Tech Mahindra

Let's Recap

- What is Test Automation ? and Test Automation Tools...
- Selenium Test Automation tool.
- Components of Selenium Test Automation tool - Selenium IDE, RC, WD and Grid.
- History of Selenium tools...
- Selenium vs QTP and vice versa...
- Selenium browsers and environments...
- Selenium IDE - GUI for creating/recording/playing test scripts/cases.
- Add-on - Firefox. Selenium IDE..
- Selenium IDE tools/menu options.
- Creating test cases and test suites using IDE....
- Recording/playing the test cases/suites...
- Understanding the IDE tabs/windows...

Agenda

- Introduction to Selenium Web Driver
- Selenium Web Driver Architecture
- Selenium Web Driver API
- Finding elements ï Locating UI Elements
- Locating UI Elements using CSS and XPath
- Web Driver : Accessing form Elements.
- Web Driver : Moving between windows, frames, alerts, inline frames.

Tech Mahindra

What is Selenium ?

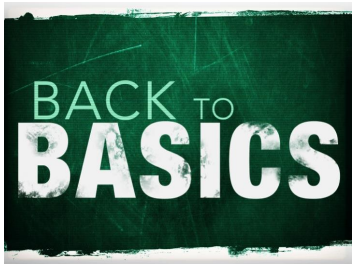
Automated testing suite for web applications

Who created Selenium?

JASON HUGGINS

Selenium Components

- Selenium IDE
- Selenium RC
- Selenium WebDriver
- Selenium Grid



Selenium Download & Installation

http://www.seleniumhq.org/download/

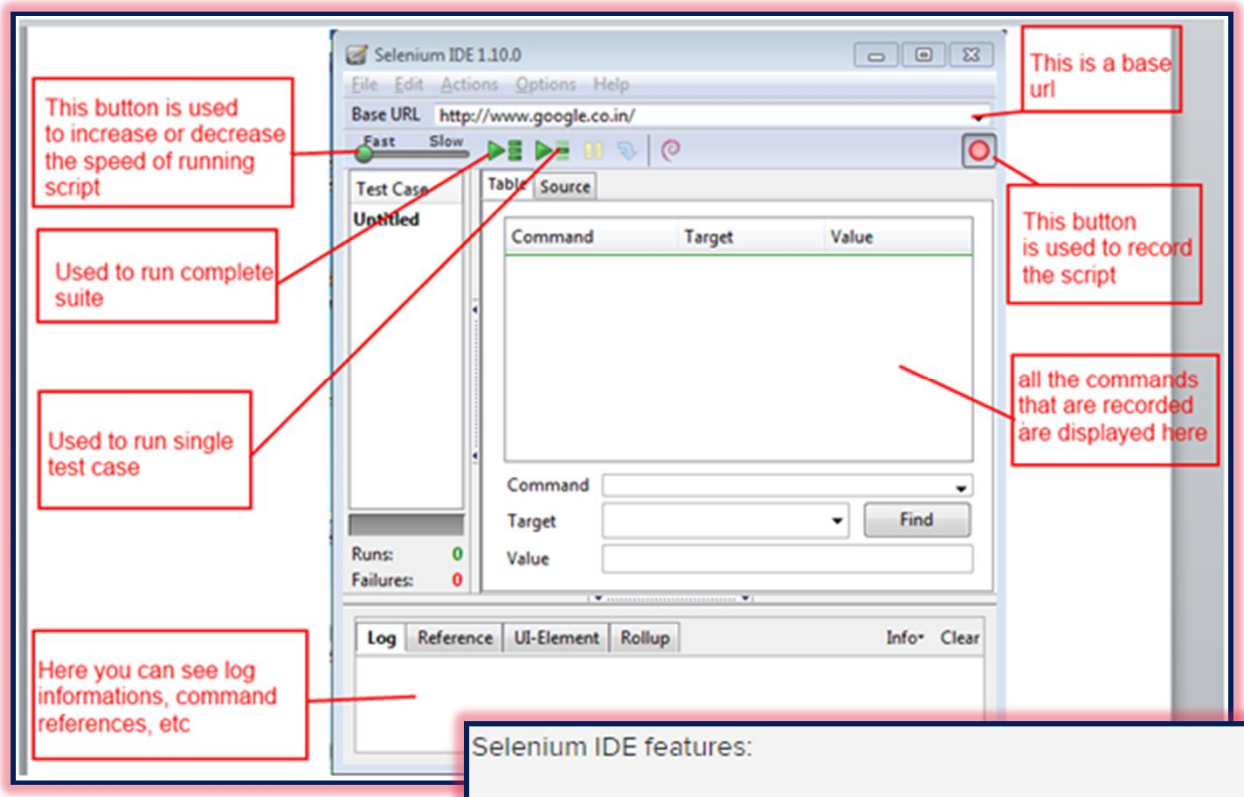
Features Of Selenium

- Supports Multiple browser : IE, Firefox, Mozilla, Safari & etc
- Also Supports Programming languages : Java,.Net, Ruby, Perl, PHP and Python

why selenium is better than qtp ?

- QTP is Commercial & Selenium is Opensource Free Tool
- QTP Supports only Windows OS & Selenium Supports Multiple OS
- QTP Supports VB & Java Scripts only bu Selenium Supports Multiple Programming Scripts

