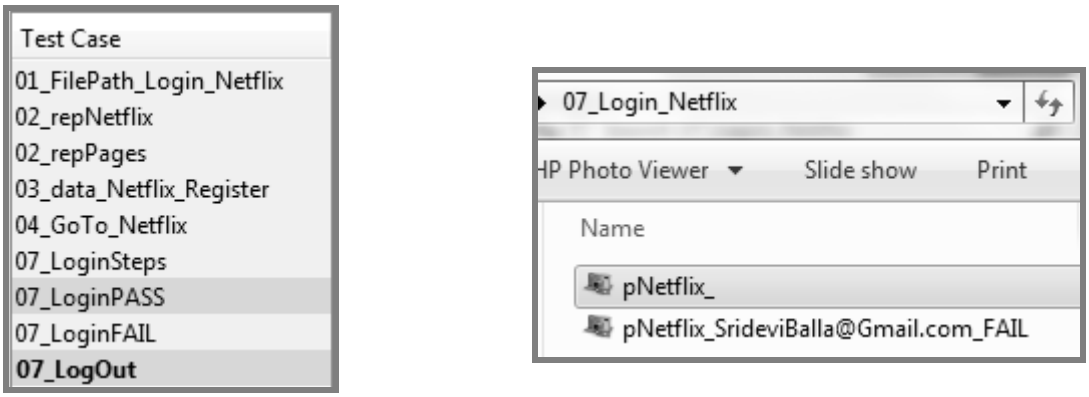



*Figure 11 – Netflix Login – FAIL*

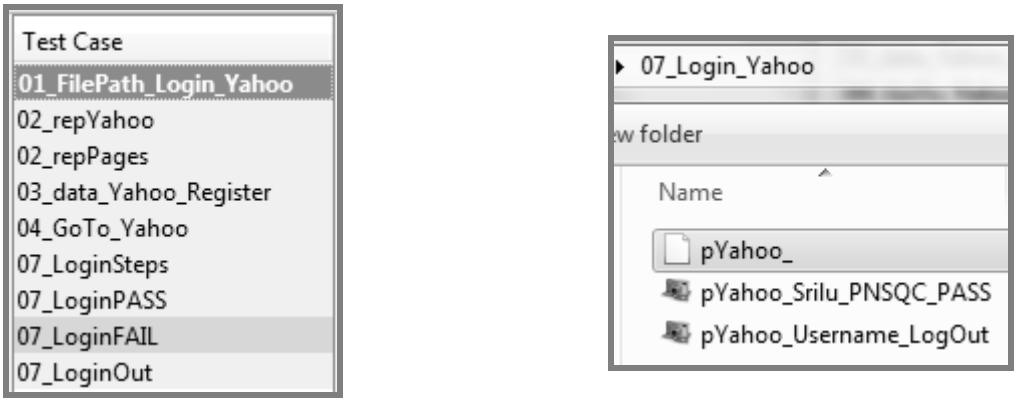



*Figure 12 – Yahoo Login – PASS*


*Figure 13 – Yahoo Login – FAIL*

# 12. SAREE

The most suitable mnemonic to summon up the steps for Green Lantern Automation concept would be "Select, Assert, wRite, Execute, Exact" (SAREE). The scripts, just like a Saree can be wrapped to any User Interface testing as long as their functionalities are similar.

**Select** elements for testing.

Select the elements that would be used for testing. Listed below are a few common elements.

- Page
- Field (text field)
- Button
- Check box
- Radio button
- Link
- Menu
- Sub Menu
- Drop Down List
- Caption
- Image
- Logo
- Text
- Section
- Dialog box

**Assert** appropriate custom name to each element in the repositories

Rename a few of the above elements to describe them fittingly. Listed below are a few elements named to fit Google page.

- Page → `pGoogle`
- Field (text field) → `pGoogle_fUsername`
- Button → `pGoogle_bSearch`
- Check box → `pGoogle_cbChose`
- Radio button → `pGoogle_rbTrue`
- Link → `pGoogle_lHome`
- Menus → `pGoogle_mMore`
- Sub Menu → `pGoogle_smTranslate`
- Drop Down List → `pGoogle_ddlSettings`
- Caption → `pGoogle_capGoogle`
- Image → `pGoogle_imgGmail`
- Logo → `pGoogle_logoGoogle`
- Text → `pGoogle_tVersion`
- Section → `pGoogle_secSearch`
- Dialog box → `pGoogle_dSuccessful`

Based on the naming convention one can (with a little bit of training) can identify that the elements belong to the page Google

**wRite** the test cases with custom names

Selenium IDE typical code would look as such **–**

| **Command** | **Target** | **Value** |
|---|---|---|
| `type` | `${pGmail_fUsername}` | `${data_Username}` |
| `click` | `${pGoogle_bSearch}` | |
| `select` | `${pGmail_ddlEnvironment}` | `${data_Environment}` |
| `assertTitle` | `${pGoogle}` | |

**Execute** the test cases in test sets.

**Exact** the element property in the repositories from time to time

# 13. Conclusion

Most of the time automated scripts are created by record and play method. This method only imitates the user's actions. This method won't log warning, error messages or perform comparisons that are particular to a function or application. Automated scripts and suits created for a particular application cannot be reused with other similar applications, especially if the UI changes are drastic. The effort and time that goes into the development of automated scripts will go waste as soon as changes are made to the application.

With the Green Lantern Framework, the scripts are safe. Major or minor UI changes to an application will not cause script death. The test cases created for one application can also be used for applications with similar functionality. Since the test cases are reusable (green), the engineers will have a lot of time to focus on creating new test scenarios versus constantly updating the same test cases.

Since the UI elements are stored in a 'repository' file, the framework enables engineers to create test cases even before an application build is released for testing. When the build is ready, the tester can populate the repository file(s).

Green Lantern Framework - select elements for the test, assert custom names in English as they appear on the interface, wRite the test cases, execute the test cases in test sets and exact the element ID or Name in the repository file.

# Works Cited

*Gmail*. (n.d.). Retrieved from https://accounts.google.com

*Green Lantern*. (n.d.). From http://en.wikipedia.org/wiki/Green_Lantern

*Netflix*. (n.d.). Retrieved from https://signup.netflix.com/Login

*SAREE*. (n.d.). From http://en.wikipedia.org/wiki/Sari

*Yahoo*. (n.d.). Retrieved from https://login.yahoo.com