

Selenium Starter Kit

Extending Base Framework

Developer's Guide



Extending Base Framework

Developer's Guide

Introduction

This document describes guidelines for extending Pega base framework shipped with the Selenium Starter Kit to support testing custom Pega applications. A sample test framework project attached below is used to guide through the extension process. Save this project and import it into your IDE.



SampleTestFramework.zip

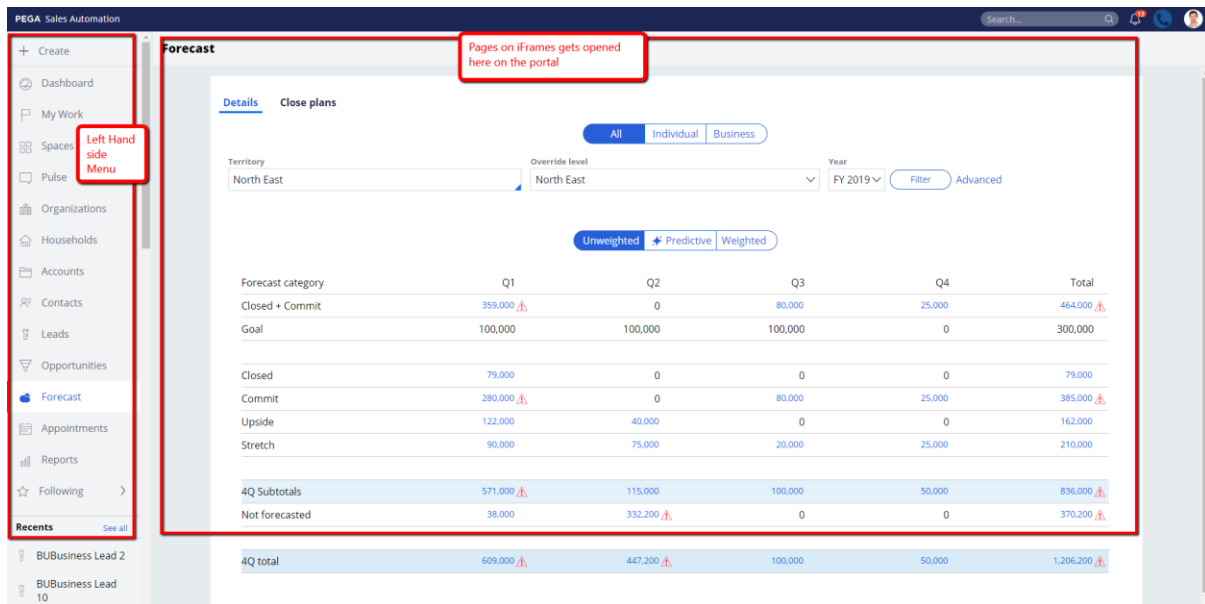
To install the sample project, follow the Selenium Starter Kit - Setup Guide.pdf sections Setting Up Environment → System Environment and Setting Up Environment → Eclipse. Once done, follow Setting Up Tests section to import the sample test framework project after extracting the above zip file. Please note that the project shown in the Setup Guide is different to the one mentioned here.

The base UI testing framework in the Selenium Starter Kit is a framework built on Selenium WebDriver. It is developed to make automation of Pega applications easier. It is achieved by creating wrappers around Selenium WebDriver classes like WebDriver, WebElement etc. Apart from wrapping Selenium classes there are lots of classes added like TopDocument, Frame, WaitHandler, BaseTestCase etc which would be introduced in the document wherever needed.

The framework is a Cucumber-based and employs the Page Object Model to ensure minimal maintenance due to UI changes in the application. Let's go through the workflow on how to extend the framework and start automating a test

Sample Test Framework and Application Used:

For the sample framework design, the application used is Sales Automation. When logged in to the application, a sales manager portal would get displayed which has different links to open in its left side navigation menu like Forecast, Spaces, Accounts, Organizations etc. When clicked on any link the respective page would be opened in a new frame in the portal, which we call it as a Frame page

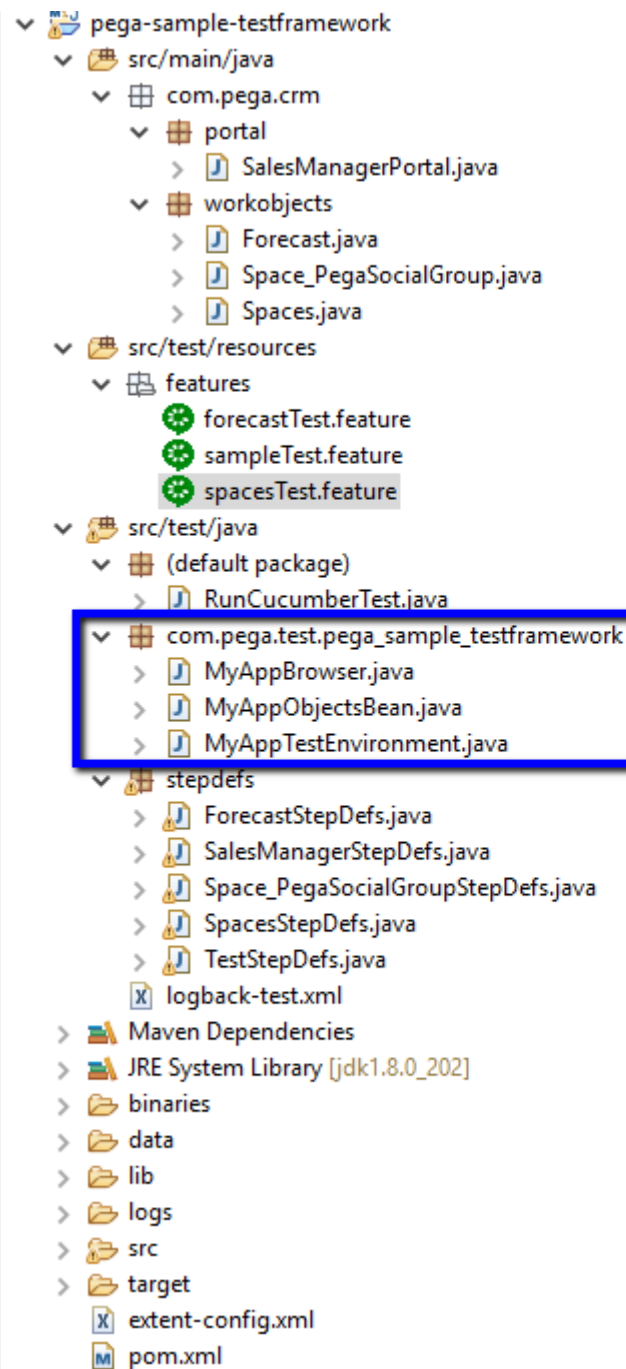


Design Model

The UI testing framework shipped with this Sample Framework is fundamentally built using the [Page Object Design Pattern](#). Implementing that pattern results in a library of page objects corresponding to various pages of the web application being tests. These page objects provide the interface to interact with UI elements that belong to that page.

For the user to be able to launch a browser, the framework provides TestBase class out of the box. Any class which extends TestBase class will be the starting point while a user starts running his test. The sample framework provides a sample class named MyAppTestEnvironment to do this job. Alongside creating a class which extends Testbase, use should also create another class which extends BrowserImpl which comes out of the box from the framework. This browser class provides out of the box methods for login and logoff operations which works for most of the pega applications. Apart from login/logoff we also provide options here like refresh, executeJavaScript, open a url, close, switchToWindow etc.

Anyone who wants to use base framework, for them we recommend writing their own MyAppBrowser class and MyAppTestEnvironment class. These classes are available for your reference at src/main/java/com.pega.test.pega_sample_testframework package as mentioned in below figure. User must Change MyApp to its application name while naming the classes.



The MyApp in these files to be replaced with name specific to user's application

MyAppTestEnvironment.java

The MyAppTestEnvironment class must extend the TestBase class which internally extends from TestEnvironmentImpl class. TestBase and TestEnvironmentImpl classes are defined in the base framework.

Override the getBrowser method to return a handle to the browser instance specific to the Application. In this project the browser class specific to this application is MyAppBrowser.

```

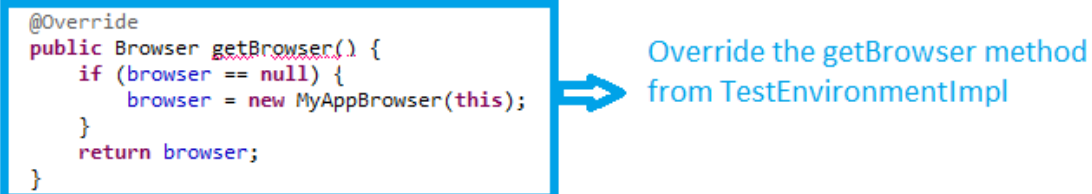
@ScenarioScoped
public class MyAppTestEnvironment extends TestBase {

    String COPYRIGHT = "Copyright (c) 2014 Pegasystems Inc.";
    String VERSION = "$Id: MyTestEnvironment.java 209030 2016-09-22 06:52:49Z SachinVellanki $";

    private Browser browser;
    private Scenario scenario;

    @Override
    public Browser getBrowser() {
        if (browser == null) {
            browser = new MyAppBrowser(this);
        }
        return browser;
    }
}

```



Override the getBrowser method from TestEnvironmentImpl

This class when instantiated launches the browser based on the properties defined in **global-settings.properties** file.

Note: User must replace App with the actual application name for MyAppTestEnvironment class.

global-settings.properties:

Global settings and test requirements are defined in /data/global-settings.properties. Changing these settings will allow to customize test execution. The following tables show the list of properties:

Application Information:

Property	Description
instance.url	URL of the application under test

Browser Configuration:

Property	Description
browser.name	Name of the browser used for testing. Supported browsers: <ul style="list-style-type: none">• chrome• firefox• ie• safari• htmlunit
chrome.driver ie.driver edge.driver chrome.driver.linux	Path to the appropriate browser binaries/driver
isChromeAutoDownload	By default, we attempt to download an appropriate chrome driver automatically through our custom utility. If it fails, set this property to false and copy the driver manually to binaries folder

Diagnostics & Debug Settings:

Property	Description
debug.mode	Boolean indicating whether to keep the browser open after test execution
enable.fullscreen.mode	Boolean indicating whether tests run in full screen mode
global.timeout	Override maximum wait time for the web elements to load (secs). Default timeout is 300 seconds.

Test Environment Configuration:

Property	Description
hub.url	URL to selenium grid hub for Cross Browser Testing. If this is not set, tests run locally
capabilities	Any custom capabilities provided by the external selenium grid providers like crossbrowsertesting / saucelabs / browserstock. Multiple capabilities can be provided by separating them with , and : capabilities=capability1:value1, capability2:value2, capability3:value3,

If there are some common steps that every test case must execute before and after executing the test steps, then we can use **@Before** and **@After** cucumber annotations. Writing these methods are optional. These methods are provided to give the user the capability to add their own steps in setup and teardown methods.

```
@Before
public void setUp(Scenario scenario) {
    this.scenario = scenario; //this object of scenario is to send to localizationUtil to take screenshot of every step failure
    System.setProperty("is.one.step.one.def", "true");
    setUp(scenario, null);
}
```

```
protected void setUp(Scenario scenario, String browserName) {
    initializeStatus();
    startRecording(scenario);
    configureBrowser();
}
```

common steps to be executed before each test case can be written into the setUp method

```
@After
public void tearDown(Scenario scenario) {
    tearDown(scenario, true, alwaysSaveVideo);
}
```

Common steps to be executed after each test case can be written in tearDown method

```
public Scenario getScenario()
{
    return scenario;
}
```

MyAppBrowser.java

This class defines the browser for a specific application. The MyAppBrowser class must extend BrowserImpl class which is a base framework class. MyAppBrowser class has methods to do operation on a browser like login, logout etc.

```
public class MyAppBrowser extends BrowserImpl {
```

This class must override **getPortal** method from BrowserImpl class which returns the object of the Landing page after Login in to the application. Portal is a page which is the first page after logging into the application.

For Example, if a user logs into the PRPC application, after logging in the user is landed into the DesignerStudio page. Thus, we can call this page as a Portal type page as it is the first page after logging in. Similarly, we can have different portals like SalesManagerPortal, CaseManagerPortal e.t.c.

One application can have multiple portals and the getPortal method should be the method which should return the object for any portal class created during framework design.

```
public <T extends Portal> T getPortal(Class<T> type) {
    T portal = null;
    String className = type.getName();
    if (className.contains("SalesManagerPortal")) {
        portal = type.cast(new SalesManagerPortal(testEnv));
    }
    if(className.equals("SalesRep"))
    {
        portal = type.cast(new SalesRepPortal(testEnv));
    }
    return portal;
}
```

The return type of the getPortal method is a Portal. All the portals that are defined as page objects are available at src/main/java/ com.pegacrm.portal. In the sample application we chose if else conditions to differentiate between different portals to return handles for the respective portal. People who extends this methods can choose any different way to return their portals

All the other methods except getPortal method are optional in the class. The list of all the methods can be seen at the end of this document.