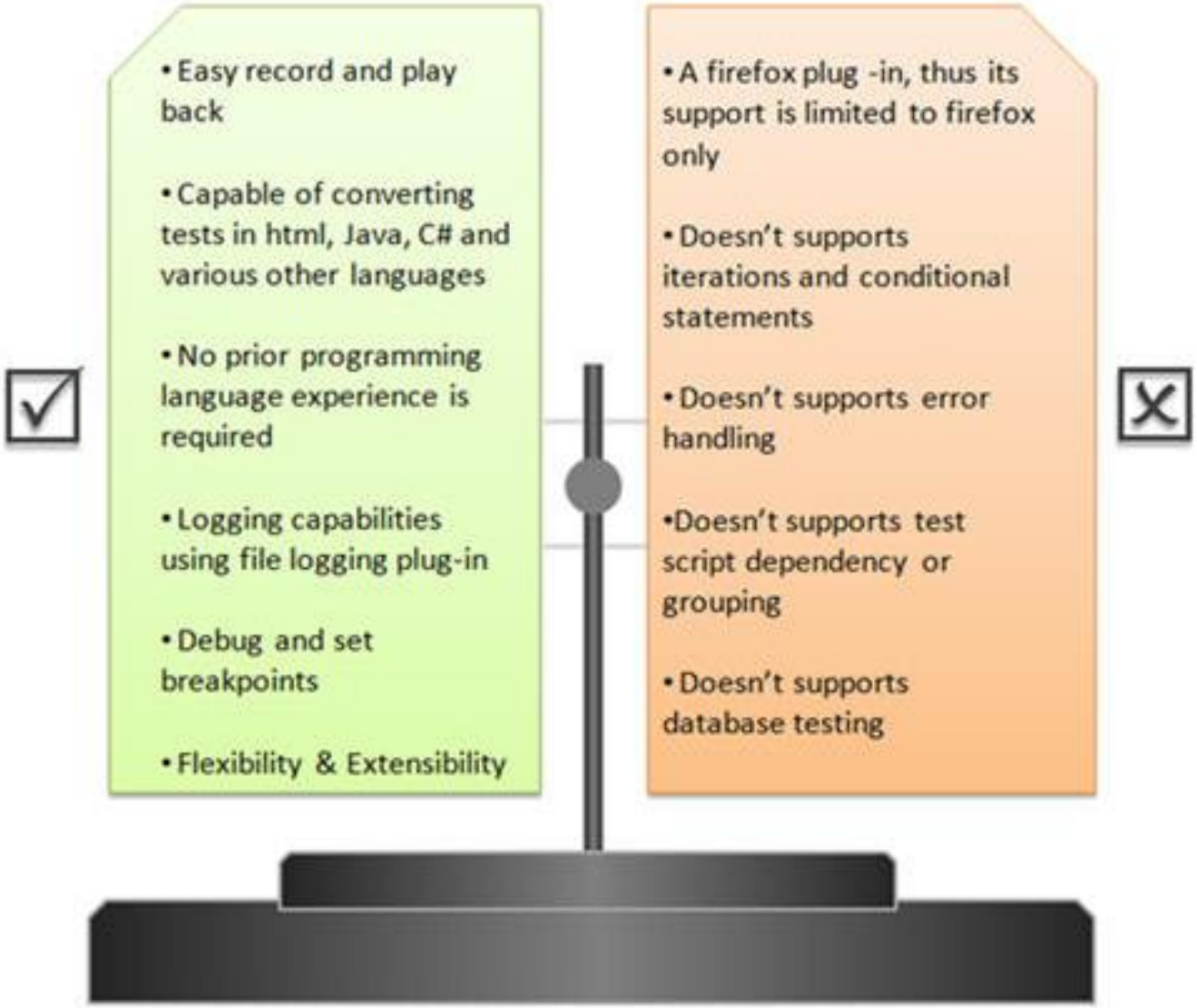
Selenium IDE and Features



Selenium IDE features:

- record/play mechanism: create/edit tests by recording the user actions on the page
- Debugging feature with step-by-step and breakpoints
- Page abstraction capabilities
- Automatic assertions for the page title
- An extensibility feature allowing the use of plugins or user extensions that extend the capabilities of Selenium IDE

Selenium IDE Advantages & Disadvantages



Tech Mahindra

Let's have

Drive

of

Selenium

Webdriver

Selenium

Selenium is a portable software testing framework for web applications.

Selenium IDE and Locators

Web Driver : Locating Elements

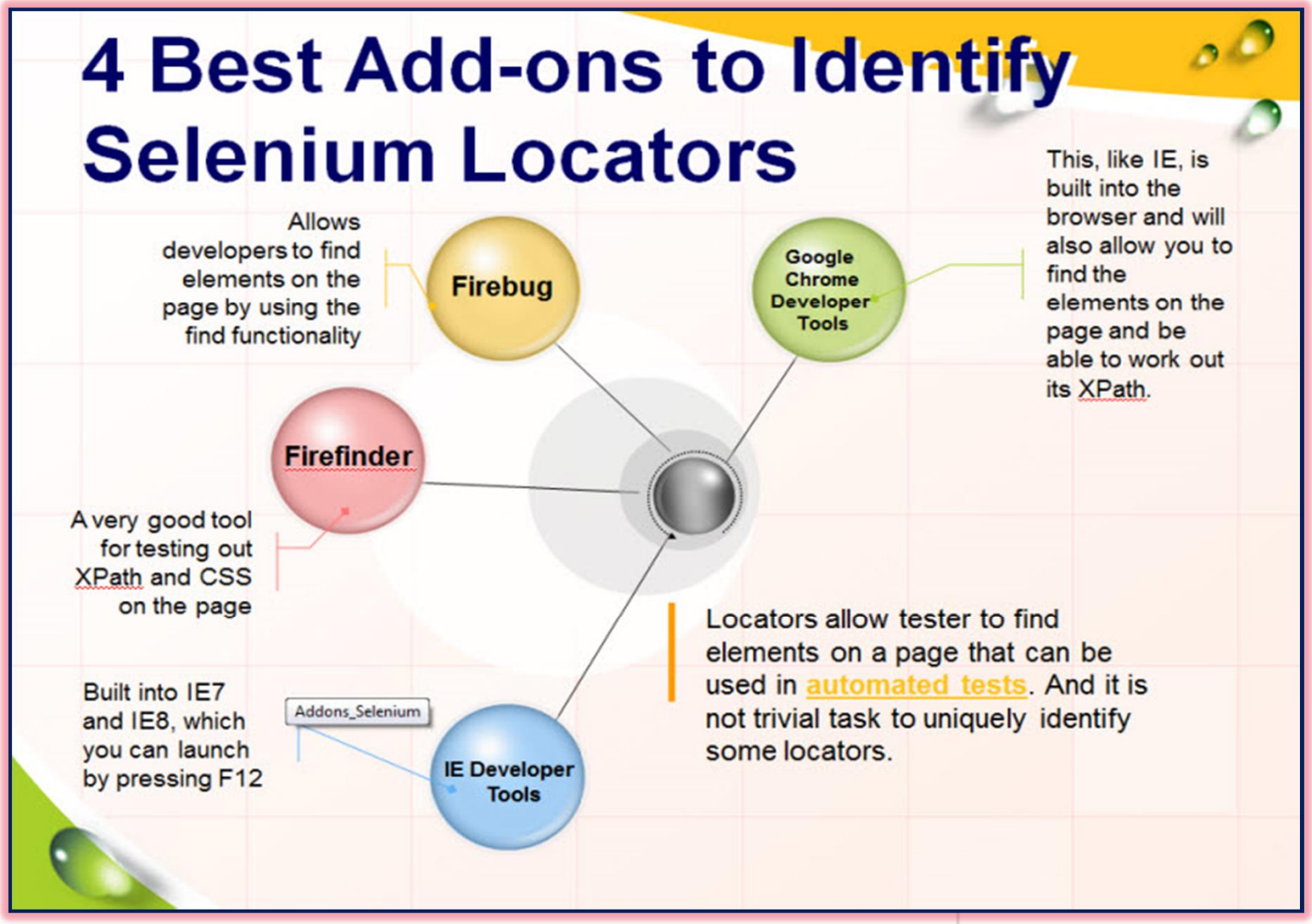
Tech Mahindra

- *Locators tell Selenium IDE which GUI elements (say Text Box, Buttons, Check Boxes etc) its needs to operate on.*

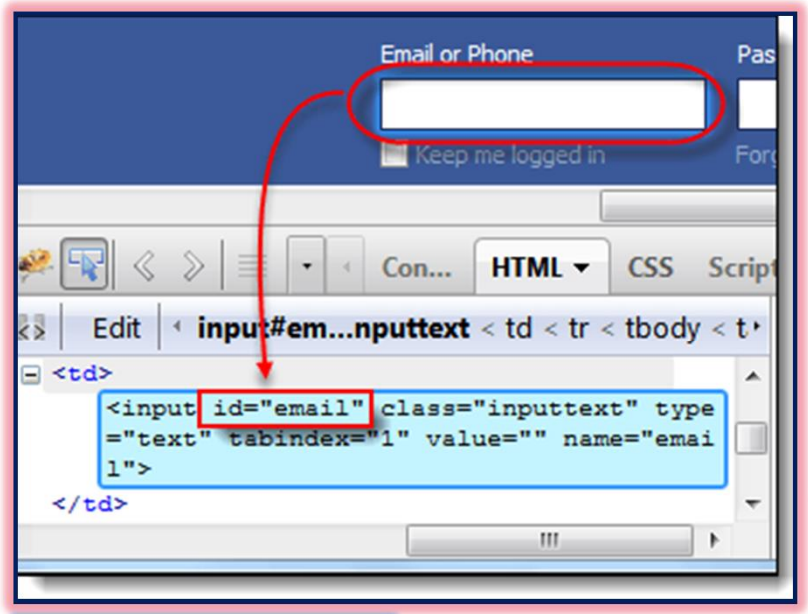
The different types of locator are:

