

Document Version History

***Prepared by Webdunia***

***Version 1.0***

***Brothers Ecommerce***

***Code Design Explanation Document***

Webdunia (India) Pvt. Ltd.

Labh Ganga 582, M G Road

Indore 452 003

Madhya Pradesh, INDIA

Tel: +91-731-4273409  
Web: [www.webdunia.net](http://www.webdunia.net)

www.webdunia.com

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **Addition/Modification** | | **Prepared By** | **Reviewed By** | **Approved By** |
| 1.0 | June 6, 2020 | Updated | Arth Kumar | | Gaurav V | Babita Jain |

Contents

[AUTOMATION FRAMEWORK 3](#_Toc46248944)

[1. SRC/Main/Java Package 4](#_Toc46248945)

[1.1 suite.constants: 5](#_Toc46248946)

[1.2 suite.testdata 7](#_Toc46248947)

[1.3 suite.util 10](#_Toc46248948)

[2. src/main/resources 11](#_Toc46248949)

[2.1 Log4j.properties 12](#_Toc46248950)

[3. src/test/java 12](#_Toc46248951)

[3.1 suite.base 13](#_Toc46248952)

[3.2 suite.init 16](#_Toc46248953)

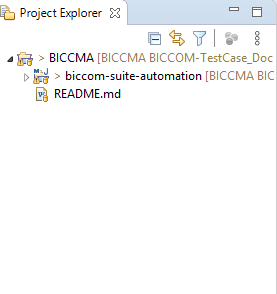
[3.3 suite.listener 18](#_Toc46248954)

[3.4 suite.page 21](#_Toc46248955)

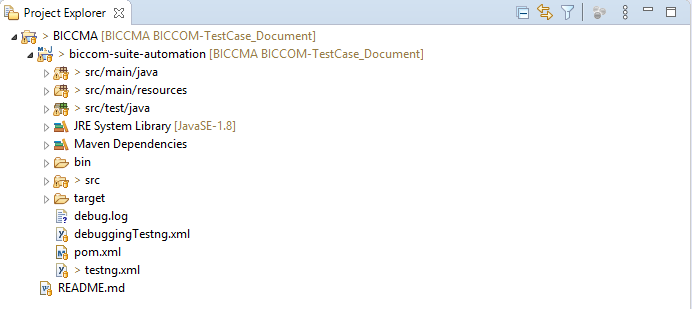
[3.5 suite.service: 22](#_Toc46248956)

[3.6 suite.test 23](#_Toc46248957)

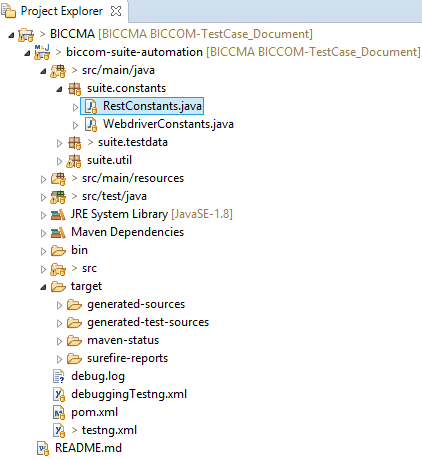
# AUTOMATION FRAMEWORK

**`**

* Open BICCMA project folder.



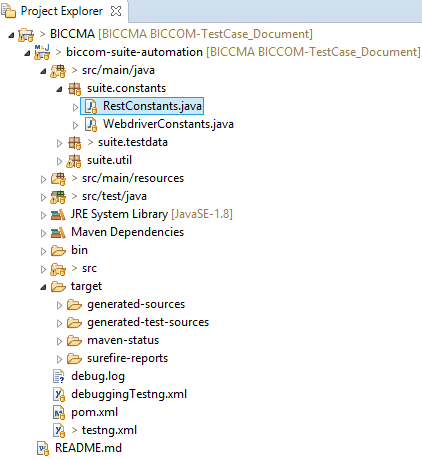
* Open biccom-suite-automation sub-folder.
* The biccom-suite-automation sub-folder contains packages and files related to automation test cases.



