Green Lantern Automation Framework

Sridevi Pinjala

IBM

Twitter: @SriluBalla

Abstract

Once upon a time, the average life for software was 7 years. Some software lasted more than 7 years. Some software lasted less than 7 years. But no matter how long the software lasted, its code was updated, improvised, changed and tweaked frequently. To handle the tweaks and to make sure the other code was not broken, regression testing came into being. Automated tools were invented to handle regression. But most automated tests also needed to be tweaked when the application code was tweaked. When the application interface changed completely, the automated scripts died.

Some companies had several interfaces for one back-end system. They had to pay the price to have all these interfaces tested with the same tests and maintained separately. There was no way out. If one particular interface has hundreds of automated scripts written already, they could not re-use the scripts on other interfaces.

Some companies had frequent changes to the User Interface of the application. Since the automated test case scripts needed to be changed every time the User Interface changed, automation did not add any value nor did it decrease the need for manpower. So, the companies had to rely on manual testing alone.

Many test automation tools came into the market to help out the needy companies. Some tools relied on the objects used for testing, some relied on the User Interface, and some relied on the Document Object Model. Almost all of these tools had a feature - to change the logical name on the elements to give them a unique identification to avoid conflict.

Biography

Sridevi Pinjala is an automation expert at IBM and is now working on IBM test automation tools. She came up with Green Lantern framework when working with Everest Consultants. She is a Certified Scrum Product Owner and currently pursuing MS in Information Management at Aspen University. She considers Job Bach her mentor without whose encouragement, she wouldn't be here to present her ideas. She identifies herself as the American in the Saree. (http://sriluballa.wordpress.com)

1. Introduction

Many companies have invested in test automation tools in hopes that the automation tool will free up manpower by taking over regression testing. The automation tools look at the skins and DOMs of the applications. They are identified by the properties assigned to the elements. (A cell could be identified by the “Row and Column”, a table can be identified by “number”, a link can be identified with ‘inner HTML or text”, and so on.)

When the skin and the Document Object Model of an application changes a lot or little, it becomes unavoidable to update and change the automation scripts a lot or a little too. And if the application User Interface was completely changed (for any reason) the automated scripts will need to be recreated from scratch.

In one of my roles as an automation engineer, I was in charge for creating test scripts for three different interfaces that used the same back end. Their functionality was same, but their user interface was completely different. I started off as everybody - creating scripts for each User Interface. But it felt like too much hard work but like “too little smart work”. So, I needed a framework where I could avoid script death by eighty percent and increase re-usability by eighty percent. I needed a disruptive framework. That is when I came up with “Green Lantern Framework”

Using the renaming feature of the automation tool that relies on the Document Object Model, the test scripts could be created once and used no matter how much the User Interface changed or the even the Document Object Model changed. For example automated test scripts created for Gmail Log-in page can be used for Yahoo Log-in page, Hotmail Log-in page or any log-in page with a user name and password.

I will demonstrate creating automated scripts using Green Lantern framework with the tool Test Complete and scripting language VBScript.

2. Green Lantern Steps

- Choose elements on the page that are targeted for automation.
- Capture the elements in a repository
- Rename them in English as they would appear on the page
- Categorize them per their type (Example – Text Field, Check box, Radio button, etc.)
- Write scripts using the pseudonym names
  - Write a routine for each element
  - Routines for text fields should have data coming from external Data sheets
  - Checkboxes are checked ON or OFF via Data Sheets
  - Log messages, Warning, Errors should be written in the script routine

2.1. Concept –

Main elements in a software application are

- Text fields
- Check boxes
- Radio buttons
- Links
- Menus
- Captions
- Images
- Logos

Every test uses some or all of the elements in a page. These elements are captured and categorized and named as they display. Every test uses some or all of the elements in a page. When these elements are captured and categorized and named the way they would display, it is easy to identify every element, easy to write a script, easy to detect element recognition failures, etc.

2.2. Choose elements for automating login function

Elements essential for Logging into a web page are the URL, the fields - User Name and Password, and button – Sign In.