Default step definitions in MyAppBrowser class:

The class provides some methods defined out of the box to support basic operations such as logging in and out of the application.

```
@Given("^A User logs in with \"(.*)\" and \"(.*)\"$")
```

This step definition is used to login to the application with a specific user id and password mentioned in the cucumber feature file

Example:

```
@Given("^A User logs in with \"(.*)\" and \"(.*)\"$")
public void login(String username, String password) {
    open();
    super.login(username, password);
}
```

Here the login method first calls the **open** method which will open the browser. Then the default **login** method from the base framework BrowserImpl class is called which will do the logging operation using the provided username and password. We use the default login from framework since in Pega, all login is same across all Pega applications.

1. @When("^User logs off from portal\$")

This step definition is used to perform the logout operation

Example:

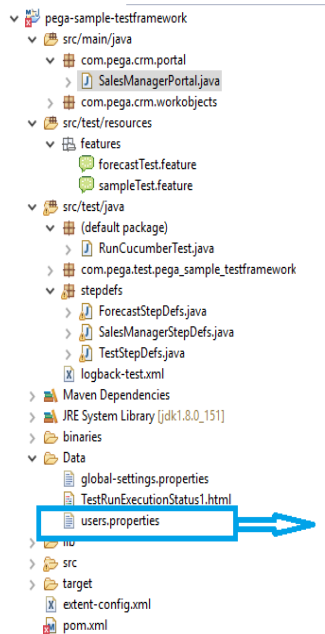
```
@When("^User logs off from portal$")
public void user_logs_off_from_portal() {
    super.logout();
}
```

Here the user_logs_off_from_portal method calls the **logout** method from the BrowserImpl class which will logout from the application

2. @Given("^A User logs in with Administrator credentials\$")

This step definition is used to do login as an administrator. It takes the admin credentials mentioned in **users.properties** file. This way user can mask the credentials.

Instead of exposing the credentials in the feature file we can hide it and ask the step definition to take the data from **users.properties** file.



User must create a file called users.properties to store the login credentials of the application

```
1 #Administrator credentials
2 ADMIN_USER_ID =AutoMarketingAdministrator
3 ADMIN_PASSWORD =install123!
4
5 #Analyst credentials
6 ANALYST_USER_ID =AutoMarketingAnalyst
7 #ANALYST_USER_ID =morimAnalyst
8 ANALYST_PASSWORD =install123!
```

Credentials to be stored in users.properties

3.

```
@When("^User logs off from portal$")
public void user_logs_off_from_portal() {
    super.logout();
}
```

Here the user_logs_off_from_portal method calls the **logout** method from the BrowserImpl class which will logout from the application

In case the login and logout methods of the application are different, although unlikely, user can override the framework login and logout methods in MyAppBrowser class.

In case, if the login method is overridden, the following lines of code should be added at the end of login method, which would be used by base framework for synchronization between scripts and the application under test. Lots of our internal waits depends on these methods.

```
pegaDriver.getDefaultFrameTabCntDiff(true);
pegaDriver.loadCustomScripts();
ObjectBean.setLoggedInUser(usr);
```

```

@Override
public <T extends Portal> T getPortal(Class<T> type) {
    T portal = null;
    String className = type.getName();
    if (className.contains("SalesManagerPortal")) {
        portal = type.cast(new SalesManagerPortal(testEnv));
    }
    if (className.equals("SalesRep")) {
        //portal = type.cast(new SalesRepPortal(testEnv));
    }
    return portal;
}

@Override
public void logout() {
    pegaDriver.waitForDocStateReady(2);
    pegaDriver.switchTo().defaultContent();
    pegaDriver.findElement(OPERATOR_MENU).click();
    pegaDriver.switchTo().defaultContent();
    pegaDriver.findElement(LOG_OFF_BUTTON).click();
}

```

logout method specific to the application

The instance of MyAppTestEnvironment is injected into the constructor of MyAppBrowser. It can be used to get handles to other relevant objects such as PegaWebDriver, Configuration, Browser. The methods and description for these classes are provided towards the end of this document

```

@Inject
public MyAppBrowser(MyAppTestEnvironment testEnv) {
    super(testEnv);
    this.testEnv = testEnv;
    configuration = testEnv.getConfiguration();
}

```

Page Objects using the Base UI Test Framework:

As mentioned earlier in this document, the framework is fundamentally built using the [Page Object Design Pattern](#). We identified 3 type of pages in any pega application which are as follows

1. Portal Pages

These are the landing pages right after the user logs in to the application, i.e the first page of any application, from where all the navigations take place

2. Frame Pages

These are the pages that gets opened in an IFrame

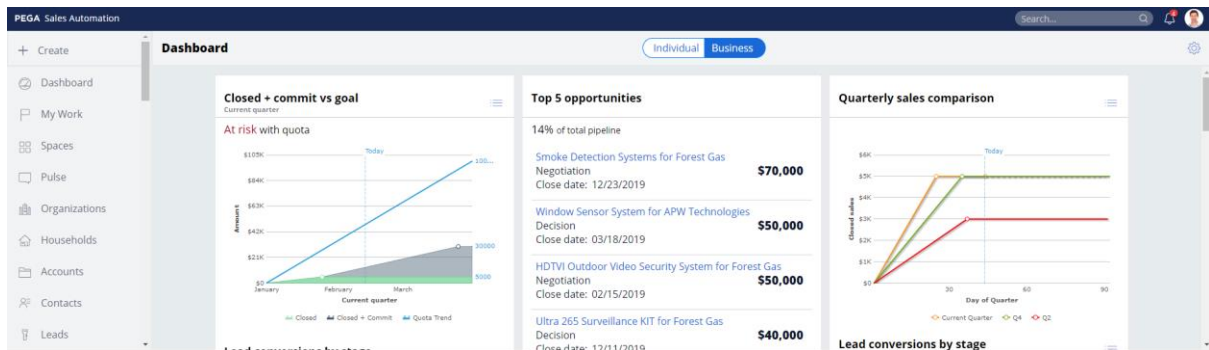
3. TopDocument Pages

These are the pages that are not under IFrames

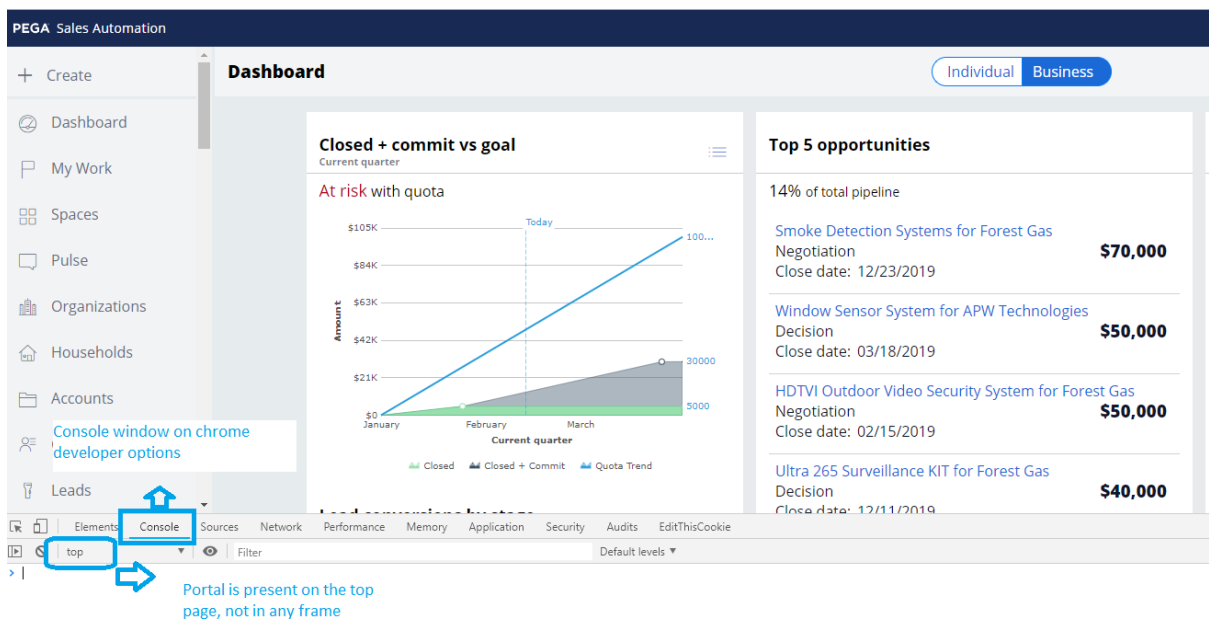
Portal Pages in Application:

When the user logs into the application, user is redirected to a landing page. For example, when user logs into the Sales Automation application using manager credentials, user is redirected into a landing page. This is considered as a portal page, say, SalesManagerPortal page. Usually the first landing page after the Login is considered as portal.

The below figure represents the SalesManagerPortal.



In any Pega application, all the portals are present on the top page, but not inside any frame. Use Chrome Developer's options and navigate to Console tab to see whether the page belongs to the top page or the frame.



It is highly unlikely that the portal will be present inside a frame.

All the portal page objects can be organized in `src/main/java/com.pegacrm.portal` package. A portal page must extend **PortalImpl** class from base framework (Internally Portal also extends TopDocument as all portals are present under top document only)

```
public class SalesManagerPortal extends PortalImpl
```

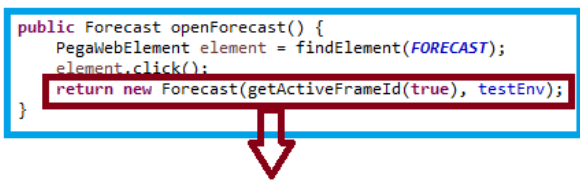
Every portal class must have a public constructor that accepts `TestEnvironment` object where the constructor calls the super class constructor.

```
public SalesManagerPortal(TestEnvironment testEnv) {
    super(testEnv);
}
```

In the `SalesManagerPortal` page there is a menu called **Forecast**, which, when clicked, opens the Forecast page. To perform this operation the `SalesManagerPortal` page has **openForecast** method, which creates the object of forecast page and returns it.

While creating the PageObject of Forecast, pass the active frame id of the Page and MyAppTestEnvironment object. Any Page Object which extends a frame would need a frameId and testenvironment object as its parameters in the constructor.

```
public class SalesManagerPortal extends PortalImpl{  
    public static final By FORECAST = By.xpath("//span[text()='Forecast']");  
    MyAppTestEnvironment testEnv;  
    public SalesManagerPortal(MyAppTestEnvironment testEnv) {  
        super(testEnv);  
        this.testEnv=testEnv;  
    }  
    public Forecast openForecast() {  
        PegaWebElement element = findElement(FORECAST);  
        element.click();  
        return new Forecast(getActiveFrameId(true), testEnv);  
    }  
}
```



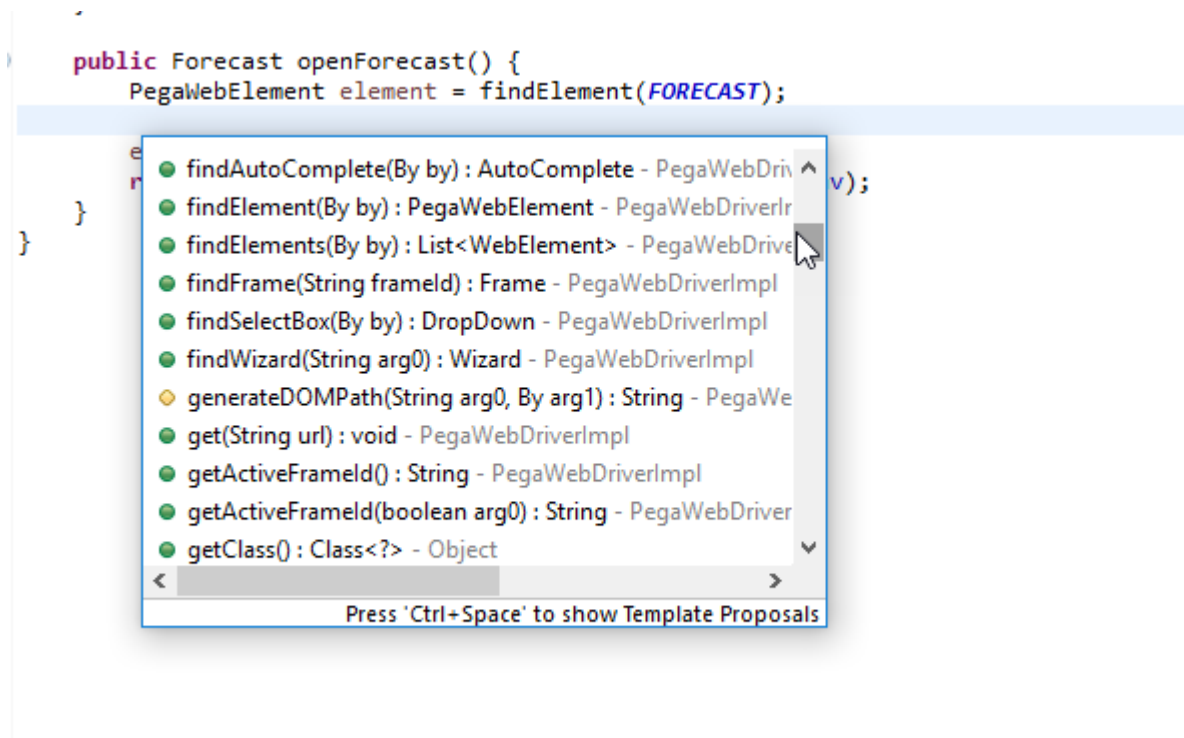
The diagram highlights the `openForecast()` method in the `SalesManagerPortal` class. A blue box surrounds the entire method, with a blue arrow pointing to the text "openForecast method to open the Forecast page from SalesManagerPortal". A red box highlights the `return new Forecast(getActiveFrameId(true), testEnv);` line, with a red arrow pointing down to the text "Passing frame id of forecast page and MyAppTestEnvironment obj while creating the object".