# SRC/Main/Java Package

Open **src/main/java** packageIt contains all the constants, test data, and libraries which will be required while designing the automation test cases. This package contains sub-packages:

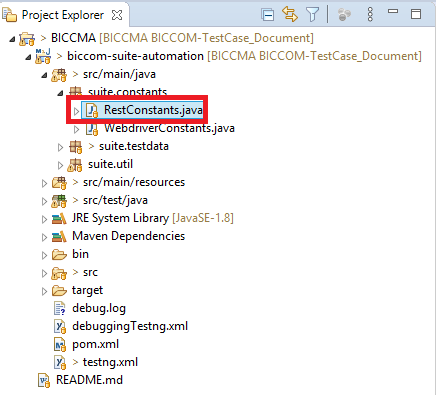
* suite.constants
* suite.testdata
* suite.util

****

# suite.constants:

open **suite.constants** sub-package**.** This contains declaration for all the constants to be used in the framework**.** It contains two java classes namely:

* **RestConstants.java**
* **WebDriverConstants.java**

****

**RestConstants.java:**

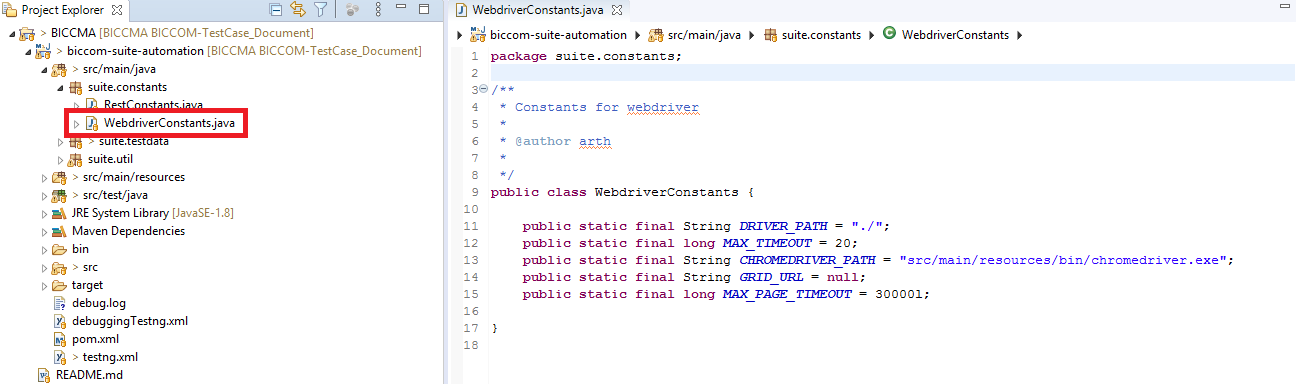
This class contains the rest-assured constants.

* **For example:** **public** **static** **final** **int** ***SUCCESS\_CODE*** = 200;

The above declared SUCCESS\_CODE variable and its value denotes the successful loading of a site or a page URL.

* **For example: public** **static** **final** **int** ***SERVER\_ERROR\_CODE*** = 500;

The above declared SERVER\_ERROR\_CODE and its value denotes that an error occurred while loading of the site or the page URL.

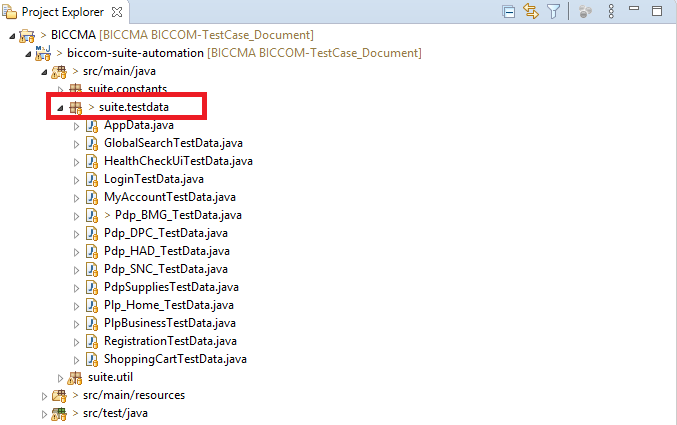


**WebDriverConstants.java**

This class contains browser invoking related constants.

