Selenium Essentials

Get to grips with automated web testing with the amazing power of Selenium WebDriver

Prashanth Sams
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'The Selenium IDE'
- A synopsis of the book’s content
- More information on Selenium Essentials

About the Author

Prashanth Sams is a test automation engineer contributing to the IT industry since 2011. He graduated with a bachelor's degree in information technology from Anna University and lives in Chennai, India, with his family. He started his career as a human resource executive. Later, he worked at an HR outsourcing (US recruiting) company that operates in Chennai.

He is very passionate about test automation and has chosen to be a professional software engineer. He is an active blogger and a moderator for http://seleniumworks.blogspot.in/, a blog about Selenium, and is a great supporter of the Selenium Community, responding diligently to questions and answers over professional networks. He loves emerging technologies with soft skills development and spends 14 to 16 hours a day on them. In a short span of time, he has gained rich experience in various projects, handling different automation tools. Prashanth's Twitter handle is @prashanthsams.
Selenium Essentials

Selenium WebDriver is an open source software-testing tool used to automate web-based applications that is platform independent and that can be accessed by any popular programming languages. It's been about a decade since Jason Huggins started the Selenium project in 2004 at Thoughtworks. Later, in 2008, Simon Stewart combined his work on WebDriver with Selenium to give a new birth to Selenium WebDriver. Today, Selenium WebDriver is the most widely used web-automation tool around the world.

This book provides guidance that will help readers grasp Selenium WebDriver concepts fast. You will learn about the advanced features of the Selenium IDE and Selenium Builder, followed by cross-browser tests, methods of Selenium WebDriver, best practices involved, and extensive ideas to create a Selenium framework.

What This Book Covers

Chapter 1, The Selenium IDE, provides intense ideas to practice record-and-playback IDEs such as the Selenium IDE and Selenium Builder.

Chapter 2, Selenium WebDriver Cross-browser Tests, helps you to do efficient compatibility tests. Here, we will also learn about how to run tests in the cloud.

Chapter 3, Selenium WebDriver Functions, delivers all the functions of Selenium WebDriver in detail with examples on each.

Chapter 4, Selenium WebDriver Best Practices, explains how to manage Selenium automation tasks with dissimilar techniques.

Chapter 5, Selenium WebDriver Frameworks, guides you to customize and build any kind of automation framework using Selenium WebDriver.
The Selenium IDE (Integrated Development Environment) is an open source record-and-playback tool for generating Selenium scripts, which is integrated with the Firefox web browser as an extension. It is a renowned web-based UI test automation tool that extracts any kind of locator from the web page. The locators can be either attribute-based or structure-based, and include ID, name, link, XPath, CSS, and DOM. The IDE has the entire Selenium Core, which allows the users to record, playback, edit, and debug tests manually in a browser. The user actions in the web page can be recorded and exported in any of the most popular languages, such as Java, C#, Ruby, and Python.

Selenium Builder is an alternative open source tool for the Selenium IDE to record and playback web applications. It is an extension of the Firefox web browser, which is similar to the Selenium IDE, but, it has some unique features that the Selenium IDE doesn't support. Selenium Builder is a standard tool from Sauce Labs that runs tests on Sauce Cloud from the Selenium Builder interface itself.

In this chapter, we will learn about:

- Selenium IDE's record and playback abilities
- Selenium IDE functions
- Selenium IDE Data Driven tests
- Selenium IDE JavaScript functions
- Selenium Builder record and playback
- Selenium Builder Data Driven tests
- Selenium Builder on cloud
The Selenium IDE

The Selenium IDE is a Firefox extension to record and playback web-based applications. However, it does more than what a record-and-playback tool would do. Breakpoints allow the users to debug IDE commands step by step on runtime. The IDE has three different types of panes, namely the left pane, test case pane, and log / reference / UI-element / rollup pane.

Launch the Selenium IDE from the Firefox Tools menu, Tools | Selenium IDE. The IDE can also be opened using the Ctrl + Shift + S shortcut or by clicking on the Selenium icon in the top-right corner of the Firefox web browser. The Selenium icon is shown in the following screenshot:

A new, untitled test case will be created in Left Pane after launching the Selenium IDE. To start with a new test case, choose New Test Case from the File menu, that is, File | New Test Case, or make use of Ctrl + N, the Windows shortcut.