Figure 1– Choose elements necessary for the test from Gmail Login page

2.3. Capture the elements to the repository

Capture the elements the fields - User Name and Password, and button – Sign In from the Gmail Login page. (Example – Figure 2)

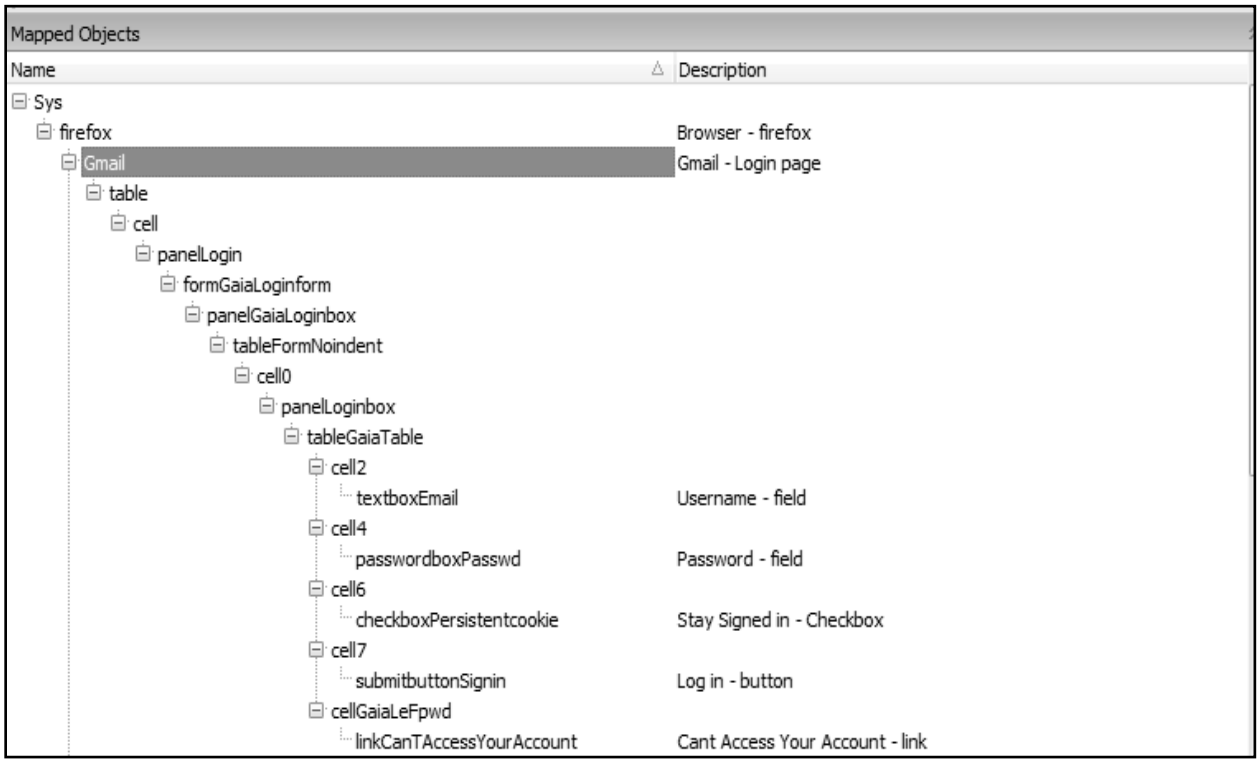

Figure 2 – Gmail Login elements Original

2.4. Rename the logical names of the elements

Firefox = MF

GmailLogin = Gmail_Login

Username (in Gmail) = UserID

Password = Password

Sign In = SignIn

Stay signed in, = StaySignedIn

Can't access your account? = ForgotPasword

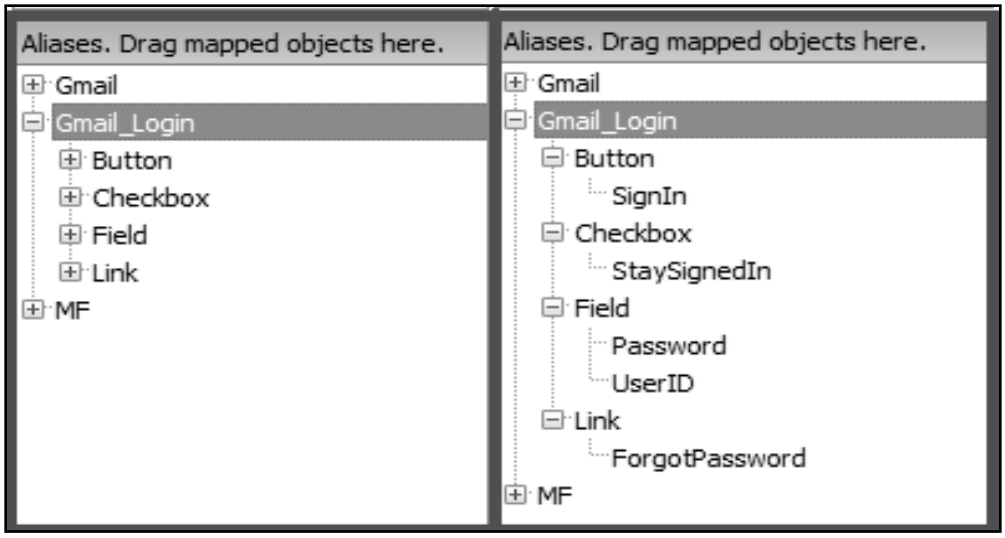

Figure 3 – Gmail Login elements renamed and categorized

2.5. Categorize the elements per their type

To categorize, choose element "panelLoginbox" (which is the parent element for all the field, checkbox, button, link elements). Drag this element and add it under Gmail and rename it as "Field". Add elements UserID and Password under the Field. Drag "panelLoginbox" again to Gmail and rename it as "Checkbox". Add element "StaySignedIn" under the Checkbox. Drag "panelLoginbox" again to Gmail and rename it as "Link". Add element "ForgotPassword" under the Link. Drag "panelLoginbox" again to Gmail and rename it as "Button". Add element "SignIn" under the Button. The end result with look like Figure 3 (Collapsed on the right and expanded on the left).

2.6. Test Scripts

Using the renamed Aliases elements, it will be effortless and uncomplicated for a new-comer or an experienced programmer to write scripts. (Note - Naming the elements and categorizing them can be decided upon studying the Document Object Model, properties of elements and such.)