* **For example**: **public static final long MAX\_TIMEOUT = 20**;

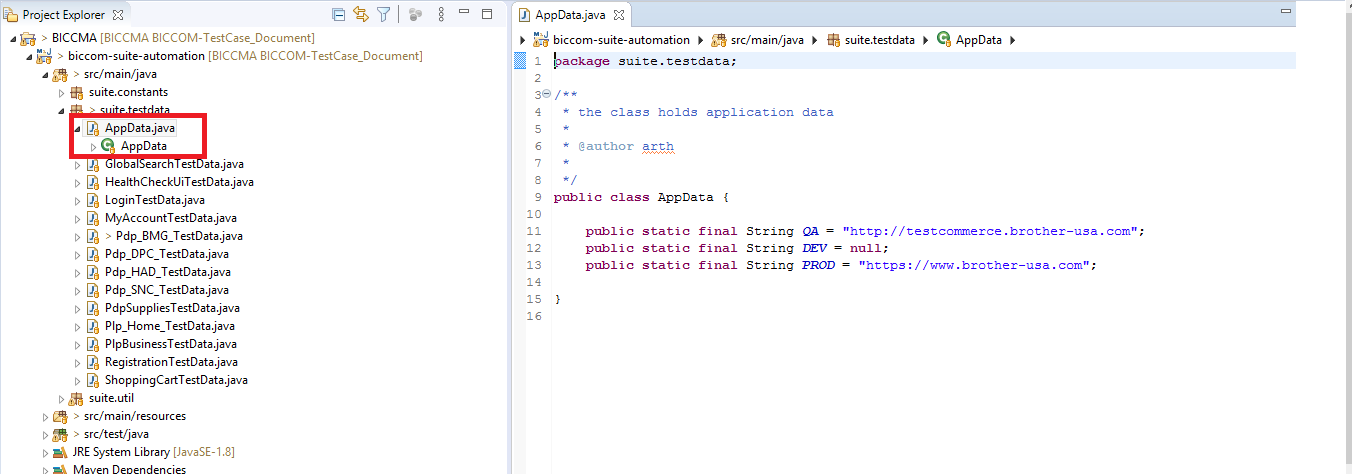
The above declared MAX\_TIMEOUT variable denotes the wait for the page to be completely loaded.



# suite.testdata

Open suite.testdata sub-package. This contains global configuration related to the project. It contains several page specific test-data classes and app test-data class.

**Global configuration related to the project.**

****

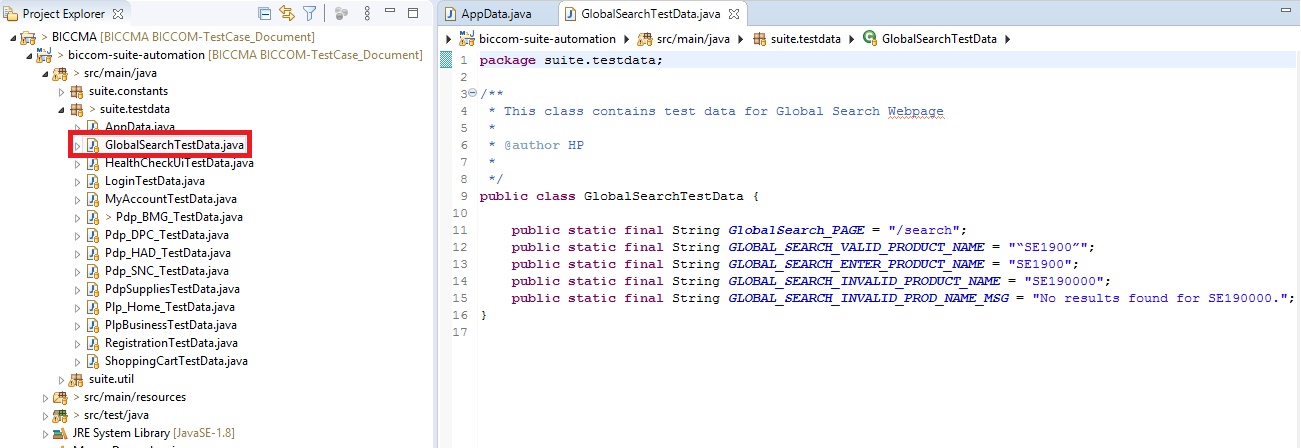
**AppData.java:**

It is the class present within the suite.testdata sub-package. This class contains global configuration related to the project. For example- it contains the base URLs.