- ID
- Name
- Link Text
- CSS Selector
  - Tag and ID
  - Tag and class
  - Tag and attribute
  - Tag, class, and attribute
  - Inner text
- DOM (Document Object Model)
  - getElementById
  - getElementsByName
  - dom:name
  - dom: index
- XPath

Locating Elements : Add-ons

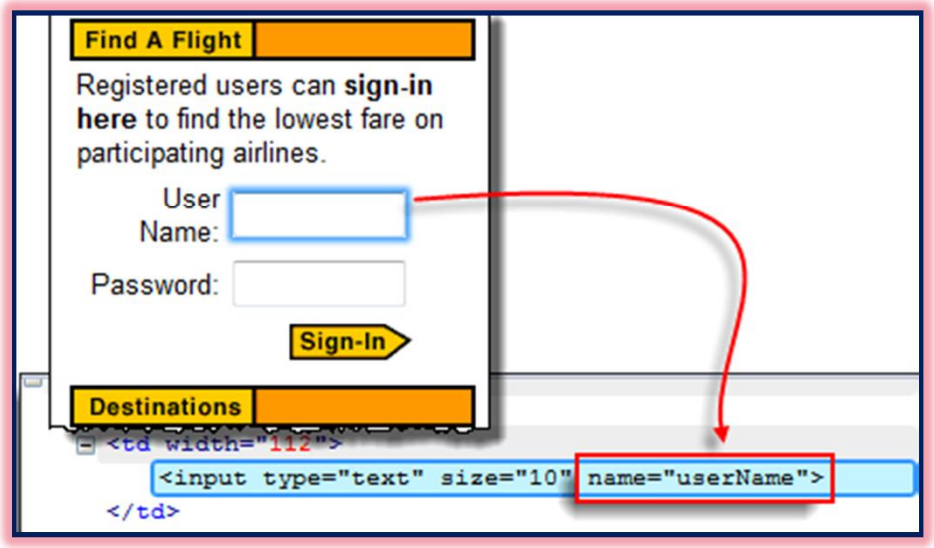


Selenium : Locating Elements Cont..



Locating by ID

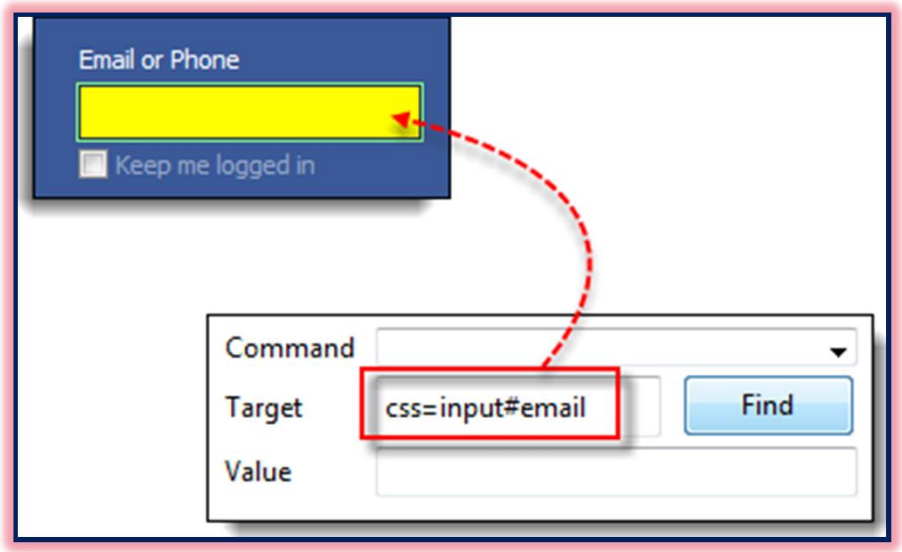
Locating by Name



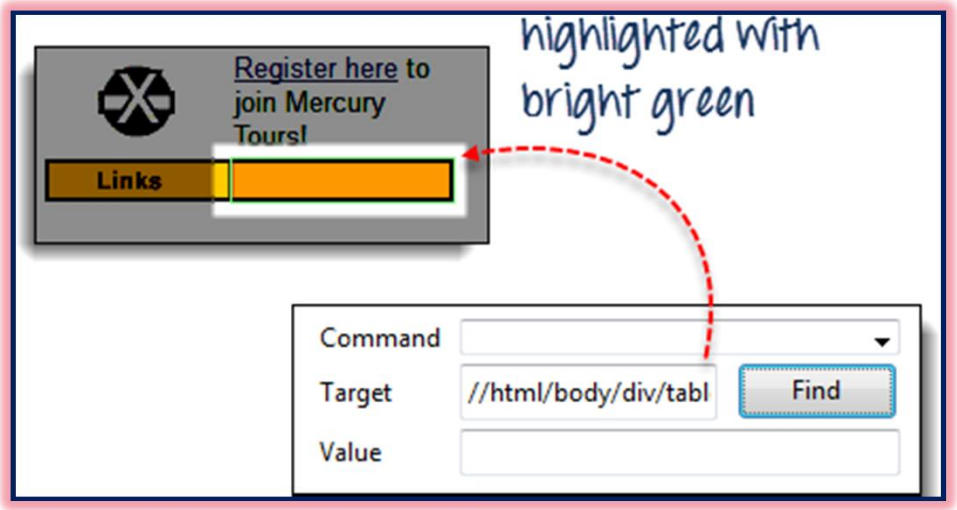
Selenium : Locating Elements Cont..

Locating by Link Text

Locating by CSS Selector

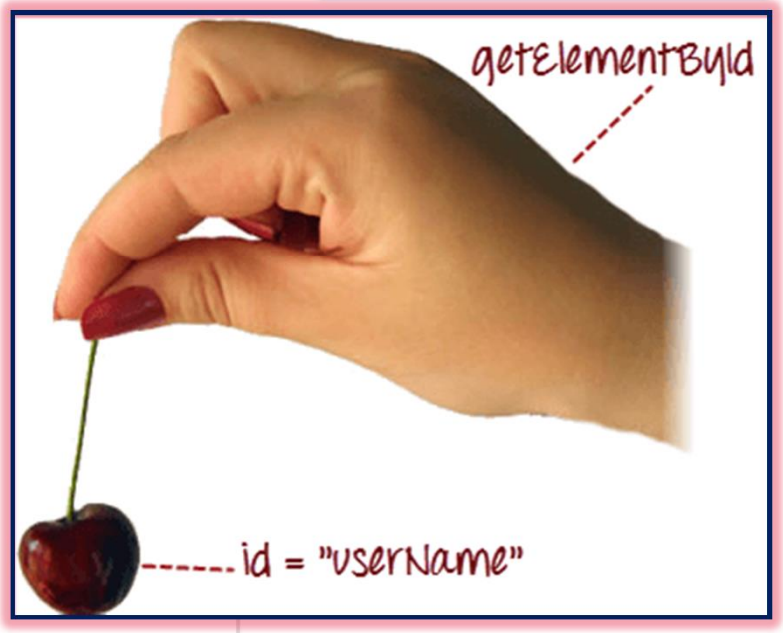


Selenium : Locating Elements Cont..



