2. Selenese

Selenium Commands - Selenese

Selenese consist of a set of Selenium commands that are used to run the selenium tests. In selenese, you can test the existence of UI elements based on their HTML tags, test for specific content, input fields, selection list option, submitting forms and table data.
There are three types of Selenium commands:

- Actions – commands that generally manipulates the state of the application (click this link, select that option).Many actions are called with “clickAndWait”. This tells Selenium that it needs to wait for the page to load.
- Accessors – examine the state of the application and store the results in a variable (storeTitle)
- Assertions – they verify that the state of the application conforms to what is expected.(make sure that the page has title x). There are used in 3 modes: “assert”, “verify”, “waitFor”

Most common Selenium commands

To conclude our introduction of Selenium, we’ll show you a few typical Selenium commands. These are probably the most commonly used commands for building tests.

- open - opens a page using a URL.
- click/clickAndWait - performs a click operation, and optionally waits for a new page to load.
- verifyTitle/assertTitle - verifies an expected page title.
- verifyTextPresent - verifies expected text is somewhere on the page.
- verifyElementPresent - verifies an expected UI element, as defined by its HTML tag, is present on the page.
- verifyText - verifies expected text and its corresponding HTML tag are present on the page.
- verifyTable - verifies a table’s expected contents.
- waitForPageToLoad - pauses execution until an expected new page loads. Called automatically when clickAndWait is used.
- waitForElementPresent - pauses execution until an expected UI element, as defined by its HTML tag, is present on the page.
- storeElementPresent - This corresponds to verifyElementPresent. It simply stores a boolean value–”true” or “false”–depending on whether the UI element is found.
- storeText - StoreText corresponds to verifyText. It uses a locater to identify specific page text. The text, if found, is stored in the variable. StoreText can be used to extract text from the page being tested.

Locating Elements

When locating an element, a target is requires. This target identifies an element in the content of the web application, and consists of the location strategy followed by the location in the format locatorType=location.
We will take into consideration the following code block for the next examples:

```
<html>
<body>
 <form id="loginForm">
  <input name="username" type="text" />
  <input name="password" type="password" />
  <input name="continue" type="submit" value="Login" />
  <p>You don’t have an account yet?</p>
  <a href="createuser.html">SignUp</a>
 </form>
</body>
<html>
```

Locating by Identifier

With this strategy, the first element with the id attribute value matching the location will be used. If no element has a matching id attribute, then the first element with a name attribute matching the location will be used.
e.g. :
identifier=loginForm
identifier=password
identifier=continue

Locating by Id

Use this when you know an element’s id attribute.
E.g.
id=loginForm

Locating by Name

The name locator type will locate the first element with a matching name attribute. If multiple elements have the same value for a name attribute, then you can use filters to further refine your location strategy.
E.g.
name=username
name=continue type=button

Locating by XPath

XPath is the language used for locating nodes in an XML document. XPath extends beyond (as well as supporting) the simple methods of locating by id or name attributes, and opens up all sorts of new possibilities such as locating the third checkbox on the page.

xpath=/html/body/form[1] (3)- Absolute path (would break if the HTML was changed only slightly)
//form[1] (3) - First form element in the HTML
xpath=//form[@id='loginForm'] (3) - The form element with attribute named ‘id’ and the value ‘loginForm’
xpath=//form[input/@name='username'] (3) - First form element with an input child element with attribute named ‘name’ and the value ‘username’
//input[@name='username'] (4) - First input element with attribute named ‘name’ and the value ‘username’
//form[@id='loginForm']/input[1] (4) - First input child element of the form element with attribute named ‘id’ and the value ‘loginForm’
//input[@name='continue'][@type='button'] (7) - Input with attribute named ‘name’ and the value ‘continue’ and attribute named ‘type’ and the value ‘button’
//form[@id='loginForm']/input[4] (7) - Fourth input child element of the form element with attribute named ‘id’ and value ‘loginForm’

Locating Hyperlinks by Link Text

This is a simple method of locating a hyperlink in your web page by using the text of the link. If two links with the same text are present, then the first match will be used.

link=SignUp

Locating by DOM

The Document Object Model represents an HTML document and can be accessed using JavaScript. This location strategy takes JavaScript that evaluates to an element on the page, which can be simply the element's location using the hierarchical dotted notation.
E.g.
dom=document.getElementById('loginForm') (3)
dom=document.forms['loginForm'] (3)
document.forms[0].elements['username'] (4)
document.forms[0].elements[3] (7)

Locating by CSS

CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents. CSS uses Selectors for binding style properties to elements in the document. These Selectors can be used by Selenium as another locating strategy.
E.g.
css=input[name="username"] (4)
css=input.required[type="text"] (4)
css=#loginForm input[type="button"] (4)