* **For example:**

**public static final String QA = "http://testcommerce.brother-usa.com";**

The above declared variable QA, contains the page URL of the test server.

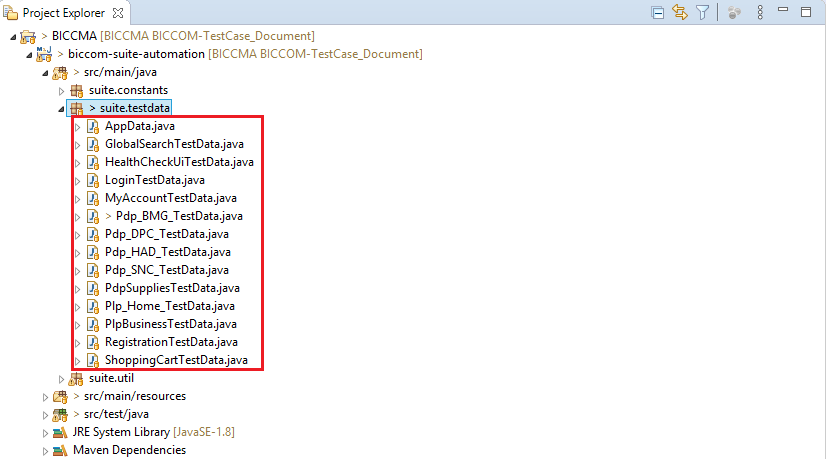
****

**GlobalSearchTestData**

It is the class present inside the suite.testdata sub-package. This class contains page specific test data related to the webpage.

* **For example: public static final String GLOBAL\_SEARCH\_VALID\_PRODUCT\_NAME = "“SE1900”";**

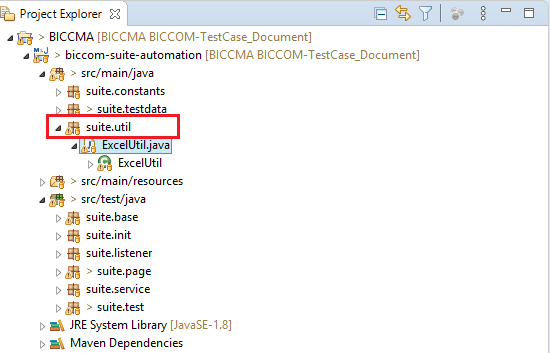
The above declared variable “String GLOBAL\_SEARCH\_VALID\_PRODUCT\_NAME” and its value denotes that the Global Search page contains the product which is denoted by a particular code which is SE1900, in this case.

****

**Test Data classes:**

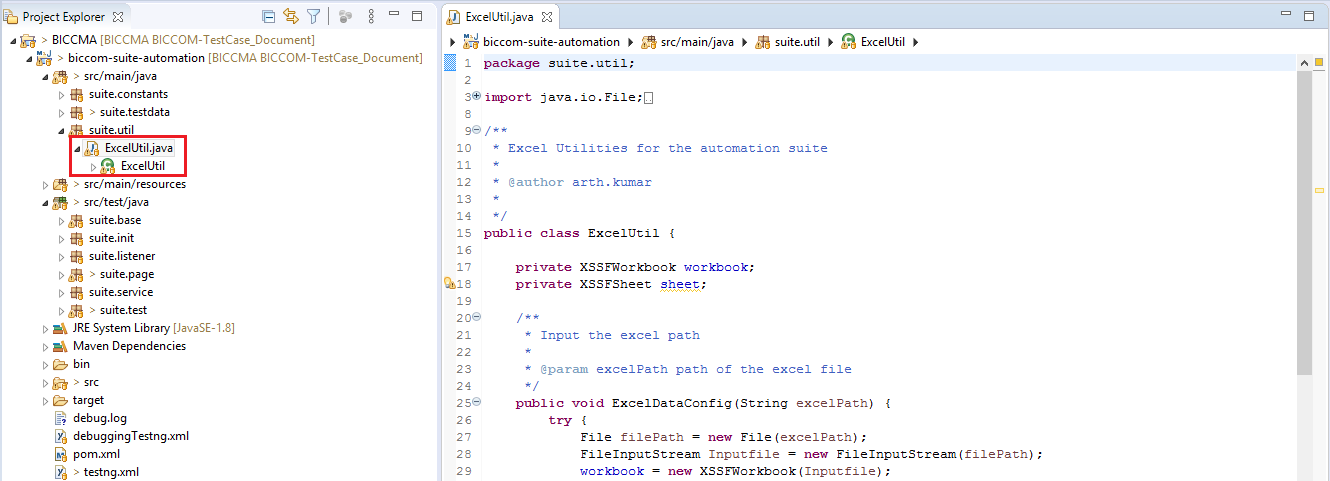
Similarly, there are other test data classes related also contains the page specific test data for the webpage. Some of these classes are:

* HealthCheckUiTestData.java
* LoginTestData.java
* MyAccountTestData.java
* Pdp\_BMG\_TestData.java
* Pdp\_DPC\_TestData.java
* Pdp\_HAD\_TestData.java, etc.



# suite.util

Open suite.util sub-package. This package contains all the common utilities which are project and framework specific.

****

**ExcelUtil.java**

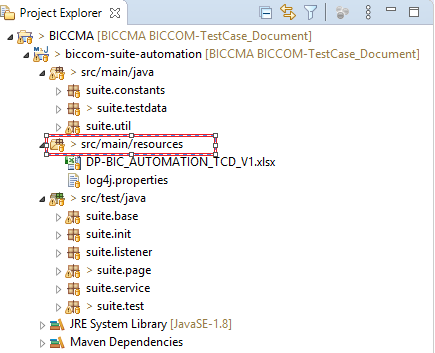
This class resides under suite.util sub-package. This class contains common excel utilities. It contains the common library that can be used in the complete project.

* **For example:**

**public String getCellValue(int sheetNumber, int rowNumber, int columnNumber) {**

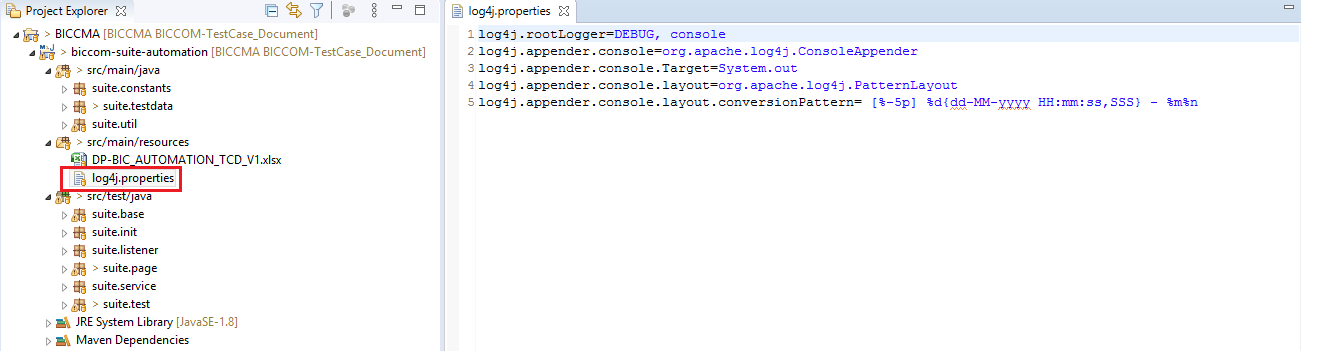
**return workbook.getSheetAt(sheetNumber).getRow(rowNumber).getCell(columnNumber).getStringCellValue();}**

The above method gets the cell value. i.e. serial number, row number, and column number of a particular record stored.

****

# src/main/resources

It contains the app test data files.