Locating by XPath

Locating by DOM



Selenium : Locating Elements Summary

Tech Mahindra

Syntax for Locator Usage

Method	Target Syntax	Example
By ID	id= *id_of_the_element*	id=email
By Name	name=*name_of_the_element*	name=username
By Name Using Filters	name=*name_of_the_element* *filter=value_of_filter*	name=tripType value=oneway
By Link Text	link=*link_text*	link=REGISTER
Tag and ID	css=*tag#id*	css=input#email
Tag and Class	css=*tag.class*	css=input.inputtext
Tag and Attribute	css=*tag[attribute=value]*	css=input[name=lastName]
Tag, Class, and Attribute	css=*tag.class[attribute=value]*	css=input.inputtext[tabindex=1]

Selenium WebDriver

- Browser Automation Framework
- It automates browser's
- http://docs.seleniumhq.org/

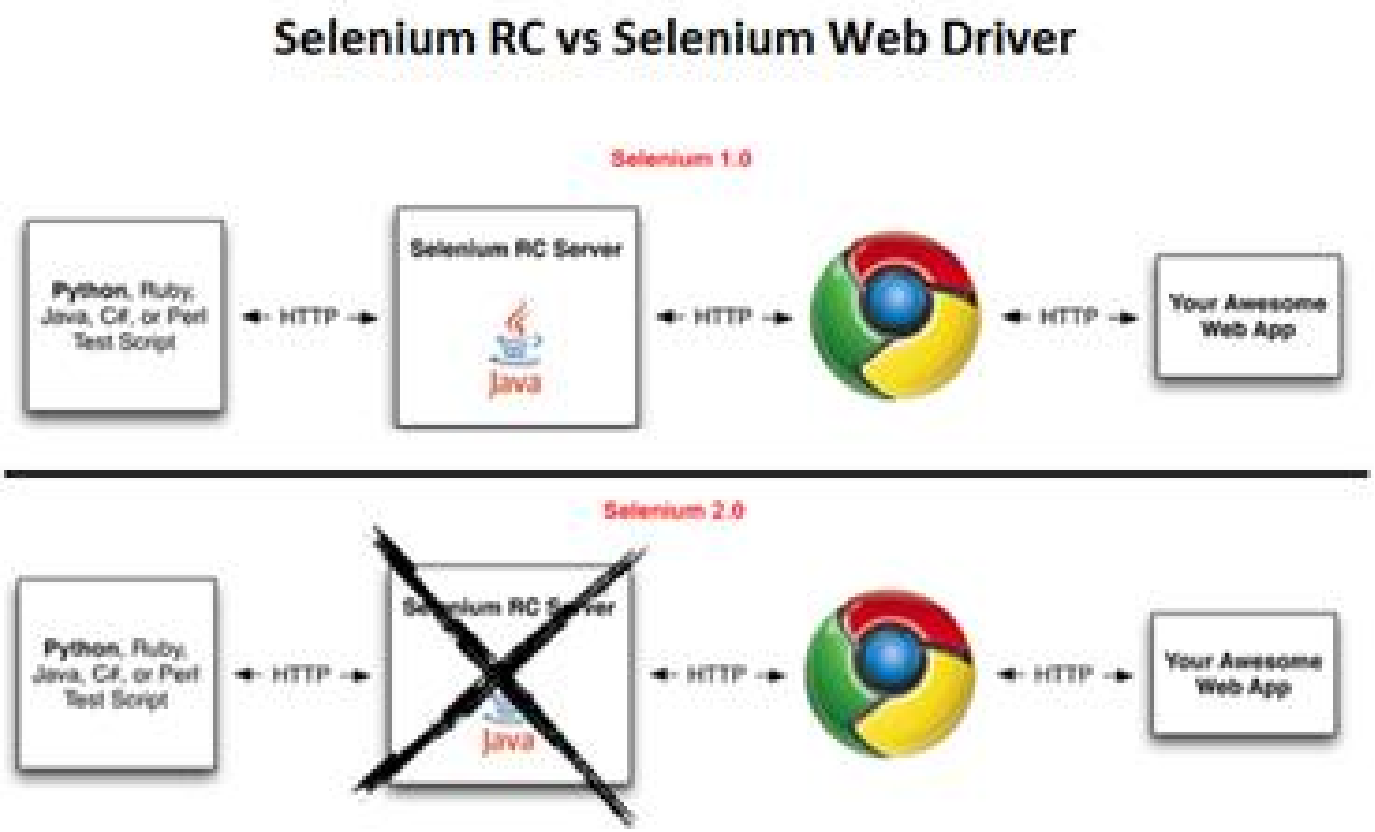
WebDriver : Introduction

- WebDriver is a ***tool for testing web applications across different browsers using different programming languages***.
- WebDriver allows you to use a programming language of your choice in designing your tests.
- WebDriver is faster than Selenium RC because of its simpler architecture.
- WebDriver directly talks to the browser while Selenium RC needs the help of the RC Server in order to do so.
- WebDriverôs API is more concise than Selenium RCôs.
- WebDriver can support HtmlUnit while Selenium RC cannot.
- The drawbacks of WebDriver are:
  - It cannot readily support new browsers, but Selenium RC can.
  - It does not have a built-in command for automatic generation of test results.