While trying to identify the locator for forecast, xpath was used, (By.xpath), as it was the best was the best option available to identify that locator. But the order of precedence would be By.id and By.name in case the options are available and then use By.cssSelector or By.xpath as the last options. In case of pega applications, you can also use data-test-ids if enabled for a locator. An example for using data test id is provided in Spaces page object in the sample application.

getActiveFrameId(true) is a base framework method which will switch to the current active frame and then return the frame id. If the user passes false as input to `getActiveFrameId` method, then the control will not switch to the active frame id and will return the old frame id.

The forecast page is present in a frame. Hence the user must switch to the active frame and then pass the active frame id while constructing the object.

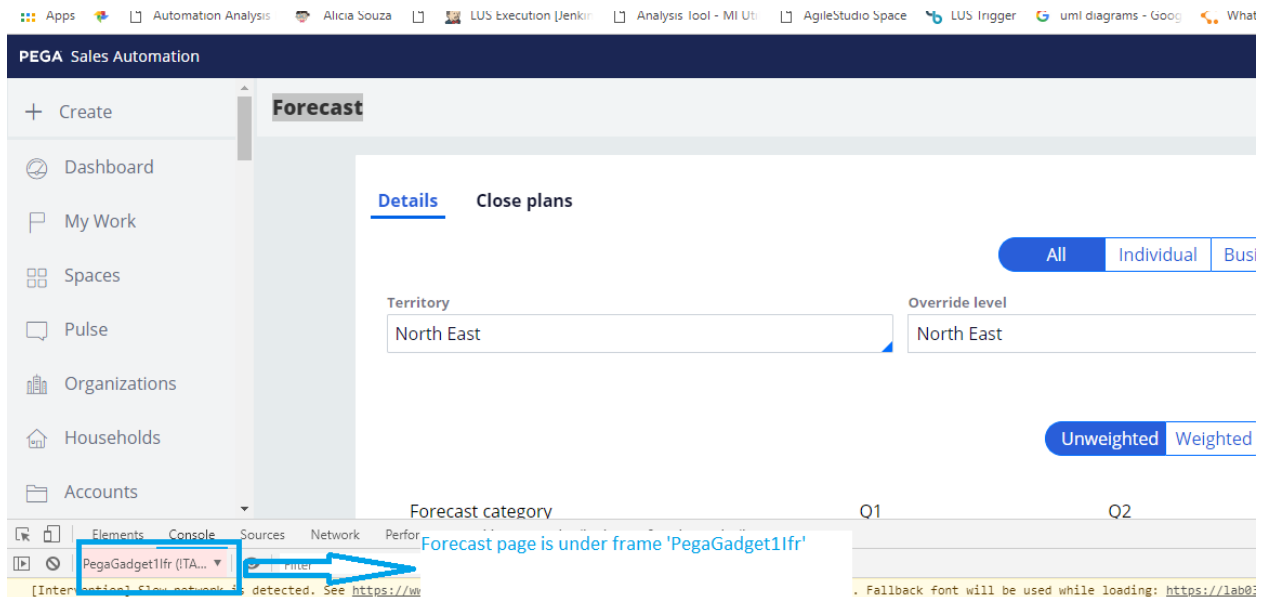
Since every portal class extends `TopDocumentImpl` class indirectly, all, the methods to do operations on the portal page like `findElement`, `findAutoComplete` etc. are readily available. One can see all the available methods by pressing CTRL+space keys on keyboard for eclipse IDE.



Forecast.java:

Every page for Pega application will either be present on top page or inside a frame. If the page is available on the top page, then page class should extend from **TopDocumentImpl** and if the page is available inside a frame then page should extend from **FrameImpl** from the base framework.

A Forecast.java is class written to keep the collection of all properties and behaviours for sample Forecast scenario used in this sample application. As seen in the below snippet, the Forecast page is under Frame 'PegaGadget1Ifn' so Forecast class should extend **FrameImpl** to get all the capabilities of **FrameImpl** class. **FrameImpl** class from the framework also has methods like **findElement**, **findAutoComplete**, **findSelectBox** etc, similar to **TopDocumentImpl**, to find different elements on the page and perform actions on them.



Code snippet for Forecast.java is as below:

```
public class Forecast extends FrameImpl {
    public Forecast(String frameID, TestEnvironment testEnv) {
        super(frameID, testEnv);
    }

    public static final By CLOSE_PLANS = By.xpath("//h3[text()='Close plans']");

    /**
     * Toggle to ClosePlans tab from forecast tab, within the same page/frame
     */
    public void switchToClosePlans() {
        findElement(CLOSE_PLANS).click();
    }
}
```

Page extends FrameImpl class

This method clicks on Close plans tab on the Forecast page

Note:

For FrameImpl Class we have the below constructors available, which should be used based the way available for us to identify a frame.

When a unique frameID is available on the iframe tag, the preference should always go to the below constructor

```
FrameImpl(String frameID, TestEnvironment testEnv)
```

The first argument, frameID, is the id for the Iframe tag for the frame which user is referring to and the second argument is the TestEnvironment object.

When there is no unique id available on an iframe tag, a css or xpath should be constructed for the iframe tag, find the iframe element on the page with findElement method and pass that PegaWebElement to this below constructor.

```
FrameImpl(PegaWebElement element)
```

The argument, element, is a WebElement which is identified with a locator for the Iframe tag for this Frame which user is referring to.

Both these above constructors are useful while dealing with a single Iframe. If there are inner frames, the above two constructors are not the right way creating the page objects. To create a page object for inner frames, use findFrame method from the outer frame page object, which returns a Frame object. Pass this frame object to the below constructor for creating a page object class for inner frames

```
FrameImpl(Frame frame)
```

The argument, frame, is the inner frame object returned by findFrame method of the outer frame. The inner frame using findFrame method can also be found either using the frame id or by finding the inner iframe as a pegawebelement.

Below is an example for creating a page object for Inner frames:

```
public class PredictionStudio extends FrameImpl{

    public PredictionStudio(String frameID, TestEnvironment testEnv) {
        super(frameID, testEnv);
    }

    @Override
    public Predictions getPredictions() {
        findElement(PREDICTIONS_TAB).click();
        Frame innerFrame = findFrame(getActiveFrameIdWithInThisFrame());
        Predictions predictions = new Predictions(innerFrame);
        return predictions;
    }
}
```

Note: For The tests shipped with the previous versions of the starter kit, there could be differences with the way Frame page object are defined.

A snippet from earlier version of framework (Selenium Starter Kit 2.0)

```
public PegaClosePlans(WebElement elmt, String elmtId) {
    super(elmt, elmtId);
}
```

which is now changed to (Selenium Starter Kit 2.1)

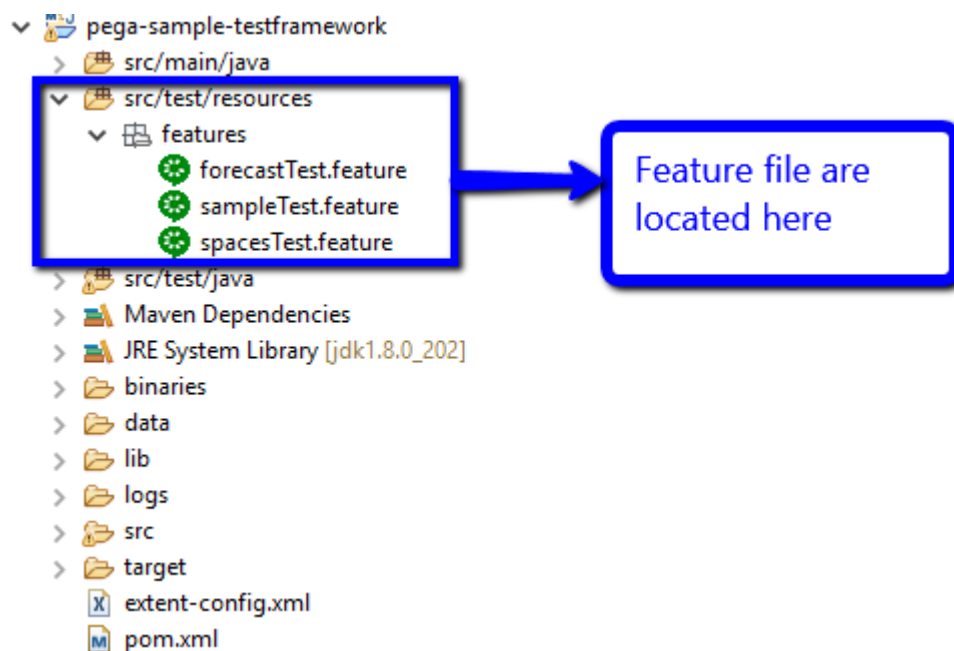
```
public PegaClosePlans(String frameId, TestEnvironment testEnv) {
    super(frameId, testEnv);
}
```

Feature files:

In this sample project, we employ Behavior-Driven Development (BDD) approach. BDD is a collaborative approach to software design and development. Discussing BDD is beyond the scope of this documentation but there are a lot of resources online for further reading. BDD separates the concerns of defining business outcomes/acceptance criterion from implementation. This separation of concern allows for Business and IT teams to collaborate efficiently.

In BDD style, Feature files, written in business language, are used to define executable specifications. Typically, Gherkin syntax is used to structure and define a feature. Step Definitions provide the implementation to corresponding to steps in a Feature file. In this example, step definitions are implemented in Java. In this sample project, we use [Cucumber BDD framework](#). For best practices on writing Gherkin, see [here](#)

All the feature files are organized under `src/test/resources/features` package.

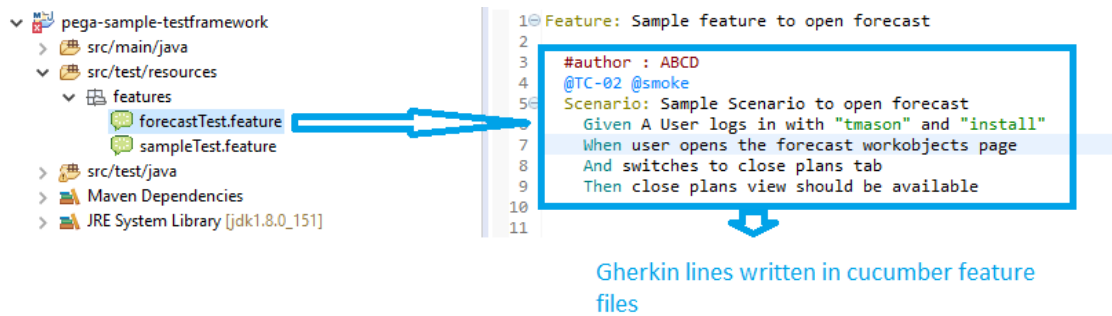


Forecast.feature:

Forecast.feature is sample feature file for the current test scenario, where user should open the Forecast page and then click on the Close plans tab and asserts a link in the close plan tab to make sure the tab is opened or not

It is always recommended to create separate packages for each scenario. In this sample project, the all the feature files are available at features package.

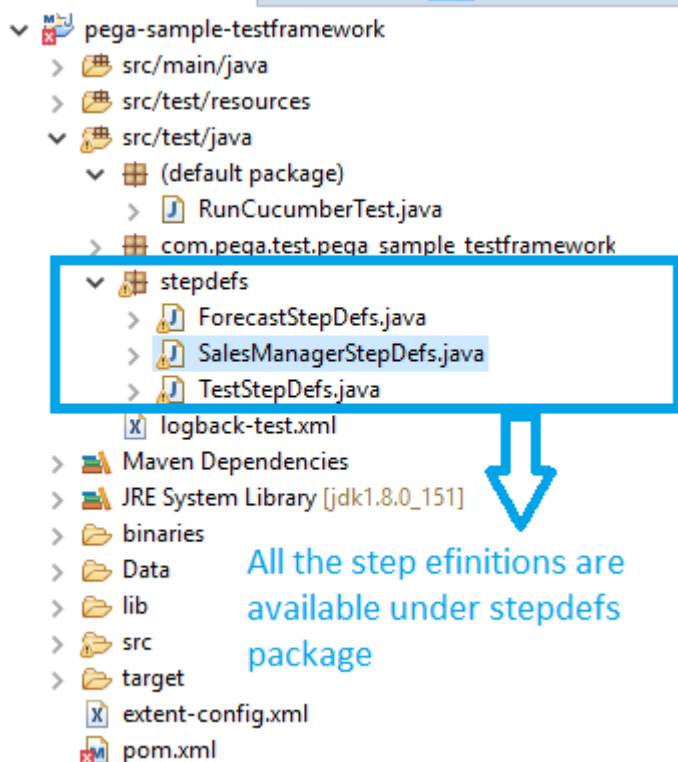
User can have duplicate gherkin lines in the same or different feature files.