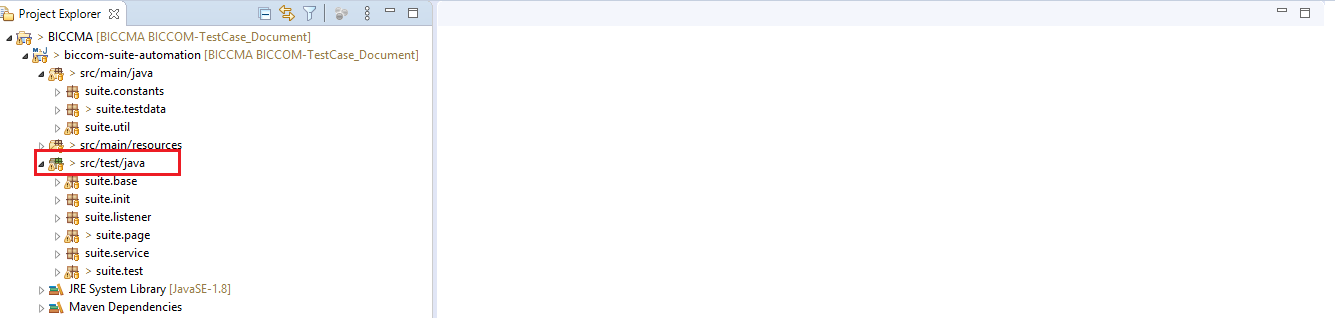
****

# Log4j.properties

It contains configurations of Log4j properties. It is used for logging configurations.

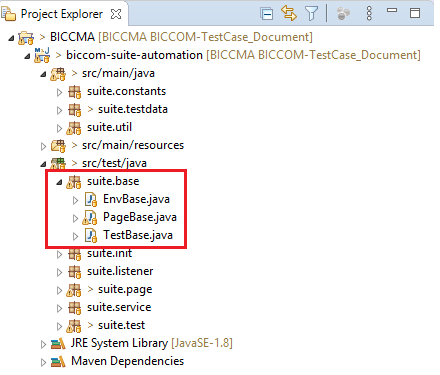
* **For example:** It contains the date format.

**{dd-MM-yyyy HH:mm:ss,SSS}**

****

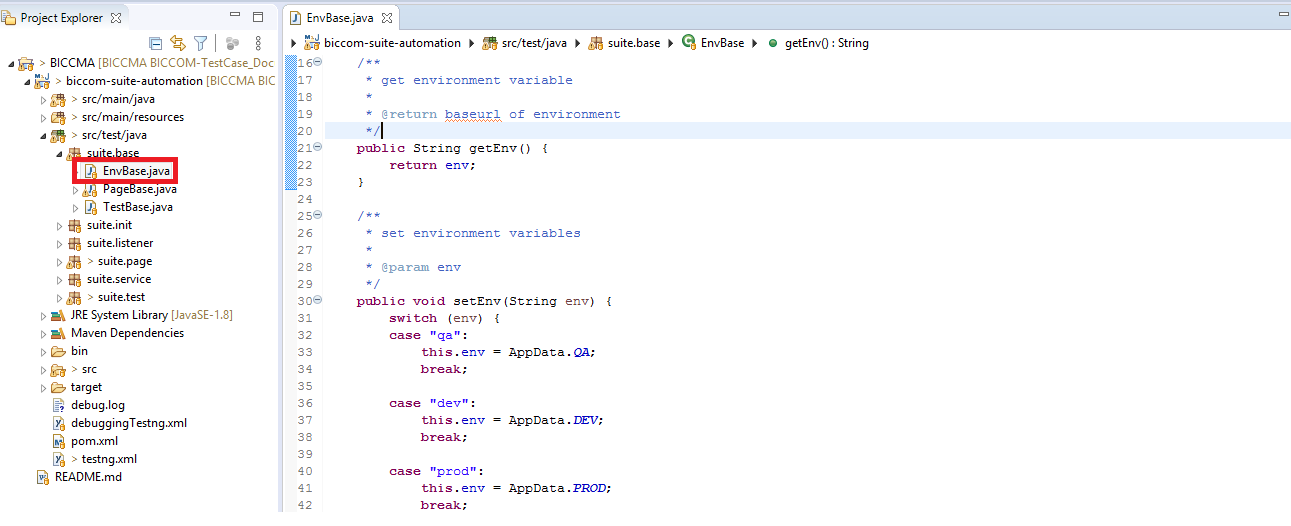
# src/test/java

This sub-package contains the .java files.