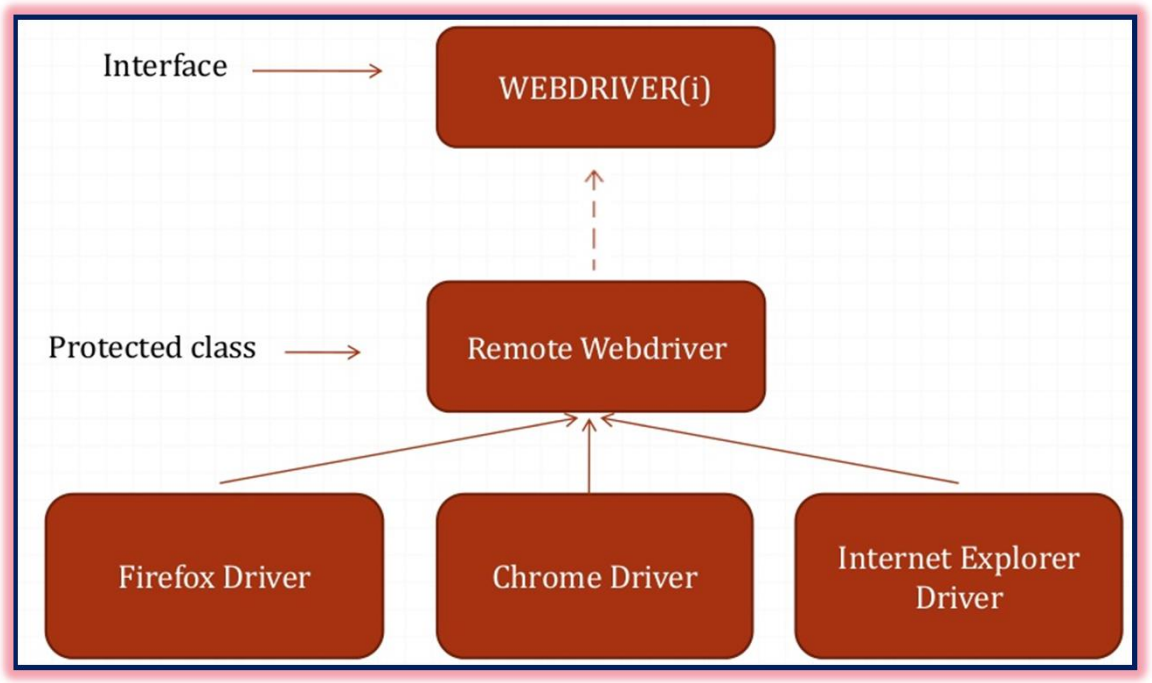
WebDriver : Workflow

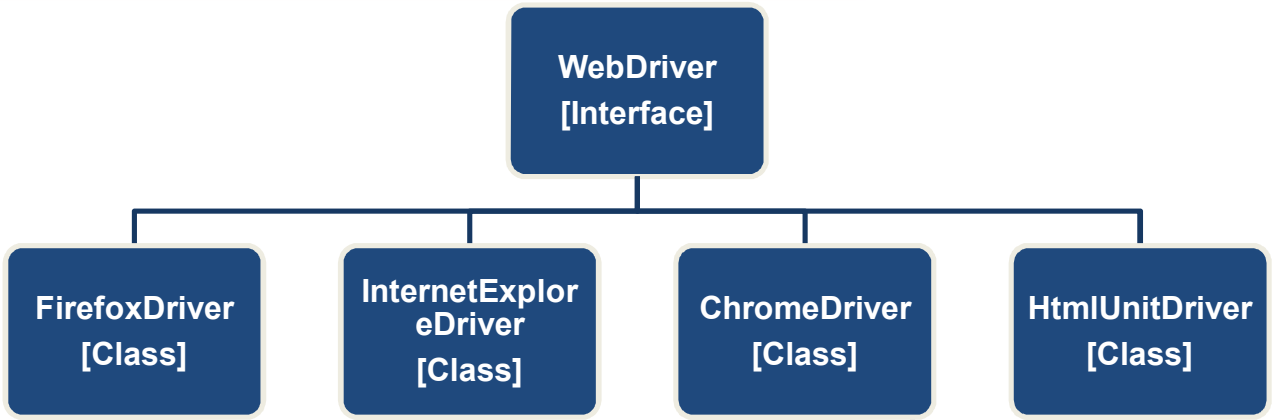
- *Selenium Web driver directly interacts with browsers while executing the test script.*
- *Automation test scripts are executed using selenium core on browsers.*
- *In Web driver, each browser has its own driver instance to execute the automation test scripts.*

Web Driver Vs Selenium RC

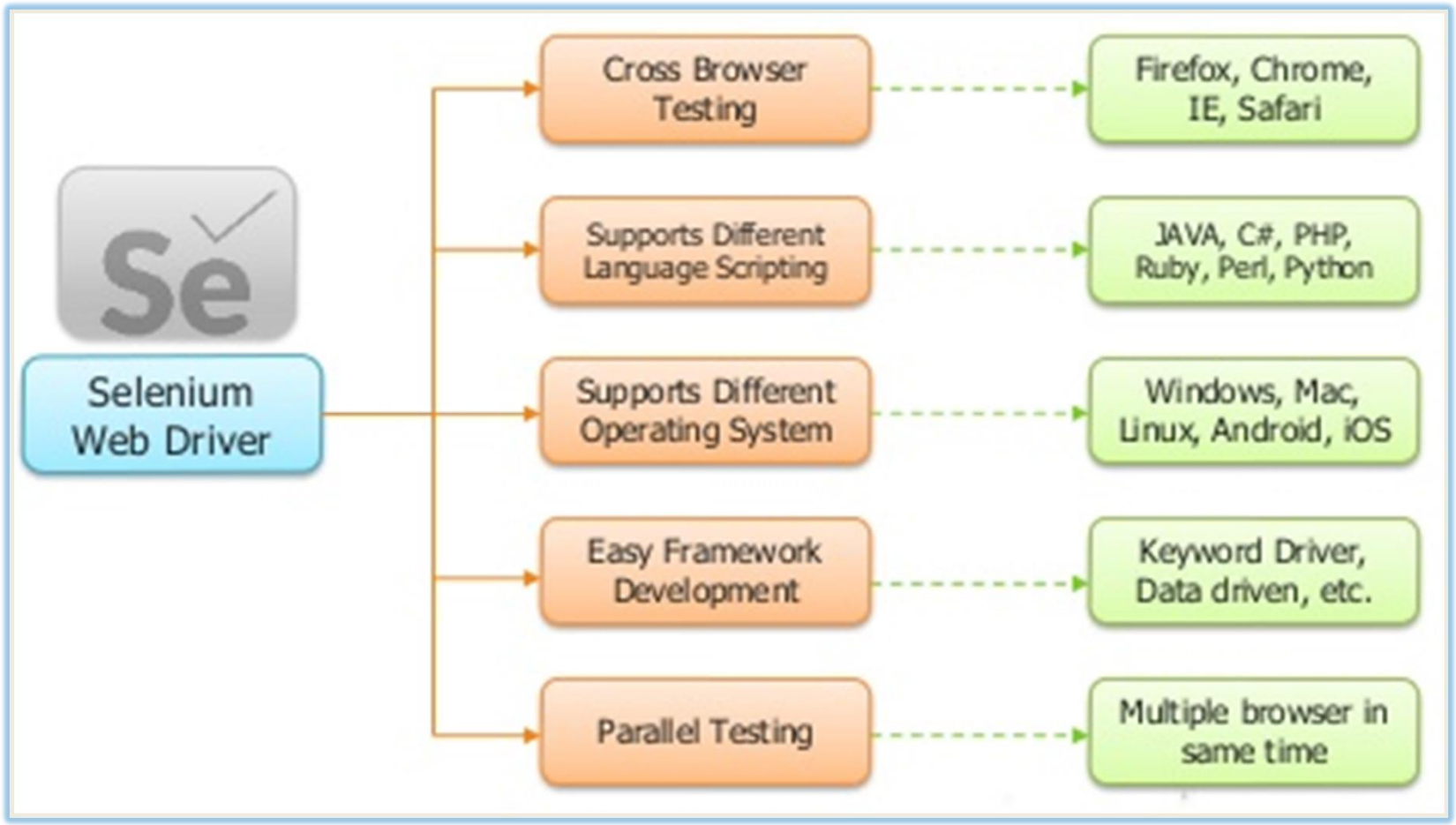


Web Driver Architecture





Web Driver : Summary



Web Driver : Locating Elements

FindElement vs. FindElements

findElement()

0 matches: throws exception (org.openqa.selenium.NoSuchElementException)

