Extract PDF text And Verify Text Present in PDF using WebDriver

Most of the applications has 'Print PDF' functionality. How to achieve this in Automation. we first need to decide is this really required to automate, if your answer is Yes then proceed further to see how we can achieve this. In Earlier tutorial we have seen validating if the file downloaded or not after clicking on download button. In this tutorial we will now see to validate Print PDF functionality by using below two ways.

There are multiple ways of doing this.

1. A very simple way without using any third party libraries.
2. Extract the text from PDF and then validate if the text you are looking is present in the PDF document or not. We should go for this ONLY when we want to validate something for sure.

Based on the requirement can decide on which one to use.

The very first way of doing this is below:

```
/**
         * To verify pdf in the URL
         */
        @Test
        public void testVerifyPDFInURL() {
                WebDriver driver = new FirefoxDriver();
                driver.get("http://www.princexml.com/samples/");
                driver.findElement(By.linkText("PDF flyer")).click();
                String getURL = driver.getCurrentUrl();
                Assert.assertTrue(getURL.contains(".pdf"));
        }
```

The second way is using third party library. In this example we will how to use 'Apache PDFBox' library

To extract text from a PDF we can use Apache PDFBox library which is one of the main feature of PDFBox. I can extract the text from variety of PDF documents. The functionality of extracting text is encapsulated in 'org.apache.pdfbox.util.PDFTextStripper'

It also provides an option to limit the text that is extracted during the extraction process by specifying the range of pages that we want to extract. For example, if the PDF has 100 pages, we can give the range from first to second page to validate the text present.

Below code snippet to specify the range which will read first and second page of the PDF. If you want to verify the text some where in the middle of the PDF you can read that and validate.

```
PDFTextStripper stripper = new PDFTextStripper();
pdfStripper.setStartPage(1);
pdfStripper.setEndPage(2);
```

NOTE: The startPage and endPage properties of PDFTextStripper are 1 based and inclusive.

Below is the example Program for the both the above discussed ways.

```
import java.io.BufferedInputStream;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;

import junit.framework.Assert;

import org.apache.pdfbox.cos.COSDocument;
import org.apache.pdfbox.pdfparser.PDFParser;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.util.PDFTextStripper;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class ReadPDF {

        WebDriver driver;

        @BeforeClass
        public void setUp() {
                driver = new FirefoxDriver();
        }

        /**
         * To verify PDF content in the pdf document
         */
        @Test
        public void testVerifyPDFTextInBrowser() {

                driver.get("http://www.princexml.com/samples/");
                driver.findElement(By.linkText("PDF flyer")).click();
```

```
                Assert.assertTrue(verifyPDFContent(driver.getCurrentUrl(), "Prince Ca
scading"));
        }

        /**
         * To verify pdf in the URL
         */
        @Test
        public void testVerifyPDFInURL() {
                driver.get("http://www.princexml.com/samples/");
                driver.findElement(By.linkText("PDF flyer")).click();
                String getURL = driver.getCurrentUrl();
                Assert.assertTrue(getURL.contains(".pdf"));
        }


        public boolean verifyPDFContent(String strURL, String reqTextInPDF) {

                boolean flag = false;

                PDFTextStripper pdfStripper = null;
                PDDocument pdDoc = null;
                COSDocument cosDoc = null;
                String parsedText = null;

                try {
                        URL url = new URL(strURL);
                        BufferedInputStream file = new BufferedInputStream(url.openSt
ream());
                        PDFParser parser = new PDFParser(file);

                        parser.parse();
                        cosDoc = parser.getDocument();
                        pdfStripper = new PDFTextStripper();
                        pdfStripper.setStartPage(1);
                        pdfStripper.setEndPage(1);

                        pdDoc = new PDDocument(cosDoc);
                        parsedText = pdfStripper.getText(pdDoc);
                } catch (MalformedURLException e2) {
                        System.err.println("URL string could not be parsed "+e2.getMe
ssage());
                } catch (IOException e) {
                        System.err.println("Unable to open PDF Parser. " + e.getMessa
ge());
                        try {
                                if (cosDoc != null)
                                        cosDoc.close();
                                if (pdDoc != null)
                                        pdDoc.close();
                        } catch (Exception e1) {
                                e.printStackTrace();
                        }
```

```
                }

                System.out.println("+++++++++++++++++");
                System.out.println(parsedText);
                System.out.println("+++++++++++++++++");

                if(parsedText.contains(reqTextInPDF)) {
                        flag=true;
                }

                return flag;
        }

        @AfterClass
        public void tearDown() {
                driver.quit();
        }
}
```

The above case works fine when the PDF file is opened in a Browser after clicking on the Print button. There are few cases where once we click on Print, it will download the pdf file.