****

# suite.base

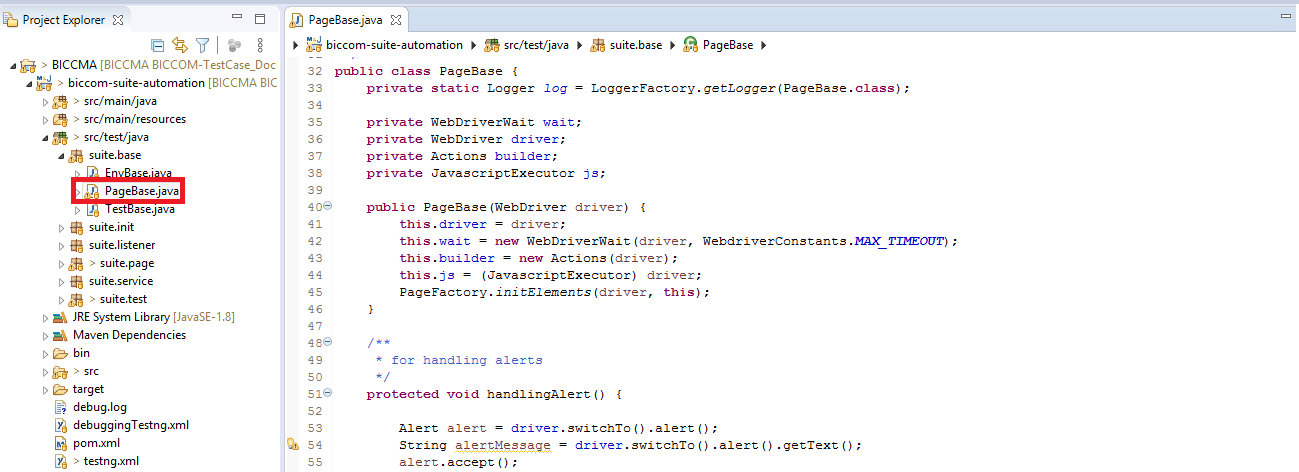
This sub-package contains the base configurations. That is, it contains the base classes.

****

**EnvBase.java**

It contains the environment specific variable such as:

* QA(for test environment)
* PROD(for production)
* DEV(for development)

****

**PageBase.java:**

It contains the base functionalities that is inherited by every page base.

* **For example**:

**protected void handlingAlert() {**

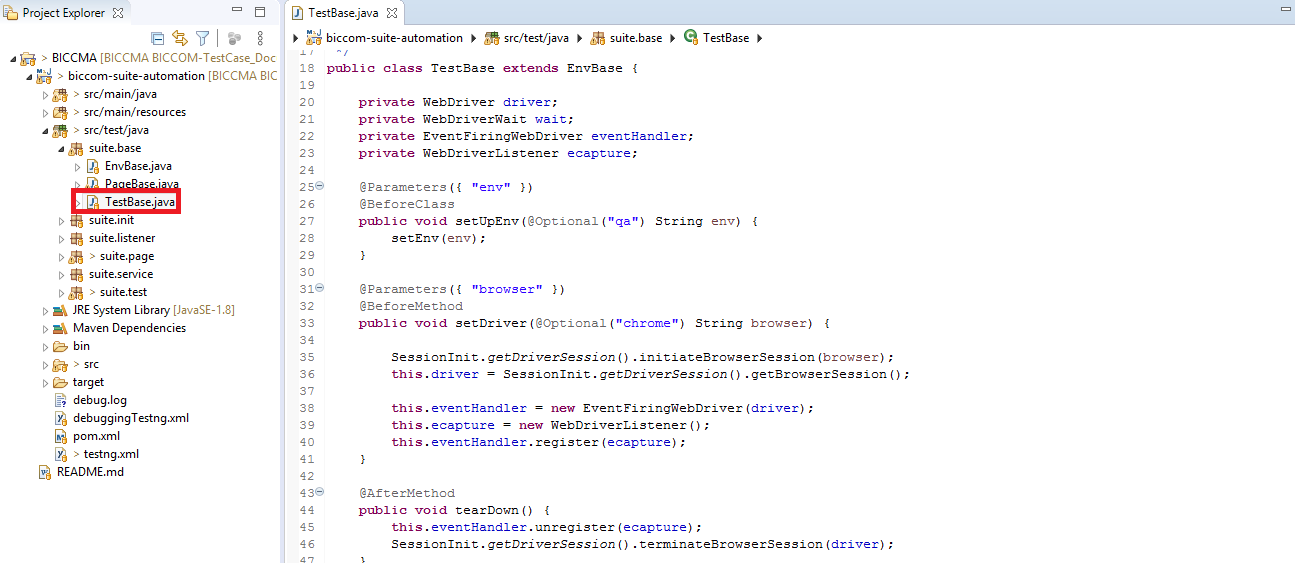
**Alert alert = driver.switchTo().alert();**