To start recording test scripts, click on the round, red icon from the playback control toolbar. By default, the record button will be active and the test scripts are recorded in Selenese, a domain-specific language that is similar to the HTML format. The playback control toolbar is shown in the following screenshot:

The Fast-Slow slider adjusts the test speed execution; the Play All button lets you run entire test cases as a test suite, where a test suite is a collection of test cases; and the Play button helps you to run the current test case. The Pause/Resume button pauses test execution for a while and allows the user to resume tests at their convenience.

The Test Case pane displays all the recorded steps with Command, Target, and Value. The Command column instructs the IDE about what to do. It comes with three different aspects, which are:

- Actions
- Accessors
- Assertions
The Selenium IDE has a list of built-in commands that let you drive tests as expected. Adding user-defined commands to the Selenium IDE is quite feasible by extending the external JavaScript methods. A command can be any one of the preceding three types. In the IDE, these commands are easily editable and replaceable with alternative commands while generating scripts.

Action commands manipulate the application state through some kind of actions, can be either action or actionAndWait. Action commands that end with the suffix AndWait allow the page to load fully before starting to execute the next command.

A few of the Action command examples are open, type, typeAndWait, select, selectAndWait, check, checkAndWait, click, and clickAndWait.

Accessors detect the application state and store results in a variable; store, storeText, and storeValue are the commands that are used to store values. For example, in the following screenshot, search is a variable and prashanth sams is the search keyword. Later on, the stored value is retrieved and is used as a parameter for an action, ${search}. The discussion in this paragraph is encapsulated in this screenshot:

<table>
<thead>
<tr>
<th>Command</th>
<th>Target</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>/?gfe_rd=cr&amp;ci=..</td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>prashanth sams</td>
<td>search</td>
</tr>
<tr>
<td>type</td>
<td>css=#gbqfq</td>
<td>${search}</td>
</tr>
</tbody>
</table>

Assertions verify the application state by validating the expected result. It is available in three different modes, namely, assert, verify, and waitFor. Assert fails and aborts the test execution upon failure, verify fails and continues the test execution upon failure, and waitFor waits for a specific condition to occur and fails upon timeout. By default, the timeout is set to 30,000 milliseconds, 30 seconds. In the Selenium IDE, the timeout can be manually configured using the Options menu.

The Target field directs the IDE to locate elements, and the general syntax for Target is as follows:

    locatorType = argument

An example for Target is as follows:

    css=#gbqfq
The Log / Reference / UI-Element / Rollup pane is displayed at the bottom of the IDE. This is shown in the following screenshot:

This pane allows the user to see the log information, command reference, UI-Element, and Rollup, among others. On installing the Neustar plugin to the Selenium IDE, a separate tab called Neustar Script Uploader will be shown along with the other tabs. Neustar WPM (formerly called Browsermob) is a web performance management tool for web page monitoring and load testing.

The log captures all the IDE test execution steps one by one and is mainly used for debugging purposes. The Debug menu in the bottom pane contains a list of options, namely Info, Debug, Warn, and Error. It lets you filter the explicit status, warning, and error messages, and certainly reduces the verbosity level.
The Reference tab gives a detailed explanation of the IDE commands upon clicking on each row from the Test Case pane. In the case of user-defined commands, the Reference tab will not include any information. Rollup executes a group of commands in one step; it is reusable and can be used any number of times within the test case. Refer to Help | UI-Element Documentation for more details about UI-Element and Rollup.

While recording test scripts, the Selenium IDE provides UI-based options for every mouse right-click on elements on a browser web page. To achieve this, right-click on the web page and hover the mouse over Show All Available Commands. The following screenshot is the result of this action:

```
open ?gfe_rh=c&et=wlZ3O7pXKcbN89geQn4CoCg8gwrc_rh=ssl
assertTitle Google
assertValue
assertText ///[id='gb']//div/div +YouGmailImages +YouSearchYouTubeMapsPL...
assertTable
assertElementPresent ///[id='gb']//div/div
verifyTitle Google
verifyValue
verifyText ///[id='gb']//div/div +YouGmailImages +YouSearchYouTubeMapsPL...
verifyTable
verifyElementPresent ///[id='gb']//div/div
waitForTitle Google
waitForValue
waitForText ///[id='gb']//div/div +YouGmailImages +YouSearchYouTubeMapsPL...
waitForTable
waitForElementPresent ///[id='gb']//div/div
storeTitle Google
storeValue
storeText ///[id='gb']//div/div +YouGmailImages +YouSearchYouTubeMapsPL...
storeTable
storeElementPresent ///[id='gb']//div/div
```