Step Definitions:

Each step written in Feature file should be associated with a step definition. The step definition is the has the java logic to perform the operations mentioned in the Gherkin lines. All the step definition files are recommended to keep under `src/test/java/stepdefs` package.

User must have `@ScenarioScoped` annotation before step definition class. This annotation comes from cucumber-guice dependency as mentioned in below description. The usage of `@ScenarioScoped` annotation allows an object to be used in different step definitions present in the same class.



SalesManagerStepDefs.java :

This class contains the step definitions used by different test cases related to Sales manager portal.

```

@ScenarioScoped
public class SalesManagerStepDefs {
    TestEnvironment testEnv;
    com.pegasys.test.pegasys_testframework.MyAppBrowser browser;
    private PegaWebDriver pegaDriver;
    private SalesManagerPortal salesManagerPortal;
    private Forecast forecast;

    @Inject
    public SalesManagerStepDefs(MyAppTestEnvironment testEnv) {
        this.testEnv = testEnv;
        pegaDriver = testEnv.getPegaDriver();
        browser = (MyAppBrowser) testEnv.getBrowser();
        salesManagerPortal = browser.getPortal(SalesManagerPortal.class);
    }

    @When("^user opens the forecast workobjects page$")
    public void user_opens_the_forecast_workobjects_page(){
        forecast = salesManagerPortal.openForecast();
        MyAppObjectsBean.setForecast(forecast);
    }
}

```

MyAppTestEnvironment is injected to the constructor of every step definition class

Step definition for the gherkin line in ForecastTest.feature

The Object of class MyAppTestEnvironment is injected to the constructor of every step definition with the help of @inject annotation. This annotation comes from the Google Guice dependency specified in pom.xml of the user's test framework.

```

<dependency>
    <groupId>com.google.inject</groupId>
    <artifactId>guice</artifactId>
    <version>3.0</version>
</dependency>
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-guice</artifactId>
    <version>1.2.4</version>
</dependency>

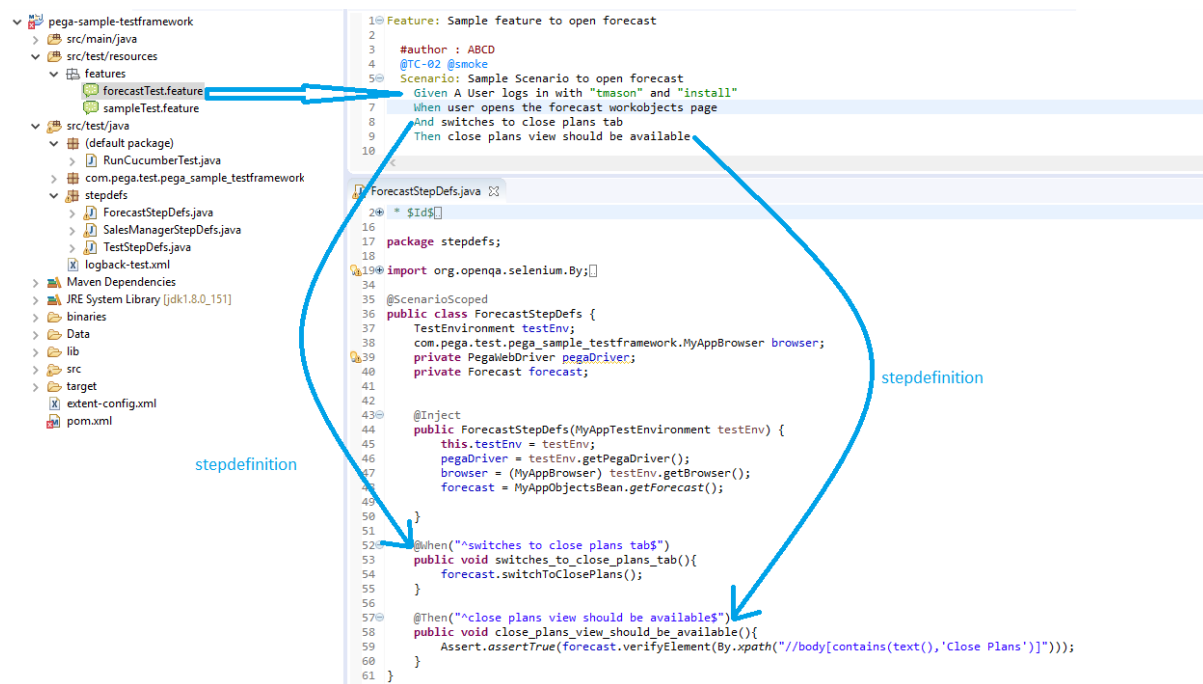
```

The object of MyAppTestEnvironment class can be used to get the handles for, MyAppBrowser , different portals etc. in the injected class.

etc. to carry out operations on these pages

ForecastStepDefs.java :

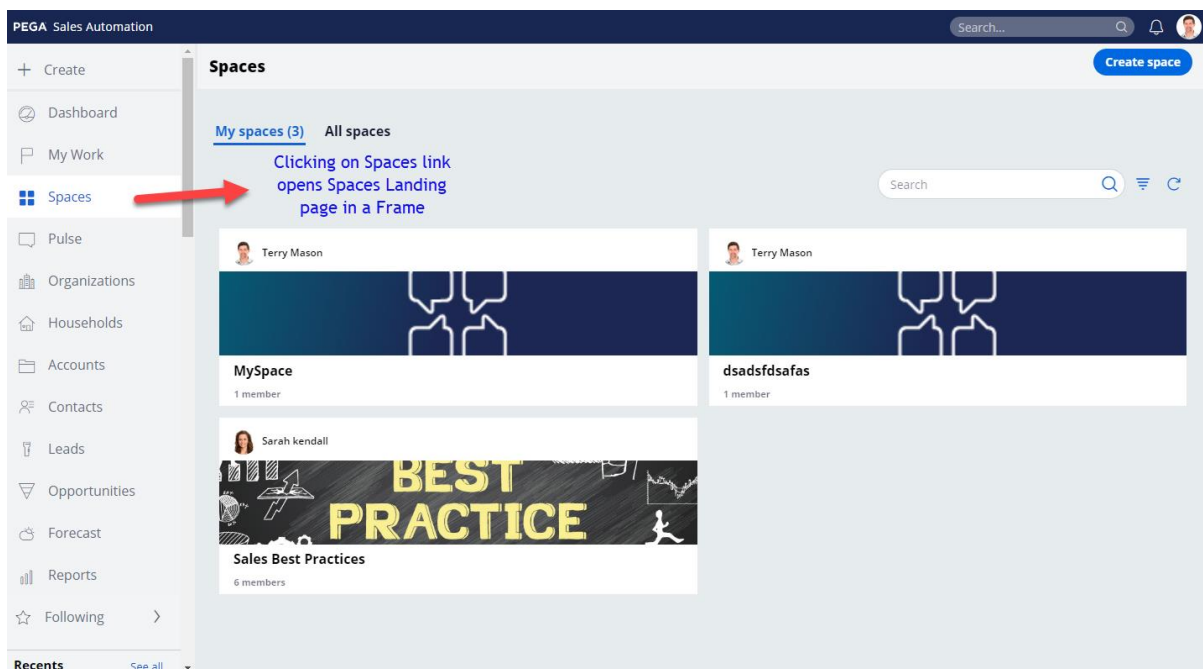
This class contains the step definitions which are used by different test cases related to Forecast page. Like any other stepdef class, this class also has a constructor with MyAppTestEnvironment injected to it, followed by various step definition methods.



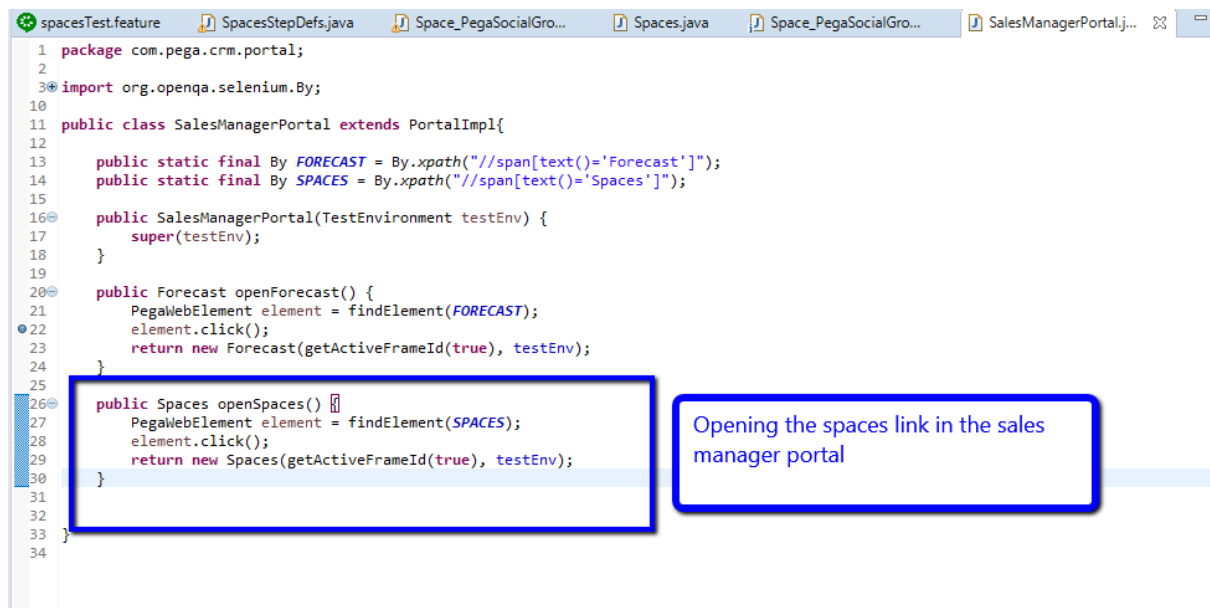
Spaces Test Case

Let's navigate through another testcase related to Spaces from SalesManagerPortalFirst, let's take a look at the page objects:

Actual flow in portal:

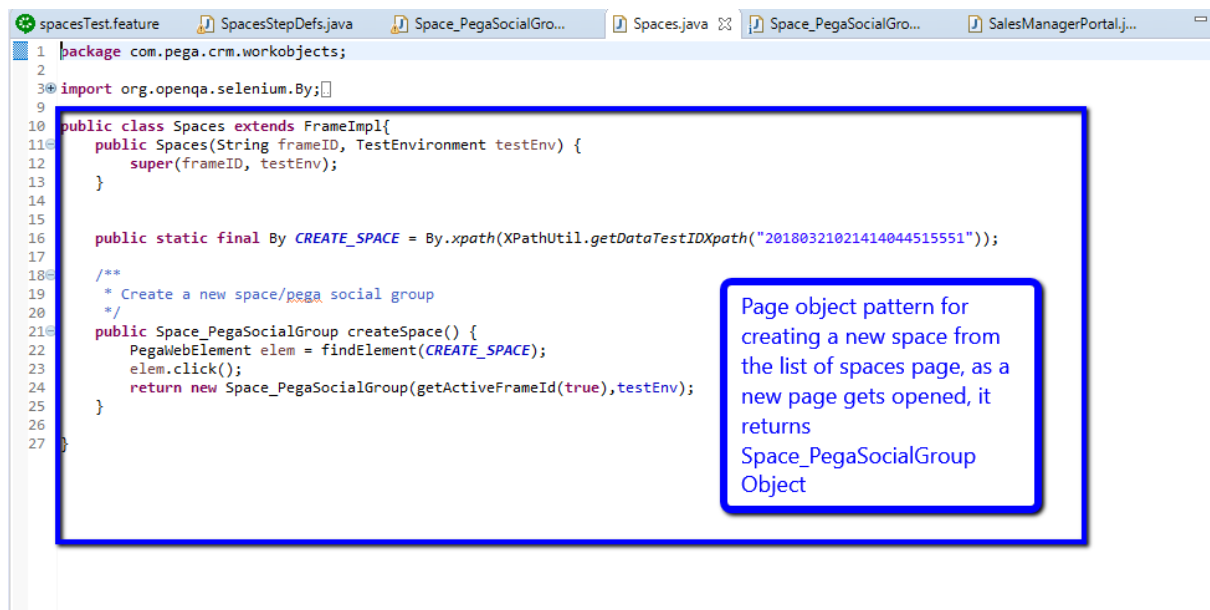


SalesManagerPortal class:



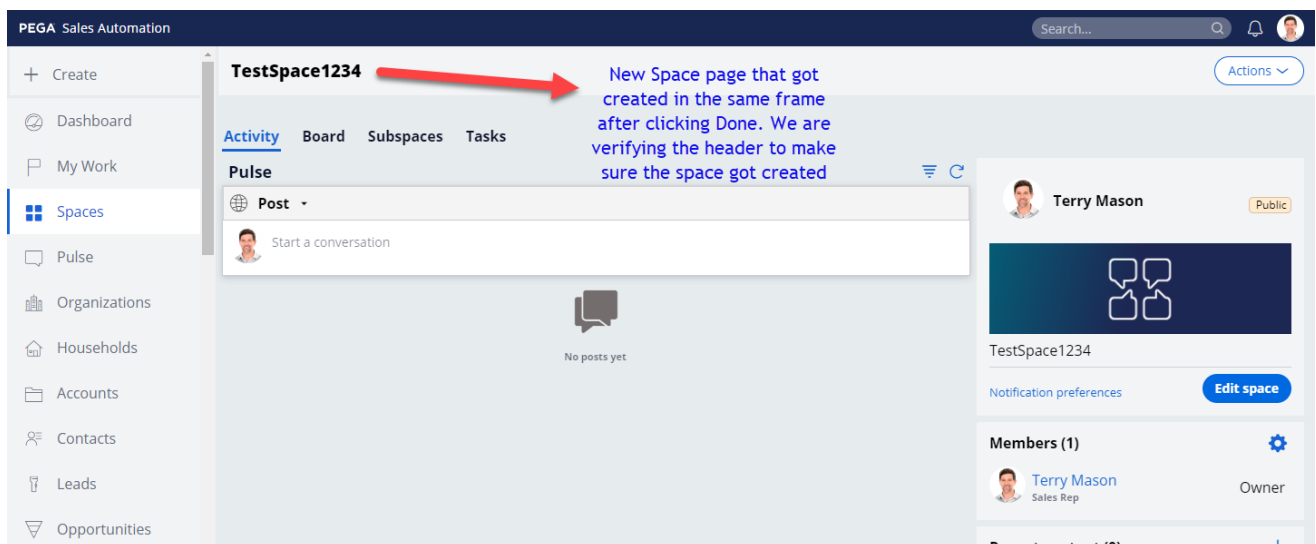
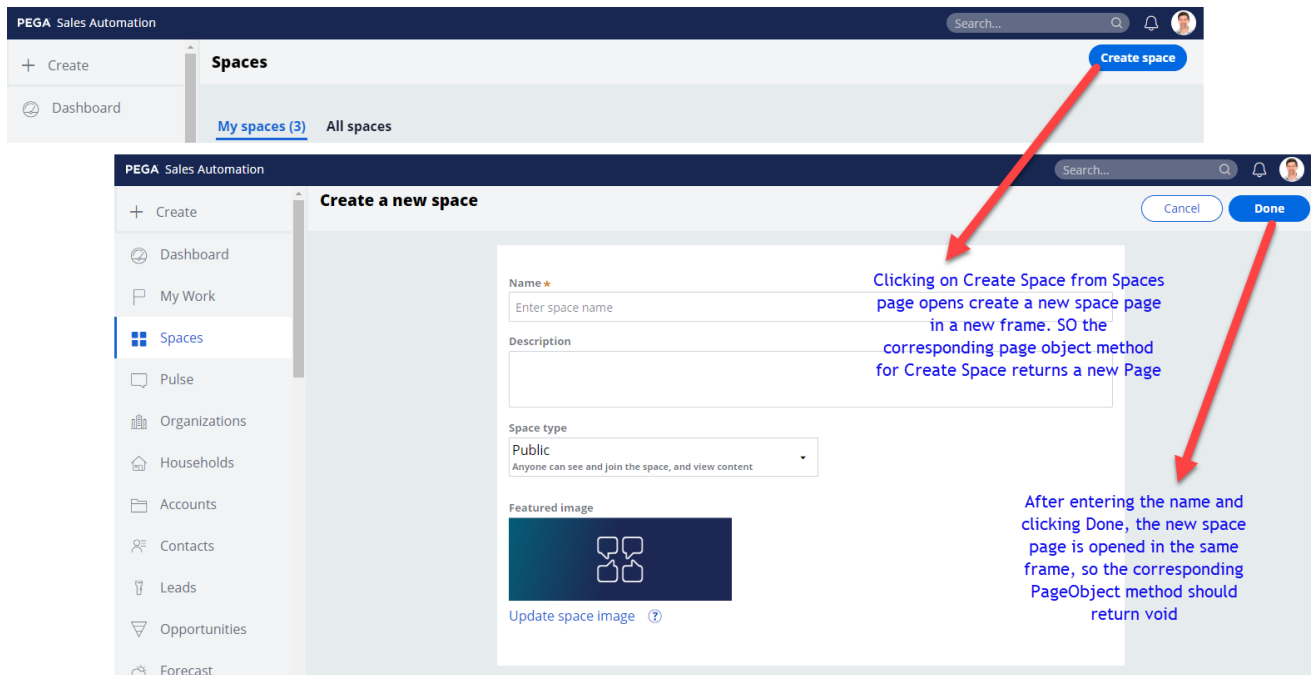
Spaces class:

An instance of this class is returned by `openSpaces` method in `SalesManagerPortal` class



This `createSpace` method is an example showcasing how a frame page can be opened from another frame. The way we create page object doesn't change as we automatically get the `activeframeid` even if the page is opened from an existing frame page, not just from the top document

Actual flow in portal:



Space_PegaSocialGroup class:

This class's object was returned by createSpace method in Spaces class. Even if we have opened an existing space instead of creating new one, then also this object should be returned as the same page gets opened in that case too.

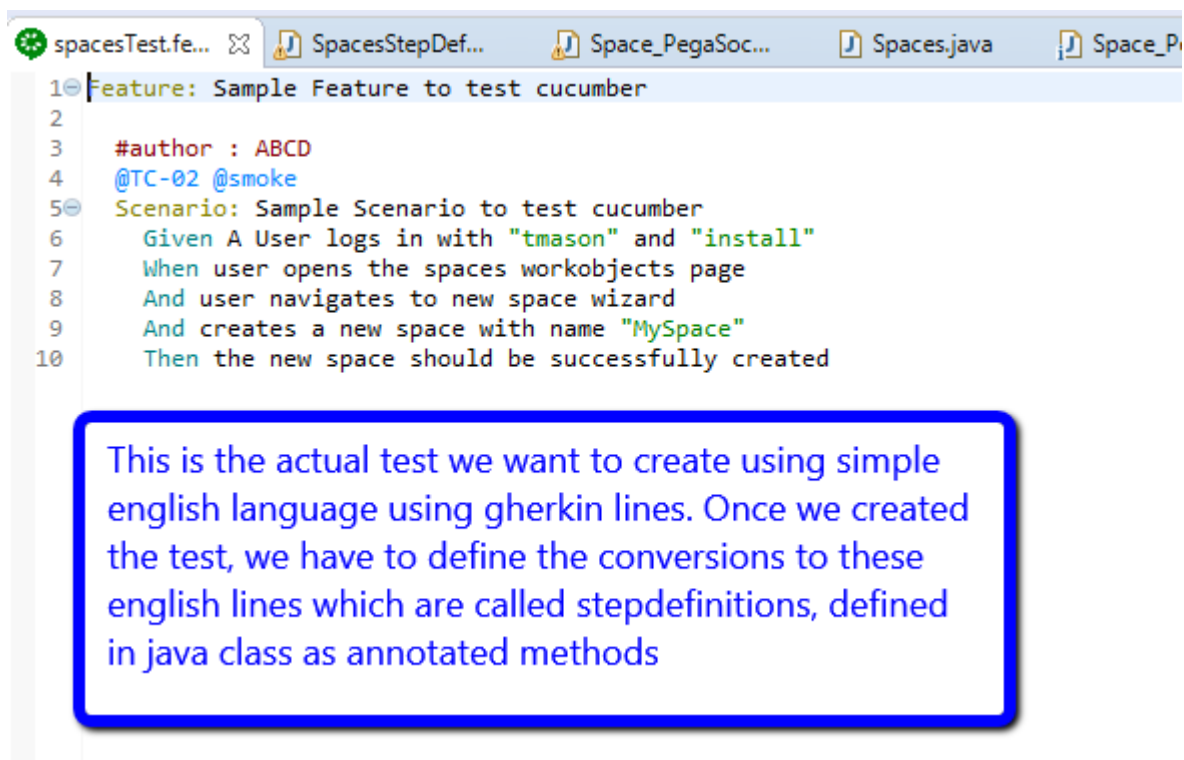


```
1 package com.pegacrm.workobjects;
2
3 import org.openqa.selenium.By;
4
5 public class Space_PegaSocialGroup extends FrameImpl {
6     public Space_PegaSocialGroup(String frameID, TestEnvironment testEnv) {
7         super(frameID, testEnv);
8     }
9
10    public static final By NAME = By.xpath(XPathUtil.getDataTestXPath("20180404041454077323962"));
11    public static final By DESCRIPTION = By.xpath(XPathUtil.getDataTestXPath("20180321013709017931436"));
12    public static final By DONE = By.xpath("//button[text()='Done']");
13
14    /**
15     * Toggle to ClosePlans tab from forecast tab, within the same page/frame
16     */
17    public void createSpace(String name) {
18        findElement(NAME).sendKeys(name);
19        findElement(DESCRIPTION).sendKeys(name);
20        findElement(DONE).click();
21    }
22
23    public boolean verifySpaceHeader(String spaceName) {
24        return verifyElement(By.xpath("//*[contains(@class,'header-title') and text()='"+spaceName+"'"));
25    }
26 }
```

Annotations and callouts in the image:

- Annotation: `By` (points to line 3)
- Annotation: `Page Object method which inputs the new space details and creates it` (points to the `createSpace` method)
- Annotation: `Verification method to check if the space was successfully created` (points to the `verifySpaceHeader` method)

Now let's define our test in a Cucumber gherkin file



```
1 Feature: Sample Feature to test cucumber
2
3 #author : ABCD
4 @TC-02 @smoke
5 Scenario: Sample Scenario to test cucumber
6     Given A User logs in with "tmason" and "install"
7     When user opens the spaces workobjects page
8     And user navigates to new space wizard
9     And creates a new space with name "MySpace"
10    Then the new space should be successfully created
```

Callout box text:

This is the actual test we want to create using simple english language using gherkin lines. Once we created the test, we have to define the conversions to these english lines which are called stepdefinitions, defined in java class as annotated methods

Now it's time for creating the stepdefinitions:

SalesManagerStepDefs class:

```

20 * $Id$
16
17 package stepdefs;
18
19 import com.google.inject.Inject;
20
21 @ScenarioScoped
22 public class SalesManagerStepDefs {
23     TestEnvironment testEnv;
24     com.pegasys.test.pegasys_testframework.MyAppBrowser browser;
25     private PegaWebDriver pegaDriver;
26     private SalesManagerPortal salesManagerPortal;
27
28
29
30 @Inject
31 public SalesManagerStepDefs(MyAppTestEnvironment testEnv) {
32     this.testEnv = testEnv;
33     pegaDriver = testEnv.getPegaDriver();
34     browser = (MyAppBrowser) testEnv.getBrowser();
35     salesManagerPortal = browser.getPortal(SalesManagerPortal.class);
36
37 }
38
39 @When("^user opens the forecast workobjects page$")
40 public void user_opens_the_forecast_workobjects_page(){
41     Forecast forecast = salesManagerPortal.openForecast();
42     MyAppObjectsBean.setForecast(forecast); //Sets the forecast object
43 }
44
45 @When("^user opens the spaces workobjects page$")
46 public void user_opens_the_spaces_workobjects_page(){
47     Spaces spaces = salesManagerPortal.openSpaces();
48     MyAppObjectsBean.setSpaces(spaces);
49 }
50
51 }
52
53
54
55
56
57
58
59
60
61

```

StepDefinition for opening the spaces page, using the page object we created for SalesManagerPortal.

Once we open the spaces page and get a page object for the same, we need to store it in Objects Bean to retrieve it later in other stepdefinition classes. We shall discuss this later in ObjectsBean section

SpacesStepDefs class:

```

20 * $Id$
16
17 package stepdefs;
18
19 import com.google.inject.Inject;
20
21 @ScenarioScoped
22 public class SpacesStepDefs {
23     TestEnvironment testEnv;
24     com.pegasys.test.pegasys_testframework.MyAppBrowser browser;
25     private PegaWebDriver pegaDriver;
26     private Spaces spaces;
27
28
29
30 @Inject
31 public SpacesStepDefs(MyAppTestEnvironment testEnv) {
32     this.testEnv = testEnv;
33     pegaDriver = testEnv.getPegaDriver();
34     browser = (MyAppBrowser) testEnv.getBrowser();
35
36 }
37
38 @When("^user navigates to new space wizard$")
39 public void user_creates_a_new_space_with_name(){
40     spaces = MyAppObjectsBean.getSpaces();
41     Space_PegaSocialGroup space_PegaSocialGroup = spaces.createSpace();
42     MyAppObjectsBean.setSpace_PegaSocialGroup(space_PegaSocialGroup);
43 }
44
45 }
46
47
48
49
50
51
52
53
54
55

```

Define your stepdefinition method using the When annotation above the method

First retrieve the spaces object created and stored in ObjectsBean class, then use the page objects methods from the spaces class for carrying out the actions on the

Space_PegaSocialGroupStepDefs class:



```
20 * $Id$
16
17 package stepdefs;
18
19 import org.openqa.selenium.By;
20
21 @ScenarioScoped
22 public class Space_PegaSocialGroupStepDefs {
23     TestEnvironment testEnv;
24     com.pegasys.pegasys_testframework.MyAppBrowser browser;
25     private PegaWebDriver pegaDriver;
26     private Space_PegaSocialGroup space_PegaSocialGroup;
27     private String name;
28
29     @Inject
30     public Space_PegaSocialGroupStepDefs(MyAppTestEnvironment testEnv) {
31         this.testEnv = testEnv;
32         pegaDriver = testEnv.getPegaDriver();
33         browser = (MyAppBrowser) testEnv.getBrowser();
34     }
35
36     @When("^creates a new space with name \"([^\"]*)\"$")
37     public void user_creates_a_new_space_with_name(String name) {
38         this.name = MyAppObjectsBean.putTimestampedValue(name);
39         space_PegaSocialGroup = MyAppObjectsBean.getSpace_PegaSocialGroup();
40         space_PegaSocialGroup.createSpace(this.name);
41     }
42
43     @Then("^the new space should be successfully created$")
44     public void the_new_space_should_be_successfully_created() {
45         Assert.assertTrue(space_PegaSocialGroup.verifySpaceHeader(name));
46     }
47 }
48
```

While we create a new space on application, we might end up running this test multiple times on sample application and hence, when we run for the second time a new space with same name should not be created. To avoid creating space with same name, we can use putTimestampedValue method from ObjectsBean, which appends a random time stamp and stores that value for this session of test to be able to retrieve in other steps by using getTimestampedValue method from the same ObjectsBean class

Use of ObjectsBean class :

In this sample project a java bean class is created with a getter and a setter methods which is used to store and retrieve the objects created for different pageobjects across multiple cucumber stepdefinition files . This class is MyAppObjectsBean.