1 match: returns WebElement instance

2+ matches: returns only first appearance in DOM

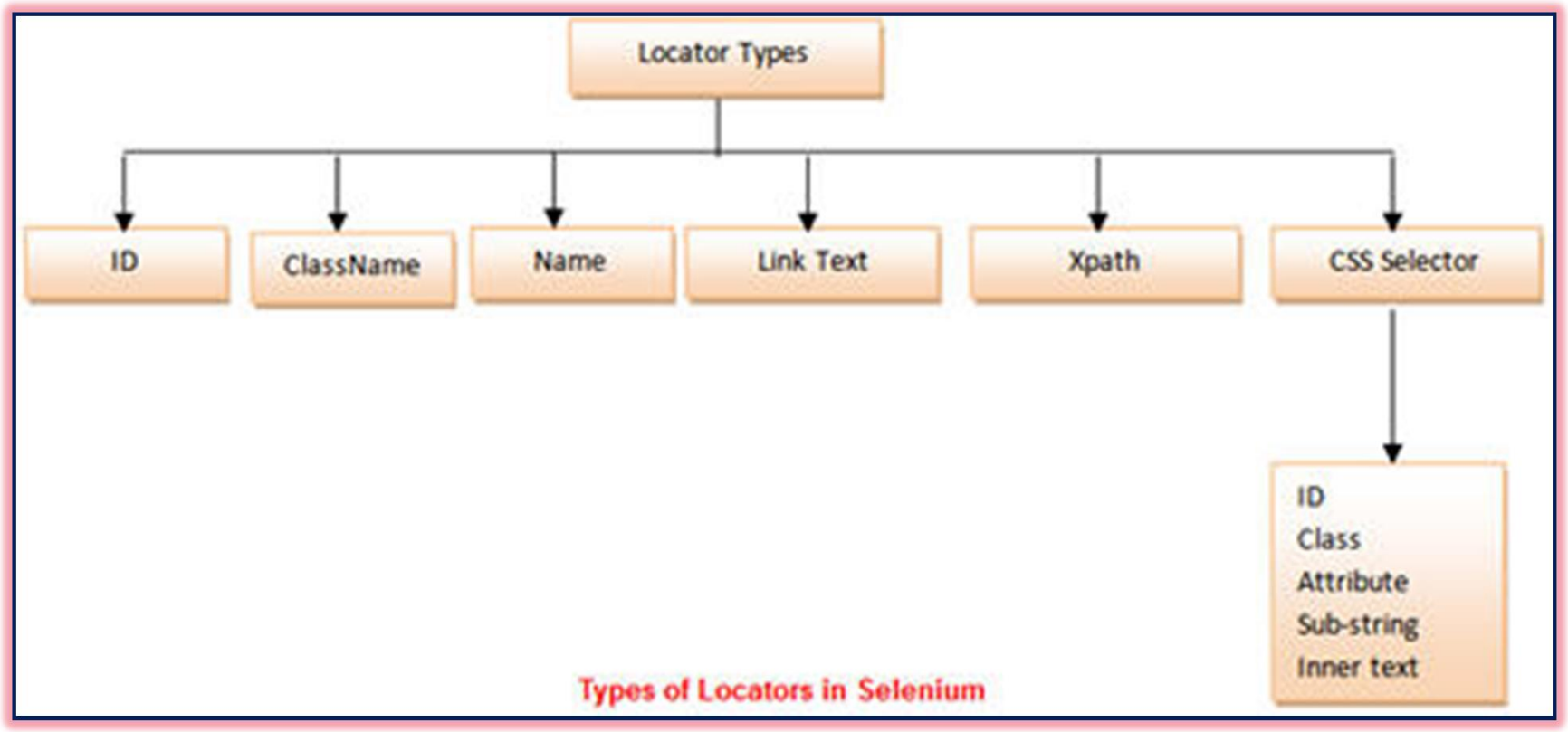
findElements()

0 matches: returns an empty list

1 match: returns list of 1 WebElement instance

2+ matches: returns list with all matching instances

Web Driver : Locating Elements Snapshot



Web Driver : Locating Elements

- **Locating UI Elements (WebElements)**
  - **By ID**
    - *WebElement element = driver.findElement(By.id("coolestWidget"));*
  - **By Name**
    - *WebElement cheese = driver.findElement(By.name("cheese"));*
  - **By Class Name**
    - *WebElement element = driver.findElement(By.className("cheese"));*
  - **By Tag Name**
    - *WebElement frame = driver.findElement(By.tagName("iframe"));*
  - **By Link Text**
    - *WebElement cheese = driver.findElement(By.linkText("cheese"));*

Web Driver : Locating Elements Contd..

- **By Partial Link Text**
  - *WebElement cheese = driver.findElement(By.partialLinkText("cheese"));*
- **By CSS**
  - *WebElement cheese =*

    driver.findElement(By.cssSelector("#foodspan.dairy.aged"));
- **By XPATH**
  - *List<WebElement> inputs = driver.findElements(By.xpath("//input"));*

Web Driver : Locating Elements Sample

```
package mypackage;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class myclass {

    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        String baseUrl = "http://www.facebook.com";
        String tagName = "";

        driver.get(baseUrl);
        tagName = driver.findElement(By.id("email")).getTagName();
        System.out.println(tagName);
        driver.close();
        System.exit(0);
    }
}
```

Web Driver : Close Vs quit

Closing and Quitting Browser Windows

close() *Sample usage:*	• Needs no parameters • **It closes only the browser window that WebDriver is currently controlling.**
quit() *Sample usage:*	• Needs no parameters • **It closes all windows that WebDriver has opened.**

Web Driver : CSS Selectors

- When we don't have an option to choose Id or Name, we should prefer using CSS locators as the best alternative
- CSS is "Cascading Style Sheets" and it is defined to display HTML in structured and colorful styles are applied to webpage.
- ***Selectors*** *are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document*.
- CSS has more Advantage than Xpath and is much more faster and simpler than the Xpath.
- In IE Xpath works very slow, where as CSS works faster when compared to Xpath.
- Syntax :-
  - ***tagName[attributename = attributeValue]***
  - Example 1: input[id=email]
  - Example 2: input[name=email][type=text]

CSS Selectors : Examples

Tech Mahindra

Å **Example – 1:**

<button class="submit btn primary-btn flex-table-btn js-submit" type="submit"
style="background-color: rgb(85, 172, 238);"> Log in **</button>**

Example 1: css=.primary-btn

Example 2: css=.btn.primary-btn

Example 3: css=.submit.primary-btn

The above can be written like below in selenium

WebElement ele1 = driver.findElement(By.cssSelector(".primary-btn"));

WebElement ele2 = driver.findElement(By.cssSelector(".btn.primary-btn"));

WebElement ele3 = driver.findElement(By.cssSelector(".btn.primary-btn"));

- **Example – 2 :**

css=input[id=email]; *In selenium we can write it as*

WebElement Email = driver.findElement(By.cssSelector("input[id=email]"));

Email.SendKeys("hello@sampleemail.com");

Web Driver : CSS Selectors

Tech Mahindra

- Using CSS locators, we can also locate elements with **sub-strings**, which are really help full when there are dynamically generated ids in webpage.
  - There are there important special characters:
    1. ***'^' symbol, represents the starting text in a string.***
    2. ***'$' symbol represents the ending text in a string.***
    3. ***'*' symbol represents contains text in a string***

- **CSS Locators for Sub-string matches(Start, end and containing text) in selenium**
  /*It will find input tag which contains 'id' attribute starting with 'ema' text. Email starts with 'ema' */

  css=input[id^='ema']

- **CSS Element locator using child Selectors**
  /* First it will find Form tag following remaining path to locate child node.*/

  css=form<label>input[id=PersistentCookie]

Web Driver : XPath

- XPath is designed to allow the navigation of XML documents, with the purpose of selecting individual elements, attributes, or some other part of an XML document for specific processing
- The Extensible Markup Language (XML) is the context in which the XML Path language, XPath, exists.
- XML provides a standard syntax for the markup of data and documents.
- *Xpath is a query language for selecting nodes from an XML document*.
- *Xpath is based on a tree representation of the XML document and provides the ability to navigate around the tree.*

Web Driver : Types of XPath

- In XPath the starting point is called the context node.
- **Absolute Xpath**
  - Absolute XPath starts with the root node or a forward slash (/).
  - The advantage of using absolute is, it identifies the element very fast.
  - Disadvantage here is, if any thing goes wrong or some other tag added in between, then this path will no longer works.
- **Example:**

  If the Path we defined as

  1. /html/head/body/table/tbody/tr/th

     If there is a tag that has added between body and table as below.