**WebDriver playback**

The WebDriver playback feature in Selenium IDE lets you run tests in any one of the most popular web browsers: Chrome, Firefox, HtmlUnit, Internet Explorer, and Opera. By default, the WebDriver playback feature is turned off and is inactive. To run Selenium IDE scripts through WebDriver, turn on the WebDriver playback settings.
Launch the Selenium IDE and choose **Selenium IDE Options** from the **Options** menu. Switch to the **WebDriver** tab and select the **Enable WebDriver** checkbox. Now, restart the Selenium IDE to enable the WebDriver playback feature. However, on changing the browser name, restarting the IDE is not necessary. The idea discussed in these two paragraphs is shown in the following screenshot:

Prerequisites for the WebDriver playback feature
The following are the prerequisites that need to be fulfilled to enable the WebDriver playback feature:

- Download the latest Selenium Server standalone library (JAR) file
- Install Java to start the Selenium Server
- Download the latest drivers for popular browsers (**chromedriver**, **IEDriver**, and so on)

Selenium Server can be initialized manually from the terminal or Command Prompt. Open the terminal or Command Prompt, locate the Selenium Server JAR file, and run the command using the following syntax:

```
java -jar selenium-server-standalone-*version-number*.jar
```

Now you can run the command:

```
java -jar selenium-server-standalone-2.44.0.jar
```
Click on the Selenium IDE Play button to drive tests through WebDriver. To run tests in the Chrome web browser, replace the text firefox with chrome under the Selenium IDE options, as shown in the following screenshot:

By default, it is essential to set the ChromeDriver path in your working machine. Download the latest ChromeDriver extension from http://chromedriver.storage.googleapis.com/index.html?path=2.9/.

Follow these configuration steps to set the ChromeDriver extension path for different platforms.

**On Windows:**

1. Double-click and open the My Computer window.
2. Right-click anywhere on the window and select Properties.
3. Click on Advanced System settings.
4. Click on Environment Variables from System Properties.
5. Under System variables, select the variable named Path and click on the Edit button.
6. Now, extract the downloaded ChromeDriver package and copy the location path.
7. Paste the extracted location in Path (under System variables) and click on OK.

**On Linux:**

Open the terminal and run the following command:

```bash
$ wget http://chromedriver.storage.googleapis.com/2.7/chromedriver_linux64.zip
$ Unzip chromedriver_linux64.zip
$ cp chromedriver /usr/local/bin
$ chmod +x /usr/local/bin/chromedriver
```
On Mac:

1. Unzip/extract the zipped package (chromedriver_mac32.zip).
2. Copy and paste ChromeDriver to /usr/bin.
3. It will prompt you to enter the admin password; enter it to set the path.

Locator prioritization

Prioritization lets you prioritize locators while recording scripts. In general, this feature helps the user by giving high priority to generate scripts with respect to the user's preferred locators. For example, by changing the cssLocator order from the fifth to the first position, the further elements generated on the Selenium IDE's target will be in CSS, that is, the locatorType will be set to CSS by default.

An example of this is CSS = argument.

Launch the Selenium IDE, choose Options... from the Options menu, and switch to the Locator Builders tab. The left-hand pane will be mounted with a list of available locator builders, such as, ui, id, link, name, css, dom:name, xpath:link, xpath:img, xpath:attributes, xpath:idRelative, xpath:href, dom:index, and xpath:position. The list of available locator builders is shown in the following screenshot:
Drag and drop locator builders on the left-hand side to change their order. Finally, click on the OK button, and restart the Selenium IDE for the changes to take effect. To reset the default settings of the Selenium IDE, click on the Reset option found in the bottom-left corner of the Selenium IDE options window pane.

Avoiding Selenium export

Exporting test cases each and every time can bother the user. The Selenium IDE provides an excellent feature to avoid such exporting trouble by using a quick solution. In general, clicking on the Source toggle button under the Test Case pane displays the current test case in the Selenese language. The Selenium IDE transforms the existing Selenese language into a user-preferred script format, such as, Java/JUnit/WebDriver.

Launch the Selenium IDE and choose Options... from the Options menu. Make sure that the option Enable experimental features is selected and click on the OK button. Click on the Format option in the Options menu and select the preferred combination format.