2.6.1. Routine for the field User ID

Step1. Start with "`Aliases.`"

Step2. Call the Page or Website you wish to use for the test. In this instance it is Gmail_Login element.

So add – "`Aliases.Gmail.`"

Step3. Call the type of element you want to use for the test. In this instance it is the Field element. So add – "`Aliases.Gmail.Field.`"

Step4. Call the element you want to use for the test. In this instance it is the UserID element. So add – "`Aliases.Gmail.Field.UserID`"

Step5. Call the function the step is supposed to perform with the element. A textbox field can either enter "text" or type in "Keys". So add = "`Aliases.Gmail.Field.UserID.Keys.`"

Step6. Text or Values to be used could be static or dynamic. I like dynamic. I like to drive my data from the external datasheet. This way any member of the team can come up with the data they would like to use, without ever touching the test scripts. So add = "`Aliases.Gmail.Field.UserID.Keys(D`**`DT`**`.CurrentDriver.Value("UserID"))`"

Step7. I like to perform a few more steps before I input the indented UserID. I like to remove any pre-existing text that might be in the field. For this purpose I copy paste the above code twice and replace the `(D`**`DT`**`.CurrentDriver.Value("UserID"))` with `("^a")` from the first line to select all the existing text. Replace the `(D`**`DT`**`.CurrentDriver.Value("UserID"))` with `("[Del]")` from the second line to delete all the existing text. So copy and paste and modify the lines to -
`Aliases.Gmail.Field.UserID.Keys("^a"); Aliases.Gmail.`
`Field.UserID.Keys("[Del]")`