  2. /html/head/body/form/table/tbody/tr/th

     The first path will not work as 'form' tag added in between

Web Driver : Types of XPath

- **Relative Xpath.**
- A relative xpath is one where the path starts from the node of your choice - it doesn't need to start from the root node.
  - It starts with Double forward slash(//)
  - **Syntax:**
    //table/tbody/tr/th
- Advantage of using relative xpath is, you don't need to mention the long xpath, you can start from the middle or in between.
- Disadvantage here is, it will take more time in identifying the element as we specify the partial path not (exact path). If there are multiple elements for the same path, it will select the first element that is identified.

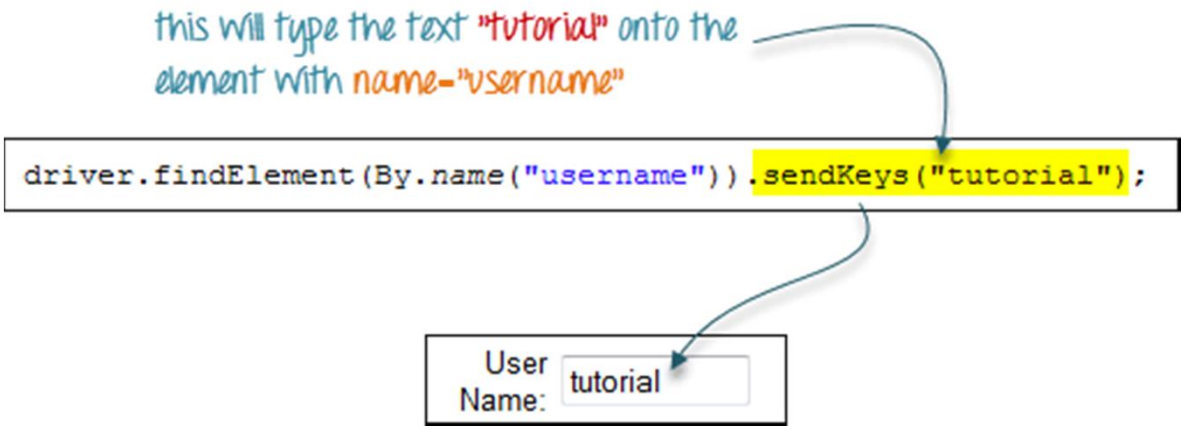
Web Driver : Locator Tree of Life

"When writing your test automation scripts it's good practice to go after the low lying fruit rather than reach for those at the top of the canopy."

The tastiest locators are the low lying fruit on the *locator* tree of life.

Web Driver : Accessing Form Elements

Entering Values in Input Boxes



The **sendKeys()** method is used to enter values into input boxes.

Deleting Values in Input Boxes

```
driver.findElement(By.name("userName")).clear();
```

The **clear()** method is used to delete the text in an input box.

This method does not need any parameter.

Web Driver : Accessing Form Elements

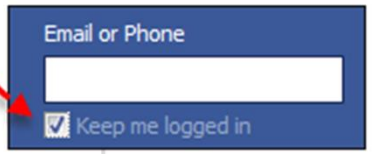
Radio Button

```
driver.findElement(By.cssSelector("input[value='Business']")).click();
```

Check Box

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    String baseURL = "http://www.facebook.com";

    driver.get(baseURL);
    WebElement chkFBPersist = driver.findElement(By.id("persist_box"));
    for(int i=0; i<2; i++){
        chkFBPersist.click();
        System.out.println(chkFBPersist.isSelected());
    }
    driver.quit();
}
```



Web Driver : Accessing Form Elements

Å The table below summarizes the commands to access each type of element.

Element	Command	Description
Input Box	*sendKeys()*	used to enter values onto text boxes
	clear()	used to clear text boxes of its current value
Check Box, **Radio Button,**	*click()*	used to toggle the element on/off
Links	*click()*	used to click on the link and wait for page load to complete before proceeding to the next command.

Web Driver : Accessing Form Elements Cont..

Drop-Down Box	*selectByVisibleText()/ deselectByVisibleText()*	selects/deselects an option by its displayed text
	selectByValue()/ deselectByValue()	selects/deselects an option by the value of its "value" attribute
	selectByIndex()/ deselectByIndex()	selects/deselects an option by its index
	isMultiple()	returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise
	deselectAll()	deselects all previously selected options
Submit Button	*submit()*	

Web Driver : Handling Windows

- There are many cases, where a application displays multiple windows when you open a website.
- Those are may be advertisements or may be a kind of information showing on popup windows.
- User can handle multiple windows using *Windows Handlers in selenium webdriver*.
- Selenium WebDriver assigns an *alphanumeric id to each window* as soon as the WebDriver object is instantiated.
- *This unique alphanumeric id is called window handle*.
- Selenium uses this unique id to switch control among several windows.
- *In simple terms, each unique window has a unique ID, so that Selenium can differentiate when it is switching controls from one window to the other*.

Web Driver : Handling Windows Steps

Step 1: After opening the website, we need to get the main window handle by using

driver.getWindowHandle();

The window handle will be in a form of lengthy alpha numeric.

Step 2: We now need to get all the window handles by using *driver.getWindowHandles();*

Step 3: We will compare all the window handles with the main Window handles and perform the operation the window which we need.

GetWindowHandle Command :

Purpose: To get the window handle of the current window.

```
String  handle= driver.getWindowHandle();//Return a string of alphanumeric window handle
```

GetWindowHandles Command :

Purpose: To get the window handle of all the current windows.

```
Set<String> handle= driver.getWindowHandles();//Return a set of window handle
```

SwitchTo Window Command :

Purpose: WebDriver supports moving between named windows using the ñswitchToò method.

```
driver.switchTo().window("windowName");
```

Web Driver : Handling Windows Steps

- Alternatively, you can pass a "window handle" to the "*switchTo().window()*" method.
- Knowing this, it's possible to iterate over every open window like so:

```
for (String handle : driver.getWindowHandles()) {
  driver.switchTo().window(handle); }
```

- Switching between windows with **Iterators**:

```
driver.findElement(By.id("id of the link which opens new window")).click();

//wait till two windows are not opened
waitForNumberofWindowsToEqual(2);//this method is for wait

Set handles = driver.getWindowHandles();
firstWinHandle = driver.getWindowHandle(); handles.remove(firstWinHandle);
String winHandle=handles.iterator().next();

if (winHandle!=firstWinHandle){
//To retrieve the handle of second window, extracting the handle which does not match to first
//window handle
secondWinHandle=winHandle; //Storing handle of second window handle

//Switch control to new window
driver.switchTo().window(secondWinHandle);
```

Web Driver : Handling Frames

- A web page which is embedded in another web page or an HTML document embedded inside another HTML document is known as a frame.
- The **IFrame** is often used to insert content from another source, such as an advertisement, into a Web page. *The <iframe> tag specifies an inline frame*.
- User cannot detect the frames by just seeing the page or by inspecting Firebug.
- We can identify the iframes using methods given below:
  - Right click on the element, If you find the option like *'This Frame*' then it is an iframe.
  - Right click on the page and click *'View Page Source*' and Search with the '*iframe*', if can find any tag name with the 'iframe' then it is meaning to say the page consisting an iframe.
- Can even identify total number of iframes by using below snippet :

```
int size = driver.findElements(By.tagName("iframe")).size();
```

Web Driver : Handling Frames contd..

- Basically, can switch over the elements in frames using 3 ways.
  - **By Index**
  - **By Name or Id**
  - **By Web Element**

Switch to the frame by index:

- Index is one of the attributes for the Iframe through which we can switch to it.
- Index of the iframe starts with '**0**'.