For example, you can select the Java/JUnit/WebDriver option, as shown in the following screenshot.

Finally, restart the Selenium IDE for the changes to take effect. In the Selenium IDE, there is no support for exporting the test cases in TestNG with the WebDriver (Java/TestNG/WebDriver) combination format.
The tab will automatically switch to the **Source** view on disabling the **Table** toggle button, as shown in the following screenshot:

![Selenium IDE Source View](image)

**The Selenium IDE clipboard**

Copying snippets through **Clipboard Format** is one of the quickest ways to obtain instantly generated scripts. Here, a snippet may contain one or two lines of code. The following screenshot displays different types of export formats available under the **Clipboard Format** option:

![Selenium IDE Clipboard Formats](image)

Launch the Selenium IDE, hover your mouse over the **Clipboard Format** option from the **Options** menu, and select the preferred combination format.
An example of a combination format is **Java/JUnit 4/WebDriver**. In general, the HTML snippet is set as default. The following screenshot shows a row being copied from the **Test Case** pane:

![Screenshot showing copied row from Test Case pane](image)

Copy the following row from the **Test Case** pane (as shown in the preceding screenshot) and paste it as a code snippet:

```java
    driver.findElement(By.id("gbqfq")).clear();
    driver.findElement(By.id("gbqfq")).sendKeys("prashanthSams");
```

### Data Driven tests

Parameterization is a part of the Data Driven technique for retrieving values from an external data source as input. In general, the Data Driven tests are used to verify the actual and expected values from an external data source. The Selenium IDE plays a major role in parameterization, as it operates with different sets of permutations and combinations. Let's see how we can use a JavaScript file as a data source for Data Driven tests. The following is the JavaScript syntax for parameterization:

```javascript
    varname = "value"
```

For example, create a JavaScript (.js) file (`Datasource.js`) that includes the following keywords:

```javascript
    Search1 = "PrashanthSams"
    Search2 = "Selenium Essentials"
```
Launch the Selenium IDE and choose **Options...** from the **Option** menu. Choose **Options...** Now, browse through Selenium IDE extensions and select the `.js` file created earlier (`Datasource.js`), as shown in the following screenshot:

Finally, restart the Selenium IDE to effect the changes. Initialize, store, and fetch values from the `.js` file one by one using the `storeEval` command, as follows:
Here, Search1 and Search2 are the two variables that retrieve the respective keywords from the JavaScript file. These values are again stored in the new variables, GoogleSearch1 and GoogleSearch2, as shown in the preceding screenshot.

**User-defined JavaScript methods**

The IDE actions, accessors, and assertions can be user-defined and customized. To achieve this, the user is supposed to add JavaScript methods to the Selenium object prototype and the PageBot object prototype. In general, the Selenium IDE verifies the user-defined JavaScript methods on launching the IDE. Selenium Core extensions (user-extensions.js) in **Options...** give support to upload the user-defined JavaScript files.

Let's discuss this with an example that involves step-by-step instructions, as follows:

1. Refer to the Google site [https://sites.google.com/site/seleniumworks/selenium-ide-data-driven](https://sites.google.com/site/seleniumworks/selenium-ide-data-driven) to download the following JavaScript files:
   - datadriven.js
   - goto_sel_ide.js
   - user-extensions.js

2. Launch the Selenium IDE and choose **Options...** from the **Option** menu, **Option | Options...**. Now, navigate to **Selenium Core extensions** and upload the JavaScript files (user-extensions.js, goto_sel_ide.js and datadriven.js), as shown in the following screenshot. Finally, restart the Selenium IDE for the changes to take effect.
3. In the Selenium IDE, the XML file is used as a data source to store values, whereas Datadriven.js is designed to support the XML file format.

Here is the syntax for XML file formatted data source:

```xml
<testdata>
  <test varname="value" />
  <test varname="value" />
  <test varname="value" />
</testdata>
```

Create an XML file with the extension .xml (data.xml). Here, varname is the variable name, and value refers to the keyword under the <test> tag.