Step 8. Insert a message in the log to know which UserID was used. So add – `Log.Message("User ID = " + Aliases.Gmail_Login.Field.UserId. Value)`

Step 9. Name the routine to match the element. The end result would look like Figure 4.

```
57  Sub fUserID()
58    Aliases.MF.Gmail_Login.Field.UserID.Keys("^a")
59    Aliases.MF.Gmail_Login.Field.UserID.Keys("[Del]")
60    Aliases.MF.Gmail_Login.Field.UserID.Keys(DDT.CurrentDriver.Value("UserID"))
61    Log.Message("User ID =  " + Aliases.Gmail_Login.Field.UserId.Value)
62  End Sub
```

Figure 4 – Routine for field User ID

2.6.2. Routine for the field Password

Step1. Copy paste the UserID routine and replace the value “UserID” with “Password”. The end result would look like Figure 5.

```
65  Sub fPassword()
66    Aliases.MF.Gmail_Login.Field.Password.Keys("^a")
67    Aliases.MF.Gmail_Login.Field.Password.Keys("[Del]")
68    Aliases.MF.Gmail_Login.Field.Password.Keys(DDT.CurrentDriver.Value("Password"))
69    Log.Message("Password =  " + Aliases.Gmail_Login.Field.Password.Value)
70  End Sub
```

Figure 5 – Routine for field Password

2.6.3. Routine for the button Sign In

Step1. Copy and paste the UserID routine. Remove the 2nd and 3rd and 4th lines.

Step2. Replace the value “Field” with “Button”.

Step3. Replace the value “UserID” with “SignIn”.

Step4. Replace the value “Keys (“^a”) with “Click ()”.

Step5. Insert a message to the log to suggest button Sign In was clicked on.. So add –

```
Log.Message("Clicked button Sign In")
```

Step6. Rename the routine to match the element name and type. The end result would look like Figure 6.

```
76
77  Sub bSignIn()
78    Aliases.MF.Gmail_Login.Button.SignIn.Click()
79    Log.Message("Clicked button Sign In")
80  End Sub
```

Figure 6 – Routine for button Sign In

2.6.4. Routine for the checkbox Stay signed In

Step1. Copy and paste the Sign In routine. Remove the 2nd line.

Step2. Replace the value “Button” with “Checkbox”.

Step3. Replace the value “SignIn” with “StaySignedIn”.

Step4. Replace the value “Click() with “Click(false)”.

Step5. Insert a message to the log to suggest the check box Stay Signed In was checked ON. So add – `Log.Message("Uncheck check box Stay Signed In")`

Step6. Rename the routine to match the element name and type. The end result would look like Figure 7.

```
72  Sub cSignedInUncheck()
73    Aliases.MF.Gmail_Login.Checkbox.StaySignedIn.ClickChecked(False)
74    Log.Message("Unchecked check box - Stay Signed In")
75  End Sub
76
```

Figure 7 – Routine for checkbox Stay Signed In

2.6.5. Routine verifying the login

The end result would more or less look like Figure 8.

```
45
46  Sub VerifyLogin()
47      If Aliases.MF.Gmail.Link.Exists Then
48          Call Log.Picture(Aliases.MF.Gmail, "Gmail Invoked", "Gmail Invoked", 300,
     0, -1)
49          Log.Message("Gmail invoked via Mozilla Firefox")
50      Else
51          Call Log.Picture(Aliases.MF.Mozilla_Firefox, "Gmail Not Invoked or
     Recognized", "Gmail Not Invoked or Recognized", 300, 0, -1)
52          Log.Message("Gmail Not Invoked or Recognized")
53      End If
54  End Sub
55
```