```
driver.switchTo().frame(int arg0);
```

Switch to the frame by Name or ID:

- Name and ID are attributes of iframe through which we can switch to the it.

```
driver.switchTo().frame(String arg0);
```

Switch to the frame by Web Element:

- We can even switch to the iframe using web element .

```
driver.switchTo().frame(WebElement);
```

Firefox Profile

- **Setting Firefox Preferences:**

```
FirefoxProfile  profile = new FirefoxProfile();
profile.setPreference("browser.startup.homepage",
                                        "http://www.google.co.in");
driver = new FirefoxDriver(profile);
```

- **Installing the Add-on:**

```
profile.addExtension("firebug.xpi");
profile.setPreference("extensions.firebug.currentVersion", "1.11.4");
```

Working with InternetExplorerDriver

- Download IEDriverServer and set the environment:
- PATH=$PATH ; \\path-to-IEdriver.

WebDriver driver = new InternetExplorerDriver();

Driver.get("http://www.google.co.in");

Synchronization in Selenium Web Driver

- It is a mechanism which involves more than one components to work parallel with Each other.

- Generally in Test Automation, we have two components

  1. Application Under Test

  2. Test Automation Tool.

- Both these components will have their own speed.

- We should write our scripts in such a way that both the components should move with same and desired speed, so that we will not encounter "*Element Not Found*" errors which will consume time again in debugging.

Categories of Synchronization

- Synchronization can be classified into two categories:

  1. Unconditional Synchronization

  2. Conditional Synchronization

Unconditional Synchronization

- In this we just specify timeout value only. We will make the tool to wait until certain amount of time and then proceed further.
  Examples: ***Wait() and Thread.Sleep();***
- The main disadvantage for the above statements are, there is a chance of unnecessary waiting time even though the application is ready.
- The advantages are like in a situation where we interact for third party systems like interfaces, it is not possible to write a condition or check for a condition.
- Here in this situations, we have to make the application to wait for certain amount of time by specifying the timeout value.

Conditional Synchronization

- It is very important to set the timeout value in conditional synchronization, because the tool should proceed further instead of making the tool to wait for particular condition to satisfy.

- In Selenium we have ***implicit Wait*** and ***Explicit Wait conditional statements.***

Implicit Wait

- An *implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available*.
- The default setting is 0. Once when we define the implicit wait, it will set for the life of the WebDriver object instance.
- It is a mechanism which will be written once and applied for entire session automatically. It should be applied immediately once we initiate the Webdriver.
- Implicit wait will not work all the commands/statements in the application.
- *It will work only for "FindElement" and "FindElements" statements*.

Implicit Wait : Implementation

- If we set implicit wait, find element will not throw an exception if the element is not found in first instance, instead it will poll for the element until the timeout and then proceeds further.

- Should always remember to add the below syntax immediately below the Webdriver statement.

Syntax: ***driver.manage.TimeOuts.implicitwait(6,Timeunit.SECONDS);***

Example:

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("www.google.com");
```

Explicit Wait

- We need to define a wait statement for certain condition to be satisfied until the specified timeout period.
- If the Webdriver finds the element within the timeout period the code will get executed.
- *Explicit wait is mostly used when we need to Wait for a specific content/attribute change after performing any action, like when application gives AJAX call to system and get dynamic data and render on UI.*

Example:

- Like there are drop-downs Country and State, based on the country value selected, the values in the state drop-down will change, which will take few seconds of time to get the data based on user selection

```
WebDriverWait wait = new WebDriverWait(driver, 10);
  wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("statedropdown")));
```

Fluent Wait

- Using *FluentWait* we can define the maximum amount of time to wait for a condition, as well as the frequency with which to check for the condition.

- And also the user can configure to ignore specific types of exceptions such as "NoSuchElementExceptions" when searching for an element.

- Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
  //Wait for the condition
  .withTimeout(30, TimeUnit.SECONDS)
  // which to check for the condition with interval of 5 seconds.
  .pollingEvery(5, TimeUnit.SECONDS)
  //Which will ignore the NoSuchElementException
  .ignoring(NoSuchElementException.class);

Solutions

- An element not being present at all in the DOM.
- An element being present in the DOM but not visible.
- An element being present in the DOM but not enabled. (i.e. clickable)
- There are pages which get displayed with the **JavaScript**, the elements are already present in the browser **DOM**, but are not visible.
- The implicit wait only waits for an element to appear in the DOM, so it returns immediately, but when you try to interact with the element you get a **NoSuchElementException**. You could test this hypothesis by writing a helper method that explicit wait for an element to be visible or clickable.

- WebDriver is a tool for testing web applications across different browsers using different programming languages.
- You are now able to make powerful tests because WebDriver allows you to use a programming language of your choice in designing your tests.
- WebDriver is faster than Selenium RC because of its simpler architecture
- WebDriver directly talks to the browser while Selenium RC needs the help of the RC Server in order to do so.
- WebDriver's API is more concise than Selenium RC's.
- WebDriver can support HtmlUnit while Selenium RC cannot.
- The only drawbacks of WebDriver are:
  - It cannot readily support new browsers, but Selenium RC can.
  - It does not have a built-in command for automatic generation of test results.

Thank you

Saradhi.Seshagiri@TechMahindra.com

Disclaimer

Tech Mahindra Limited, hereinafter referred to as TechM provide a wide array of presentations, and reports with the contributions of various professionals. These presentations and reports are for information purposes and private circulation only and do not solicit any offer to buy or sell any securities mentioned therein. They do not purport to be a complete description of the markets conditions or developments referred to in the material. While utmost care has been taken in preparing the above, we shall not be responsible for its accuracy. We shall not be liable for any direct or indirect losses arising from the use thereof and the viewers are requested to use the information contained herein at their own risk. These presentations and reports should not be reproduced, re-circulated, published in any media, website or otherwise, in any form or manner, in part or as a whole, without the express consent in writing of TechM or its subsidiaries. Any unauthorized use, disclosure or publication of this information contained herein is prohibited. Unless specifically noted, TechM is not responsible for the content of these presentations, and the opinions of the presenters. Individual situations and local practices and standards may vary, so viewers and others utilizing information contained within a presentation are free to adopt differing standards and approaches as they see fit. You may not repackage or sell the presentation. Products and names mentioned in materials or presentations are the property of their respective owners and the mention of them does not constitute an endorsement by TechM. Information contained in a presentation hosted or promoted by TechM is provided "as is" without warranty of any kind, either expressed or implied, including any warranty of merchantability or fitness for a particular purpose. TechM assumes no liability or responsibility for the contents of a presentation or the opinions expressed by the presenters. All expressions of opinion are subject to change without notice.

Version History

Version History				
Version No	**Date**	**Created/ Changed by**	**Changes made**	**Reviewed by**
1	24-Jan-16	Saradhi Seshagiri	- Conversion to 2016 template - Adding the new contents	Prakash Goteti

Tech Mahindra