The user creates the object of Forecast page in SalesManagerStepDefs.java. However, the same forecast object should be used throughout the test case in different step definitions which can be scattered in various classes. Thus, to transfer a page object from one class to another class we use JavaBean class.

This is needed because as and when a gherkin file is being executed, cucumber-jvm tries to find a matching step definition in java classes, then the cucumber guice will create an object using the constructor which has @Inject annotation on top of it automatically. An explicit object creation step for a step definition class is not added, hence will not have an option to transfer a state of a page object to different step definition classes. So, to store these page objects whenever they are created, developer should push them to static variables in ObjectsBean class via the setter methods and reuse them in different step definition classes via the getter methods.

java bean class

```
1 package com.pegatest.pegasample_testframework;
2 import com.pegatest.crm.workobjects.Forecast;
3
4 public class MyAppObjectsBean {
5     private static Forecast forecast;
6
7     public static Forecast getForecast() {
8         return forecast;
9     }
10
11     public static void setForecast(Forecast forecast) {
12         MyAppObjectsBean.forecast = forecast;
13     }
14 }
15
16
17
18
```

getters and setters for forecast object

```
19 package stepdefs;
20 import org.openqa.selenium.By;
21
22 @ScenarioScoped
23 public class ForecastStepDefs {
24     TestEnvironment testEnv;
25     com.pegatest.pegasample_testframework.MyAppBrowser browser;
26     private PegawebDriver pegawebDriver;
27     private Forecast forecast;
28
29     @Inject
30     public ForecastStepDefs(MyAppTestEnvironment testEnv) {
31         this.testEnv = testEnv;
32         pegawebDriver = testEnv.getPegawebDriver();
33         browser = (MyAppBrowser) testEnv.getBrowser();
34         forecast = MyAppObjectsBean.getForecast();
35     }
36
37     @When("^switches to close plans tab$")
38     public void switches to close plans tab() {
39         forecast.switchToClosePlans();
40     }
41 }
42
```

User gets the already created object of Forecast page from the java Beans class

Same forecast object is used in every stepdefinition files

The first step definition class where an object is created, if user wants to use the same object in a different step definition class, then user must use one of the setter methods to store the data in the MyAppObjectsBean class.

```

9+ import org.openqa.selenium.By;
4
5 @ScenarioScoped
6 public class SalesManagerStepDefs {
7     TestEnvironment testEnv;
8     com.pegatest.pegasampletestframework.MyAppBrowser browser;
9     private PegaWebDriver pegaDriver;
0     private SalesManagerPortal salesManagerPortal;
1     private Forecast forecast;
2
3
4 @Inject
5     public SalesManagerStepDefs(MyAppTestEnvironment testEnv) {
6         this.testEnv = testEnv;
7         pegaDriver = testEnv.getPegaDriver();
8         browser = (MyAppBrowser) testEnv.getBrowser();
9         salesManagerPortal = browser.getPortal(SalesManagerPortal.class);
0
1     }
2
3 @When("^user opens the forecast workobjects page$")
4     public void user_opens_the_forecast_workobjects_page(){
5         forecast = salesManagerPortal.openForecast();
6         MyAppObjectsBean.setForecast(forecast);
7     }
8 }

```

⇒ User stores the forecast object into MyAppObjectsBean class

```

ForecastStepDefs.java
2+ * $Id$
6
7 package stepdefs;
8
9+ import org.openqa.selenium.By;
4
5 @ScenarioScoped
6 public class ForecastStepDefs {
7     TestEnvironment testEnv;
8     com.pegatest.pegasampletestframework.MyAppBrowser browser;
9     private PegaWebDriver pegaDriver;
0     private Forecast forecast;
1
2
3 @Inject
4     public ForecastStepDefs(MyAppTestEnvironment testEnv) {
5         this.testEnv = testEnv;
6         pegaDriver = testEnv.getPegaDriver();
7         browser = (MyAppBrowser) testEnv.getBrowser();
8         forecast = MyAppObjectsBean.getForecast();
9     }
0 }

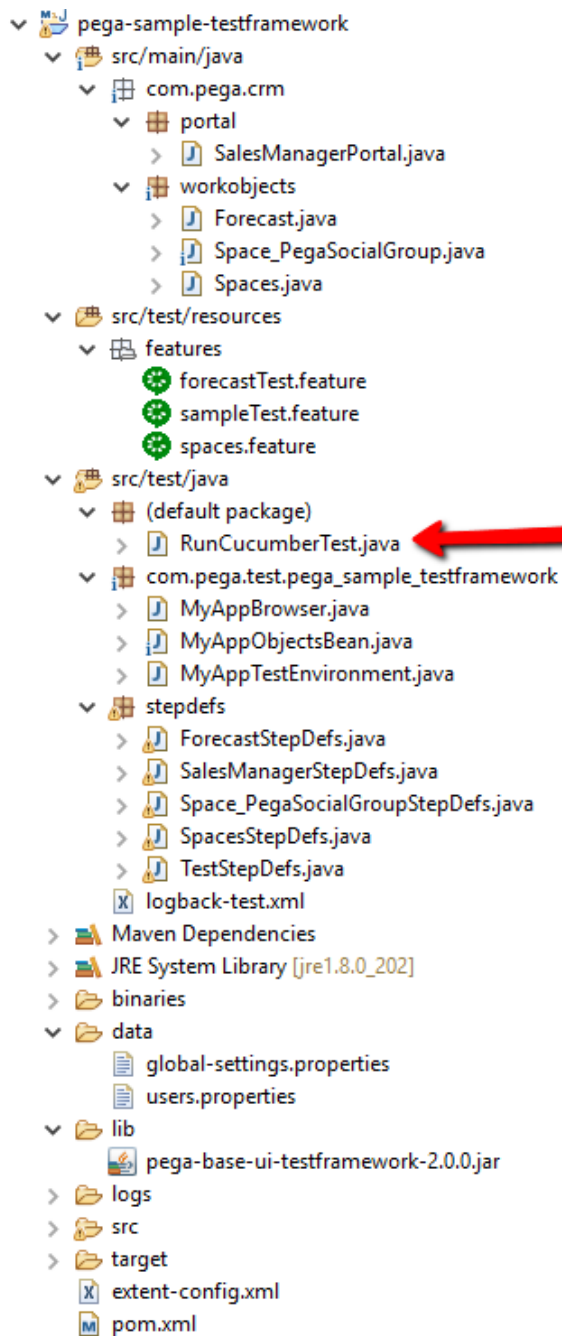
```

⇒ User retrieves the object of Forecast page from MyAppObjectsBean class in a different step definition file

RunCucumberTest.java:

This is the class from where cucumber starts executing the tests. This class must have a `@CucumberOptions` annotation. This annotation tells Cucumber a lot of things like where to look for feature files, what reporting system to use and some other things also. But for the sake of this project, it's configured for `dryRun` and `monochrome`. The test runner class also acts as an interlink between feature files and step definition classes.

User must extend the runner class from `AbstractTestNGCucumberTests` class.



Cucumber Runner Class

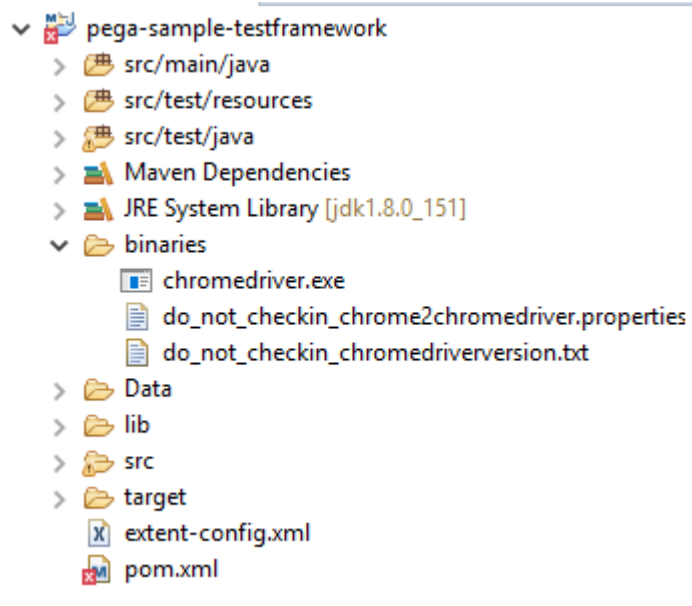
Running the Tests

Once you have your new tests developed, running them follows the same steps as running the out of the box tests. One can check the Selenium Starter Kit - Running Tests guide to run the tests

Binaries and lib folders

Folder *binaries*:

By default, we attempt to download an appropriate chrome driver automatically through our custom utility. If it fails, copy the driver manually to binaries folder. User can also place the binaries for other browsers in the same folder.



lib Folder:

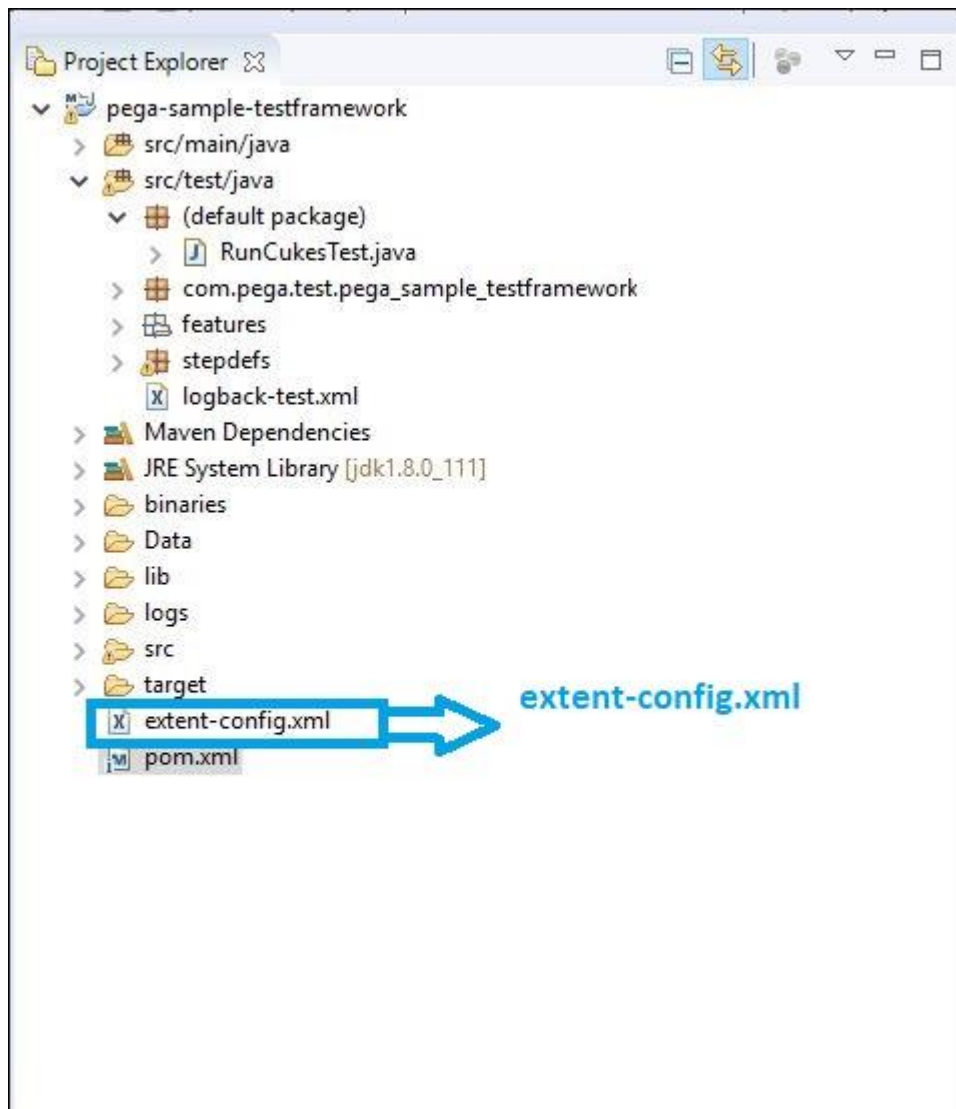
The lib folder contains the library `pega-base-ui-testframework-2.0.0.jar`, which is the base MI framework.