Let's create an XML file with the .xml extension (data.xml). For example, refer to the following code snippet:

```xml
<testdata>
  <test phrase="selenium essentials" />
  <test phrase="seleniumworks.com" />
  <test phrase="prashanthsams" />
</testdata>
```

4. Take a look at these details: `loadTestData` is a user-defined command that fetches the XML data source, while and `endWhile` do looping, whereas `nextTestData` checks for the data from the next row in the data source. The user can add any number of JavaScript methods. The following screenshot shows this step in detail:
Selenium IDE JavaScript functions

In addition to user-defined JavaScript commands introduced through `user-extensions.js`, the Selenium IDE allows the user to create JavaScript queries or functions directly in the Target field. For example, let's run a Google search by getting a random number between 1 and 100, as follows:

The following HTML source tags let you convert the steps into runnable test scripts:

```html
<tr>
  <td>store</td>
  <td>javascript('Random number ' + Math.floor(Math.random() * 100); search</td>
</tr>
<tr>
  <td>echo</td>
  <td>${search}</td>
</tr>
<tr>
  <td>open</td>
  <td>/</td>
</tr>
<tr>
  <td>type</td>
  <td>id=gbqfq</td>
</tr>
</tr>
```

The following HTML source tags let you convert the steps into runnable test scripts:

```html
<tr>
  <td>store</td>
  <td>javascript('Random number ' + Math.floor(Math.random() * 100);)</td>
</tr>
<tr>
  <td>search</td>
</tr>
<tr>
  <td>echo</td>
  <td>${search}</td>
</tr>
<tr>
  <td>open</td>
  <td>/</td>
</tr>
<tr>
  <td>type</td>
  <td>id=gbqfq</td>
</tr>
</tr>
```
Simple JavaScript execution

The predefined Selenium IDE JavaScript command `runScript` is a very powerful command that lets the user execute simple JavaScript functions directly from the IDE, for example, `javascript{alert("Hello!")};`

Let's see how we can disable an active textbox and enable an inactive textbox using the following code snippet:

```javascript
document.getElementsByName('****')[0].setAttribute('disabled', '');
document.getElementsByName('****')[0].removeAttribute('disabled');
```

Mouse scroll

The `scroll` event is currently unavailable in the Selenium IDE. However, the `user-extensions.js` file includes a JavaScript method that lets you scroll the mouse through the web page.

Refer to the Google site https://sites.google.com/site/seleniumworks/selenium-ide-tricks to download user-extensions.js. This user extension file includes IDE commands like `while`, `endWhile`, `gotoIf`, `gotoLabel`, and `push`. Increase the value to 10 based upon the vertical length of the web page, as shown in the following screenshot:
Parameterization using arrays

The Selenium IDE command `storeEval` is used to store values in a variable while running scripts, whereas `storedVars` is a JavaScript associate array with string indexes containing variables. In the following example, `storeEval` reserves the list of rivers in an array, and `getEval` is a command for initiating and incrementing the values. Some of the commands used in this section are purely user-defined, such as `while`, `endWhile`, and so on. Here, the `endWhile` command is used to break the loop once the value inside the array reaches the maximum limit. The following screenshot gives us a clear idea of what is being discussed here:

<table>
<thead>
<tr>
<th>Command</th>
<th>Target</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getEval</code></td>
<td></td>
<td><code>i = 0</code></td>
</tr>
<tr>
<td><code>while</code></td>
<td><code>i &lt; storedVars[&quot;rivers&quot;].length</code></td>
<td></td>
</tr>
<tr>
<td><code>storeEval</code></td>
<td><code>i</code></td>
<td><code>data</code></td>
</tr>
<tr>
<td><code>echo</code></td>
<td><code>javascript(storedVars[&quot;rivers&quot;][storedVars[&quot;data&quot;])</code></td>
<td></td>
</tr>
<tr>
<td><code>getEval</code></td>
<td><code>i++</code></td>
<td><code>x</code></td>
</tr>
<tr>
<td><code>endWhile</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Let's see another example of advanced parameterization concepts using the Selenium IDE. Refer to the Google site https://sites.google.com/site/seleniumworks/selenium-ide-tricks to download the `user-extensions.js` file. The following screenshot captures the essence of this discussion:
In this example, the values are pushed into the array manually, and it does a Google search one by one.