In these cases we should do in the below way: We need to change the below code

```
URL url = new URL(strURL);
BufferedInputStream file = new BufferedInputStream(url.openStream());
PDFParser parser = new PDFParser(file);

convert as below
                File file = new File("D:/Paynetsbicardbill.pdf");
                PDFParser parser = new PDFParser(new FileInputStream(file));
```

We should pass the path where the document is downloaded.

‹ Element is not clickable at point SeleniumWebdriverException Optional annotation in TestNG ›

Selenium Tutorials:

Selenium Tutorials

Add new comment

Comments

good to reading

Permalink Submitted by Viewer on Mon, 04/20/2015 - 15:55

good to reading

reply

Selenium Tutorials

- Introduction to selenium
- Basic HTML concepts required for Selenium
- Locators for Selenium
- How to Run Webdriver in chrome browser?
- How to run webdriver in IE browser?
- Selenium webdriver using Eclipse
- Synchronization in Selenium Webdriver
- XPath tutorial for Selenium
- CSS selectors for Selenium with example
- Firefox Not Connected Exception in Selenium
- Handle windows popups using Selenium Webdriver
- Working with Frames in Selenium Webdriver
- How to handle javascript alerts, confirmation and prompts?
- Defining Firefox Profile preferences using Selenium Webdriver
- Why Automation Testing is required?
- Importance of Automation Testing
- Mouse Hover Actions in Selenium Webdriver
- StaleElementReferenceExceptions in Selenium Webdriver
- Difference between Webdriver get() and Webdriver navigate()
- Navigation Methods in Webdriver with Examples
- Handling Cookies in Webdriver
- How to addcookie with Selenium Webdriver
- How to delete Cookies in Selenium Webdriver
- WebDriver Methods
- Automation Test Frameworks
- Taking Screenshot using Webdriver
- Webdriver SELECT Methods to work with Dropdowns
- Webdriver Select with Multiple Attribute
- Handling keyboard events and mouse hover events using Webdriver
- Set browser width and height in Selenium Webdriver
- Selenium Automation Framework Example
- UI Map or Object Repository using Properties File
- Test Data in automation framework
- Reporting Structure in Selenium Automation Framework
- Test Configuration File / Executable File
- Uploading a file with Selenium Webdriver

- Upload file using AutoIT
- Uploading a file with sendKeys method
- Read data from Properties file using Java Selenium
- Scrolling web page with Selenium Webdriver using java
- Keyword Driven Framework Example
- Install AutoIt in Windows
- Examples with xpath and Css (ID, Name, Text and Links)
- Testing in multiple browsers using selenium and testng
- Working with Checkbox using Resuable Methods
- Working with Select examples
- File upload using Robots
- Download file using selenium webdriver
- Perform operations on new window
- Introduction to Page Object Model Framework
- Mouse hover using javaScript Executor
- Working with Multiple Checkboxes with Safecheck
- Click element using JavaScriptExecutor
- Drag and Drop using Webdriver Action Class
- WebDriver Waits Examples
- Simple Page Object Model example
- Double Click on element using Webdriver
- Right Click Context Menu example
- Working with AutoComplete Text box
- Working with Date Picker Calendar
- Working with Ajax controls using Webdriver
- Get Attribute values using Webdriver
- Get Css values using Webdriver
- Resizing a web element using movebyoffset
- Validate Downloaded file after clicking on downloaded button/ link
- How to verify entries in Exported CSV file
- Element is not clickable at point SeleniumWebdriverException
- Extract PDF text And Verify Text Present in PDF using WebDriver
- Handling Authentication Window with WebDriver (In Firefox, Chrome and IE)

Recent Post

Handling Authentication Window with WebDriver (In Firefox, Chrome and IE)

Automation Framework

- Framework Structure
- Object Repository

Selenium Webdriver Tutorials

Locators for Selenium

Optional annotation in TestNG

Parameterization in TestNG using testng.xml

- Test Data
- Test Scripts
- Reports
- Configuration and Execution

Tags Cloud

Asserts (1) AutoComplete (1) AutoIt (2) Automation (3) Build.xml (1) Checkboxes (2) Cookies (3) Css (2) Css Examples (2) DataProvider (1) Date Picker (1) Drag and Drop (1) Exceptions (4) File Download (3) Firefox Profile (2) Framework Frameworks (5) iFrames (1) Interview Questions (1) JavascriptExecutor (3) JUnit (2) Keyword Driven (1) Listeners (3) Manual Testing (1) Maven (5) Mouse Hover (2) Page Object Model (2) Page Scroll (1) Parallel Test (3) properties file (1)

- Advanced Webdriver Tutorials
- Selenium Framework Examples
- JavascriptExecutor Examples

Interview Questions

- Testing Concepts
- Quality Standards
- General Questions
- Roles & Responsibilities
- Technology & Environment

We welcome your questions and requests