Results & Reporting

Extent-config file:

By using this external XML file (extent-config.xml), we could change the details such as Report Theme (either standard or dark), Report Title, Document Title etc.,

This is present in our sample project at the location as seen in below snippet:



extentReport.html:

extentReport.html is an html report generated by executing test case in a project. For generation of extentReport.html you need to have two dependencies to be included in pom.xml. Those are as shown in below snippet:

```
<dependency>
<groupId>com.vimalselvam</groupId>
<artifactId>cucumber-extentreport</artifactId>
<version>3.1.1</version>
</dependency>
<dependency>
<groupId>com.aventstack</groupId>
<artifactId>extentreports</artifactId>
<version>3.1.1</version>
</dependency>
```

dependencies for
extentReport

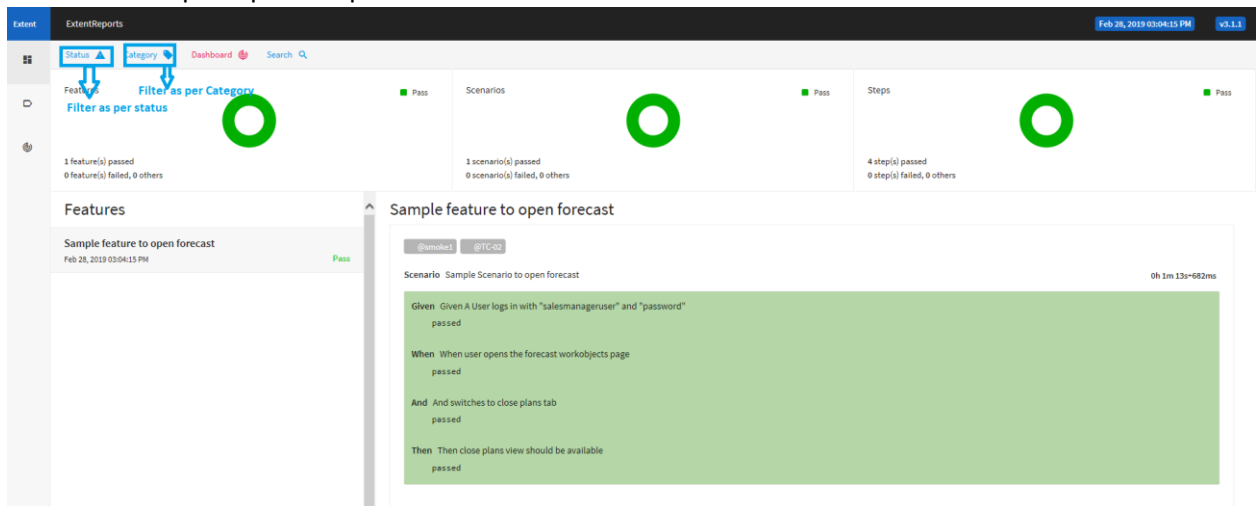
The next step is to add plugin for extent report in RunCucumberTest.java. The code snippet for same will look as in below:

```
RunCucumberTest.java
1 |
2 | import cucumber.api.CucumberOptions;
3 |
4 |
5 |
6 | @CucumberOptions(plugin = {"com.vimalselvam.cucumber.listener.ExtentCucumberFormatter:target/extentReport.html"})
7 | public class RunCucumberTest extends AbstractTestNGCucumberTests{
8 |
9 |     String COPYRIGHT = "Copyright (c) 2018 Pegasystems Inc.";
10 |    String VERSION = "$Id: RunCukesTest.java 121819 2018-01-26 07:29:51Z SachinVellanki $";
11 |
12 | }
13 |
14 |
```

Once we execute test case, result will be available in target folder with html file extentReport.html.

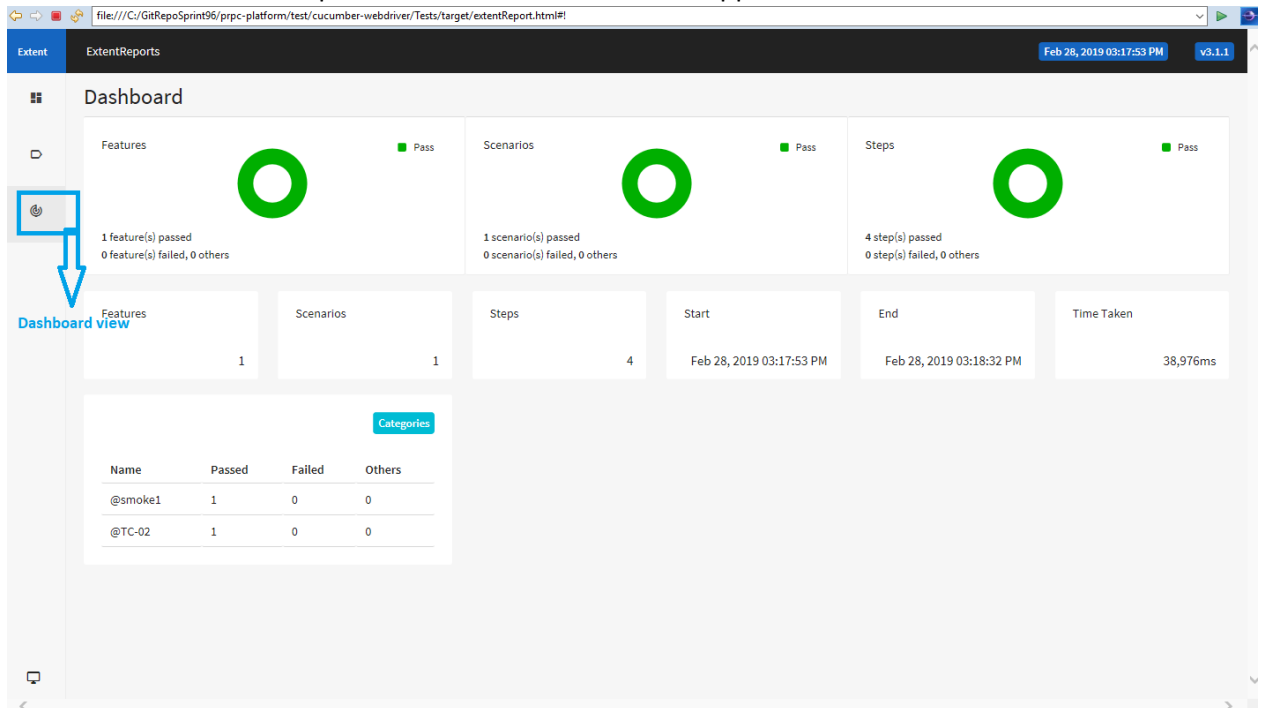
Passed Test Report

Below is a sample report for passed test cases



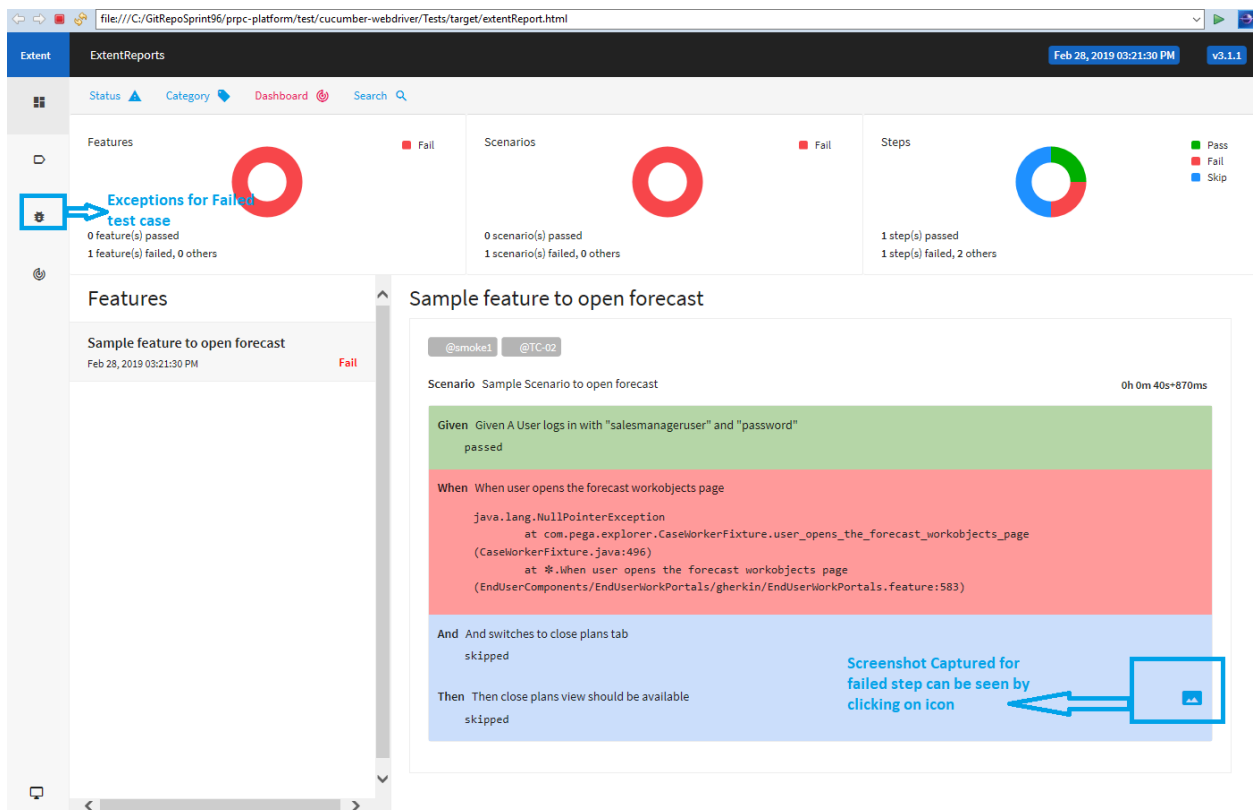
User can filter reports by status (passed, failed, skipped) or by category (by tags) as shown above.

Dashboard view for the report will be as shown in below snippet:



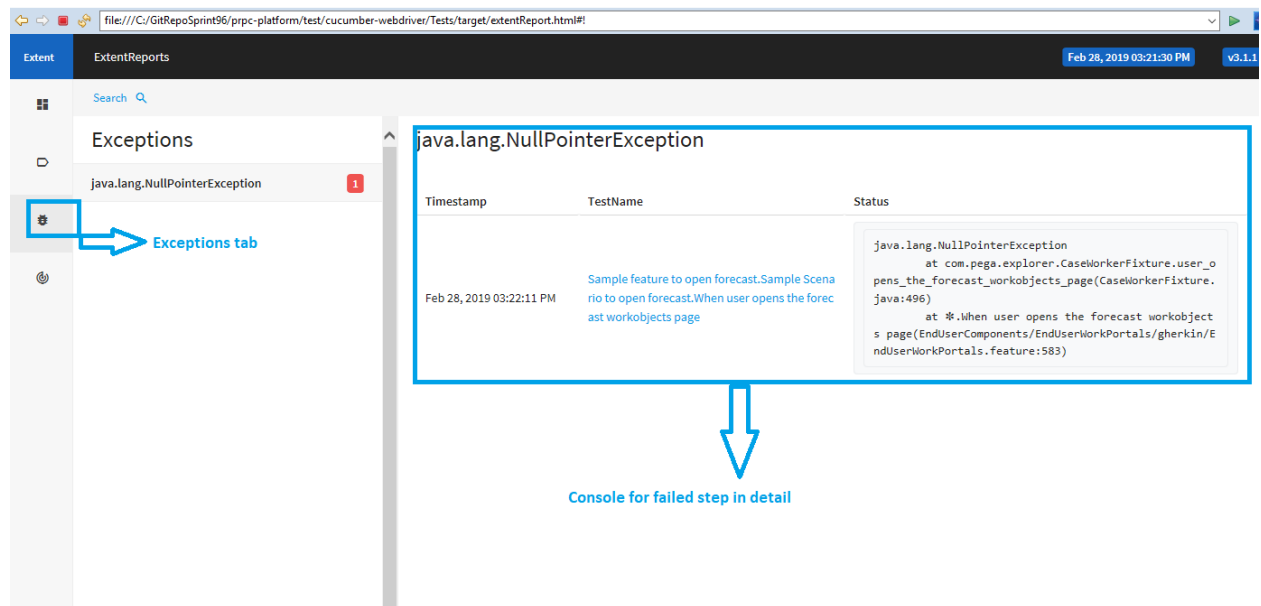
Failed Test Report

Extent report for the failed test case will look as below:



Screenshot captured at failed step can be seen by clicking icon as shown in above snippet.