Figure 8 – Routine to Verify Login

2.6.6. Routine calling all the login steps

The end result would more or less look like Figure 9.

```
65  Sub LoginSteps()
66     Call fUserID
67     Call fPassword
68     Call cSignedInUncheck
69     Call bLogIn
70     Call VerifyLogin
71  End Sub
```

Figure 9 – Routine to combine all elements for login Function

2.6.7. Main routine for validating the function Login

Many steps, ideas, routines, conditions, go into it. The end result would more or less look like Figure 10 and Figure 11.

```
''' ################## LOGIN SCRIPT FOR Gmail VIA MOZILLA FIREFOX ################
Sub Login()

    Set Driver = DDT.ExcelDriver("C:\SriluBalla\DataSheets\Login.xlsx", "Gmail",
 True)
    Driver.Name = "Login"
    Log.AppendFolder("Gmail LogIn with Browser Mozilla Firefox")

  If Not Aliases.MF.Gmail_Login.Exists Then

       TestedApps.TerminateAll()
       TestedApps.firefox.Run()
```

Figure 10 – Script for Login Routine (continued…)

```
12
13      Aliases.MF.ToURL(DDT.CurrentDriver.Value("URL"))
14      Log.Message("URL =  " + DDT.CurrentDriver.Value("URL"))
15
        If Aliases.MF.Gmail_Login.Exists Then
17         Call LoginSteps
18      Else
19         Call Log.Picture(Aliases.MF.Mozilla_Firefox, "Unknown Error", "Unknown
    Error", 300, 0, -1)
           Call Log.Error("Gmail Log In Not available")
21         Call Runner.JustStop(False)
22      End If
23
         If Not Aliases.MF.Gmail.Exists Then
25          Call Log.Picture(Aliases.MF.Mozilla_Firefox, "Unknown Error", "Unknown
    Error", 300, 0, -1)
            Call Log.Error("Gmail page In Not Available or Log in Failed")
27       End If
28
29     End If
30     Log.PopLogFolder()
31
32     DDT.CloseDriver(Driver.Name)
33
34  End Sub
```

Figure 11 – Script for Login Routine

2.7. Test run results (Log files)

The end result of a test run log would more or less look like Figure 12 and read –

Gmail LogIn with Browser Mozilla Firefox

URL = www.gmail.com (URL text comes from data sheet)

UserID = Srilu.Balla (URL text comes from data sheet)

Password = (URL text comes from data sheet)

Gmail Invoked via Mozilla Firefox

Type	Message	Time	Priority	Has Pict...
	Gmail LogIn with Browser Mozilla Firefox	11:1...	Normal	
	The application "C:\Program Files (x86)\Mozilla Firefox\firefox.exe" started.	11:1...	Normal	
	URL = www.gmail.com	11:1...	Normal	
	UserID = Srilu.Balla	11:1...	Normal	
	Password =	11:1...	Normal	
	The check box is already unchecked.	11:1...	Normal	
	Checkbox Stay Signed In Checked OFF	11:1...	Normal	
	Button Sign In Clicked	11:1...	Normal	
	Gmail Invoked	11:1...	Normal	
	Gmail invoked via Mozilla Firefox	11:1...	Normal	

Figure 12 – Gmail login Test Result

A lot of work went into the Login script. (Phew)

3. User Interface Change

Now imagine the Gmail login User Interface changes from Gmail to Hotmail or yahoo. All the time, effort and ideas put into creating the original script have to be redone in most conventional conditions. But with Green Lantern framework the script is safe as long as there is a repository that matches it.

If the aliases DOM's for Hotmail and Yahoo are prepared similar to the aliases DOM for Gmail, the Scripts will work with little or no modification and log similar messages and check for similar functionality.

3.1. Green lantern Steps for Yahoo Login

Step1. Create Repository for Hotmail Login Page

Step2. Rename the elements and categorize tem by type and rename the logical names to match the aliases names for the Gmail login page. Result would look similar to Figure 13.