**Selenium Builder**

Selenium Builder is a record-and-playback tool similar to the Selenium IDE and is an extension of the Firefox web browser. It has some unique features that the Selenium IDE doesn't support, for example GitHub integration to export and commit test suites/cases in GitHub and TestingBot integration to run tests in the cloud. It also provides more language support than the Selenium IDE, including for languages such as JSON, Java/TestNG, NodeJS WD, NodeJS Mocha, and NodeJS Protractor. Selenium Builder is expected to be the future of the Selenium IDE, with advanced features. Here's a screenshot of Selenium Builder:
Recording and playback

Upon confirmation of the addition of the Selenium Builder extension into the Firefox browser, open the web page that is being tested (such as www.google.com). There are a number of ways to open Selenium Builder, such as:

- Right-click on the web page and select Launch Selenium Builder.
- Choose Selenium Builder from the Tools menu, that is, Tools | Web Developer | Launch Selenium Builder
- Alternatively, you can use the Ctrl + Alt + B shortcut keys

Selenium Builder is supposed to have an option Selenium 2 to record WebDriver test scripts. The user can easily identify the page being actively recorded, as it has a green-colored tab, as shown in the following screenshot:

Selenium Builder allows you to have control over the web app to start recording test scripts. It allows the users to verify their tests using a button, Record a verification. Clicking on the verification button highlights the text to be verified all over the page on mouseover. The mouseover function is an excellent feature of Selenium Builder that helps the user to add mouse hover functionality whenever needed. Select the Record mouseovers checkbox to record mouseover actions. The following screenshot shows this functionality:

Stop recording the scripts and export (File | Export) the tests in the preferred combination format.
Data Driven tests

In general, Selenium Builder is capable of importing test scripts saved by the Selenium IDE, since both the IDEs’ native format is Selenese. Exporting Selenium scripts in the Java/TestNG/WebDriver format, which avoids all the extra human effort when working with TestNG, is feasible in Selenium Builder. TestNG is one of the most popular unit testing frameworks and is similar to JUnit for Java bindings.

Despite the fact that Selenium Builder is a record-and-playback tool, it allows users to perform basic Data Driven tests by receiving input from the data source. The values can also be stored temporarily in Selenium Builder using the Manual Entry option (Data | Manual Entry). To do so, create a variable and assign the value, as shown in the following screenshot:

Selenium Builder comprises higher-level support for Data Driven testing that lets you drive tests using the JSON, XML, and CSV file formats. CSV file formats can be used to fetch data files with large volumes. Let’s see some of the file formats involving Data Driven tests.

Testing using a JSON file

The following is the syntax for a JSON file formatted data source:

```json
[
    { "varname": "value", "varname": "value", ... },
    { "varname": "value", "varname": "value", ... },
    ...
]
```
For example, create a JSON file with the .json extension (data.xml). Here, foo is the variable name, whereas value is defined with the keywords prashanth and sams, as follows:

```
[{"foo": "prashanth"}, {"foo": "sams"}]
```

To address the values stored in a JSON file, it is recommended to call the variables, for example, `${foo}`, in the pre-recorded steps, as shown in the preceding screenshot.

### Testing using an XML file

The following is the syntax for an XML file formatted data source:

```
<testdata>
    <testvarname="value" />
    <testvarname="value" />
    <testvarname="value" />
</testdata>
```

For example, create an XML file with the .xml extension (data.xml). Here, search is the variable name, and value refers to the keyword under the `<test>` tag, as shown in the following code snippet:

```
<testdata>
    <test search="selenium essentials" />
    <test search="prashanthsams" />
```
Selenium Builder on the cloud

Selenium Builder allows users to run cross-browser tests on the cloud directly from the IDE interface. To integrate Selenium Builder with Sauce and enable export and playback scripts on Sauce OnDemand, it's necessary to install the Selenium Builder Sauce plugin first. Launch Selenium Builder, click on Manage plugins, and install the Sauce for Selenium Builder plugin. Also, users should create a Sauce account before running cloud-based Selenium tests. For further details on Selenium Builder on the cloud, refer to https://saucelabs.com/. The following screenshot shows us the Plugins page:
Obtain the Sauce access key after logging in to your Sauce account as a real user or by clicking look up access key in Selenium Builder. The access key is unique for each user.

Choose Run on Sauce OnDemand from the Run menu. Make sure that Sauce Settings is furnished before running the test cases. All you need to do is enter the Sauce Username, Sauce Access Key, Browser, and OS versions. Finally, log in to Sauce and verify the test results. The tests are recorded in video and images for user preview.

Summary

In this chapter, you learned about the Selenium IDE functions, along with Selenium Builder, and observed how to handle the Selenium IDE to automate simple tests.

In the next chapter, we will discuss advanced compatibility-testing techniques using Selenium WebDriver. It lets you drive tests on different browsers.
Where to buy this book

You can buy Selenium Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.