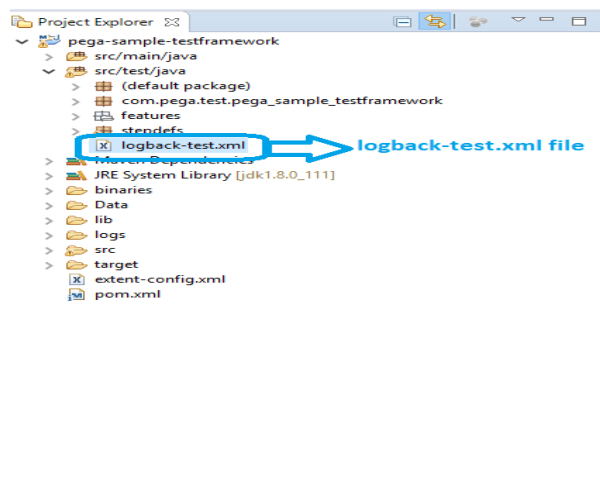
For failed test cases extent report will give Exceptions tab to make debugging easy. Exceptions tab will look like as shown in snippet below:



Control the Logging mechanism

Logback-test.xml:

With logback-test.xml, user is provided with logging utility. logback will try to configure itself using the file logback-test.xml. logback is nothing but a logging framework. We have logback-test.xml file in folder structure as shown in below snippet:



By default the logging level is set to info mode. To turn it to a different mode, open the logback-test.xml file and change the root level to one of debug, warn or error modes


```

logback-test.xml
9  <appender name="RootSiftAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
10    <discriminator>
11      <Key>testname</Key>
12      <DefaultValue>application</DefaultValue>
13    </discriminator>
14    <sift>
15      <appender name="FILE-${testname}"
16        class="ch.qos.logback.core.rolling.RollingFileAppender">
17        <File>logs/${testname}.log</File>
18        <rollingPolicy
19          class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
20          <FileNamePattern>${testname}.%i.log</FileNamePattern>
21          <MinIndex>1</MinIndex>
22          <MaxIndex>100</MaxIndex>
23        </rollingPolicy>
24        <triggeringPolicy
25          class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
26          <MaxFileSize>5MB</MaxFileSize>
27        </triggeringPolicy>
28        <layout class="ch.qos.logback.classic.PatternLayout">
29          <Pattern>%d{ISO8601} %-5level %C{1} [%M:%L] [%thread] - %msg%n</Pattern>
30        </layout>
31      </appender>
32    </sift>
33  </appender>
34  <root level="info">
35    <appender-ref ref="RootSiftAppender" />
36    <appender-ref ref="STDOUT" />
37  </root>
38
39 </configuration>

```

Generate a video for a test

For recording video, set property 'enable.video.recording' from global-settings.properties to be true.

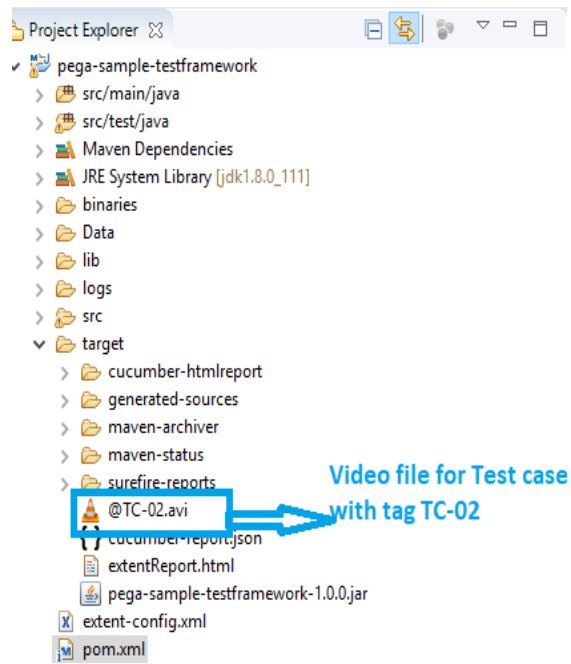
```

# -----
# Other settings
# -----
l10n.language=EN
chrome.extension.paths=
enable.video.recording=true
l10n.bundle.path=/localization/TextToTrans.
launch.brower.with.addOns=false
firefox.addOns.paths=

```

enable.video.recording should be set 'true' for video recording

Video file for the executed test case will be available under target folder with that tag name of executed test case. To record a video, user must leave the screen as is and let the test run on the foreground. If tests are running in parallel video recording is not supported.



API Reference


TopDocument

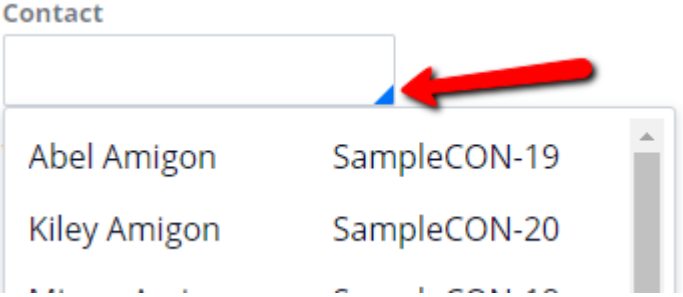
As mentioned above in the document, this is a framework class which should be extended by a pageobject class when a html page is not on any frame and directly resides in top level context. We will see what methods and fields are available out of the box from this class. All the methods in TopDocument class first switches the driver context to defaultcontent, i.e, switches out of all the frames.

Fields

Name	Type	Description
testEnv	TestEnvironment	Fundamental class of the framework that describes test environment. You can retrieve any framework object using testEnv
pegaDriver	PegaWebDriver	The wrapper created on top of Selenium's webdriver with some additional methods
actions	Actions	The Selenium's driver actions class object
scriptExecutor	ScriptExecutor	A base framework class which has methods to execute javascript directly on the instance opened in the browser

Methods

Name	Arguments	Description
findElement()		Finds an Element on the page using Selenium's findElement method and wraps the element with framework class PegaWebElement and hence returns a PegaWebElement
findElements()		Finds all the elements on the page with given location strategy and returns a list of WebElements
findSelectBox()		This method should be used instead of findElement to find a DropDown field on a page. It returns an object for DropDown class. This class has wrapper methods for handling dropdown related actions Lead source 
findAutoComplete()		This method should be used instead of findElement to find a AutoComplete field on a page. It returns an object for AutoComplete class. This class has wrapper methods for handling autocomplete related actions

		
<code>findFrame()</code>		Finds a frame on the top level document and switches to it. It returns a <code>Frame(Base Framework Class)</code> Object
<code>getActiveFrameID()</code>		Returns the current active frame on the application. If using the overloaded method and provides argument as true, then it switches to the active frame as well (Only for frames with id as <code>PegaGadget<x>Ifr</code>)
<code>handleWaits()</code>		Returns a <code>WaitHandler</code> class Object, which has methods handling wait times and scrip to application synchronization
<code>verifyElementVisible()</code>		Booolean indicating visibility of an element on a page after finding the element.
<code>verifyElement()</code>		Verifies if an element is available on page and returns true or false

Frame class:

As mentioned above in the document, this is a framework class which should be extended by a `pageobject` class when a html page is on a Frame. Almost all the methods that we see in the `TopDocument` class will be available in the `Frame` class, but the execution would happen after switching to the frame context for every method.

PegaWebDriver class:

`PegaWebDriver` is a wrapper on top of `Selenium WebDriver` class. Apart from the `WebDriver` API's, we added few more methods to make automation on Pega applications easier. A lots of methods that `PegaWebDriver` has like `findElement`, `findSelectBox`, `verifyElement` etc are already called by the `TopDocument` and `Frame` classes, which a user should call from those respective classes only. Apart from these methods following methods are available

Methods

Name	Arguments	Description
<code>waitForDocStateReady()</code>		waits for the entire document to load including background page loading
<code>switchToActiveFrame()</code>		switches to current active frame on the application (Only for frames with id as PegaGadget<x>Ifr)
<code>verifyAndWaitIfThrobberPresent()</code>		verifies if a throbber is present and waits for the throbber to disappear from the page.

PegaWebElement class:

PegaWebElement is a wrapper class around Selenium WebElement class, which is returned when findElement method is used in TopDocument, Frame and PegaWebDriver classes. Apart from the regular methods there are few methods added to this class which helps user to do actions on an element like moving the virtual mouse to a particular element, draganddrop, rightclick etc.

Methods

Name	Arguments	Description
<code>doubleClick()</code>		Double clicks an element using Selenium Actions class. It wraps up all the code needed to initialize the actions class
<code>rightClick()</code>		Right clicks an element using Selenium Actions class. It wraps up all the code needed to initialize the actions class
<code>Mouseover()</code>		Mouse hovers on the element using the javascript mouseover method
<code>scrollIntoView()</code>		Scrolls an element in to view with a javascript method
<code>doClickWithMouse()</code>		<ol style="list-style-type: none">1) This method works only when a browser is launched in a full screen mode and doesn't work in a Selenium Grid environment.2) This method has to be used for local executions only using the pixel information for the element

		<ol style="list-style-type: none"> 3) It clicks on an element by moving the virtual mouse icon to the element.
<code>doRightClickWithMouse()</code>		<ol style="list-style-type: none"> 1) This method works only when a browser is launched in a full screen mode and doesn't work in a Selenium Grid environment. 2) This method has to be used for local executions only using the pixel information for the element 3) It right clicks on an element by moving the virtual mouse icon to the element.
<code>moveMouseToThis()</code>		<ol style="list-style-type: none"> 1) This method works only when a browser is launched in a full screen mode and doesn't work in a Selenium Grid environment. 2) This method has to be used for local executions only using the pixel information for the element 3) It moves the virtual mouse icon to the element.
<code>dragAndDrop()</code>		<ol style="list-style-type: none"> 1) This method works only when a browser is launched in a full screen mode and doesn't work in a Selenium Grid environment. 2) This method has to be used for local executions only using the pixel information for the element 3) It drags and drops the current element to the element provided in the method arguments by moving the virtual mouse icon to the element.

DropDown class:

DropDown class provides methods to handle a dropdown element found on a webpage. Its object is returned when we call findSelectBox method from the TopDocument or Frame class

Methods

Name	Arguments	Description
<code>selectByVisibleText()</code>		Selects an option based on the given text
<code>selectByIndex()</code>		Selects an option with the help of given index based on index attribute of the option elements
<code>selectByValue()</code>		Selects the option based on the value attribute of the available options
<code>getOptions()</code>		Returns list of all the options available as webelements, on the select element

AutoComplete class:

AutoComplete class provides methods to handle an autocomplete element found on a webpage. Its object is returned when we call findAutoComplete method from the TopDocument or Frame class

There are two methods in AutoComplete class which help user in typing in the value to the field, wait for the options to appear and then select the option appearing on the field, then wait for the document to be ready for next action. All these actions are wrapped internally in to the following methods

Methods

Name	Arguments	Description
<code>setValue()</code>		Used when the autocomplete values populated is just a normal list of values and not appear as a dropdown options in the html
<code>selectValue()</code>		Used when the autocomplete values populated are displayed in options tag within the html page

So which method to use while using one of the above methods to set an autocomplete field should be made by inspecting the html tag of the list of options that gets autopopulated

ScriptExecutor class:

ScriptExecutor class provides few predefined options to the users to perform actions on elements via Javascript execution. Apart from the predefined options, users can also execute their own script by providing the script to the executeScript method

Methods

Name	Arguments	Description
<code>mouseover()</code>		Used when the autocomplete values populated is just a normal list of values and not appear as a dropdown options in the html
<code>clear()</code>		Used when the autocomplete values populated are displayed in options tag within the html page
<code>sendKeys()</code>		Sets the value attribute with the given string for an element
<code>fireKeyboardEvent()</code>		Fires a given keyboard event like onKeyPress/onKeyDown/onKeyUp etc on a given element
<code>click()</code>		Fires javascript click method on the given element
<code>rightClick()</code>		Fires javascript right click method on the given element
<code>getInnerText()</code>		Returns the innertext or text content for a given element
<code>executeJavaScript()</code>		Executes given javascript code on the top level page context





















Utilities:

Apart from the classes we created to help create page objects and make the automation experience on Selenium webdriver easier, we have also added lots of utility classes to the framework which can be seen below.

These utility classes make it easier to handle few automation scenarios when there is some extra java code is needed to automate the scenario.

Say, for example, we need to get the current date/month/year to be used in automating a scenario, then instead of writing the code for getting the current date, you can use the DateUtil class methods from the Framework and get these values with the help of predefined methods.

The javadocs on the methods for these framework classes should help you with what each of these util class methods does.

- ▼  > util
 - >  AssertUtil.java
 - >  DataBaseUtil.java
 - >  DataTestIdUtil.java
 - >  DataUtil.java
 - >  DateUtil.java
 - >  ElementUtil.java
 - >  ExcelUtil.java
 - >  FileUtil.java
 - >  GlobalConstants.java
 - >  GroupAsserts.java
 - >  HTTPUtil.java
 - >  LocalizationUtil.java
 - >  LoggerUtils.java
 - >  MailUtil.java
 - >  PDFUtil.java
 - >  RecorderUtil.java
 - >  ScreenshotUtil.java
 - >  WaitUtil.java
 - >  XPathUtil.java