Step3. Copy paste the Login Script for Gmail - Open a blank script and name it Yahoo_Login and Copy paste the Gmail Login Script

Step4. Replace "Gmail" with "Yahoo" using 'find and replace' (Ctrl + H)

Step5. Verify the replaced text and run the script. Result would look similar to Figure 14.

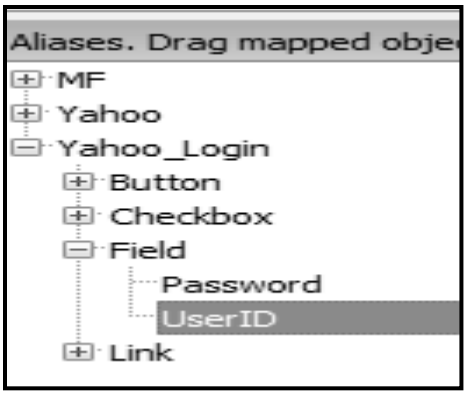

Figure 13 – Yahoo login screen repository

3.2. Test run results (Log files)

The end result of a test run log would more or less look like Figure 14 and read –

Gmail LogIn with Browser Mozilla Firefox

URL = www.mail.yahoo.com (URL text comes from data sheet)

UserID = srilu_kiku (URL text comes from data sheet)

Password = (URL text comes from data sheet)

Yahoo Login successful or Yahoo Login failed.

Type	Message	Time	Priority	Has Pict...
⊟	Yahoo LogIn with Browser Mozilla Firefox	14:4...	Normal	
	The application "C:\Program Files (x86)\Mozilla Firefox\firefox.exe" started.	14:4...	Normal	
	URL = www.Mail.Yahoo.com	14:4...	Normal	
	UserID = srilu_kiku	14:4...	Normal	
	Password =	14:4...	Normal	
	The check box is already unchecked.	14:4...	Normal	
	Checkbox Stay Signed In Checked OFF	14:4...	Normal	
	Button Sign In Clicked	14:4...	Normal	
	Yahoo Login Failed	14:4...	Normal	
	Yahoo Login Failed	14:4...	Normal	
	Unknown Error	14:4...	Normal	
	Yahoo page In Not Available or Log in Failed	14:4...	Normal	

Figure 14 – Yahoo login Test Result

4. A few issues with conventional automation

- Most automation is record and play. This technique will execute a few steps on an application, but wont log full length messages, warnings and won't do comparisons.

- A lot of time goes into automating regression suits. A lot of effort goes into automation. A lot of ideas go into automation. All of this effort goes waste with a few tweaks to the User Interface.

- Several developers (from all over the world) are developing scripts for automation. Unless there was a coding standard defined, each developer would pursue their own style of repository naming and script writing. The automation suits will not be consistent.

- Automated suits with various coding standards are hard to merge and maintain.

- Automated scripts created for one application cannot be used if changes are made to the application. They will need to be updated as well.

- A test had failed due to recognition issues, by looking at the logical names in the repository or the script alone it is not enough to identify which element needs updating. Because sometimes several elements could be named same and several times the names given on the page are not used for the logical name.

5. Conclusion

If this framework is understood correctly, engineers using it can create automation scripts way before the application is developed. Most routines can be utilized over and over after changin a few names in the routines which can be accomplished via find and replace (Ctrl+H).

Several applications with similar functionality could use the same routines and scripts that were already created. Now, the engineer and the testers can come up with more scenarios.

Green Lantern Framework is compatible with the tool TestComplete. Automation tools – Quick Test Professional, Rational Functional Tester, Selenium etc also can have reusable scripts. The approach to these tools varies from the current approach but the concept is similar.

Changes in the DOM and the User Interface of any application is inevitable. During such situations automation script death is inevitable. Many automation tools have a feature to rename the elements being used for tests. When these elements are cleaverly and clearly renamed to suit the elements (also making it easy to remember during script creation), script death can be minimized or altogether avoided and simultaneously the reusability of the script can be increased.

I would like my learned and knowledgeable colleagues to test and question these ideas. I need their invaluable advice, guidance and participation to enhance this framework and to implement on other automation tools.