**String alertMessage = driver.switchTo().alert().getText();**

**alert.accept();**

**}**

The above method is used me we get a popup which contains accept/allow button and cancel/dismiss buttons. In other words, this popup is also called as an “Alert”. When an alert popup on a page, in order to close or accept the alert, we need to switch from our browser window to the alert window, and then we can either accept or dismiss the alert.



**TestBase.java**

It contains the test configuration, start and termination of the browser session.

* **For example**:

@Parameters({ "env" })

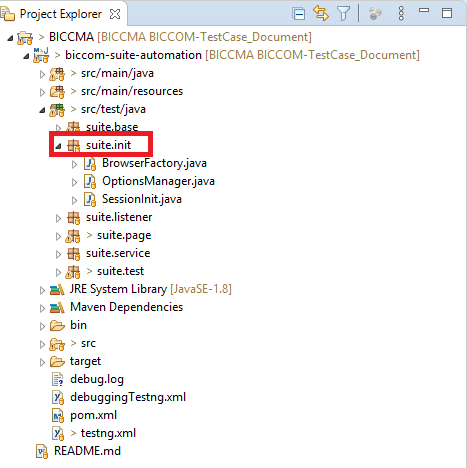
@BeforeClass

**public** **void** setUpEnv(@Optional("qa") String env) {

setEnv(env);

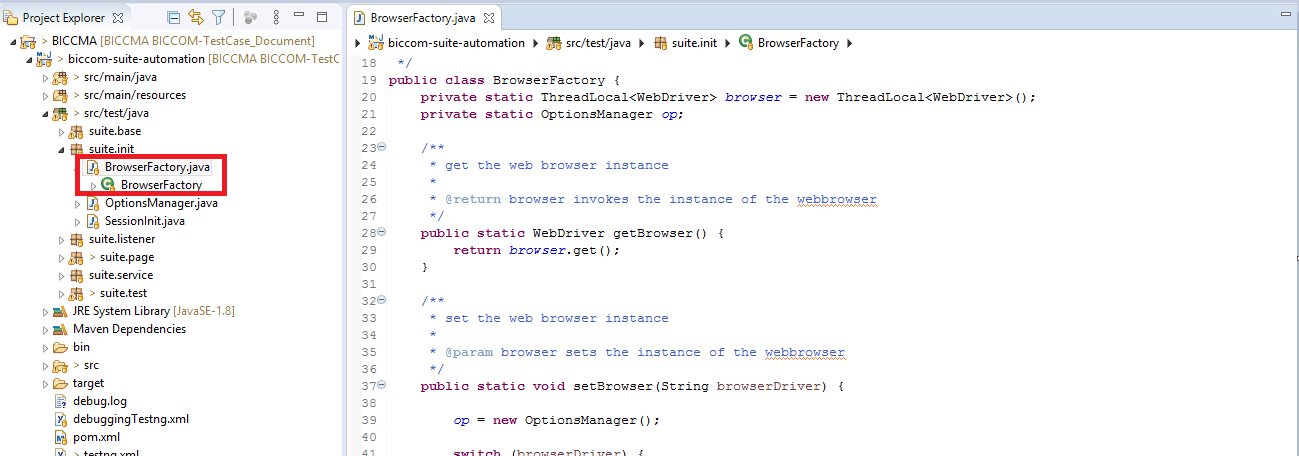
}

This method setup the environment(qa/prod/dev) on which the test cases will be run.



# suite.init

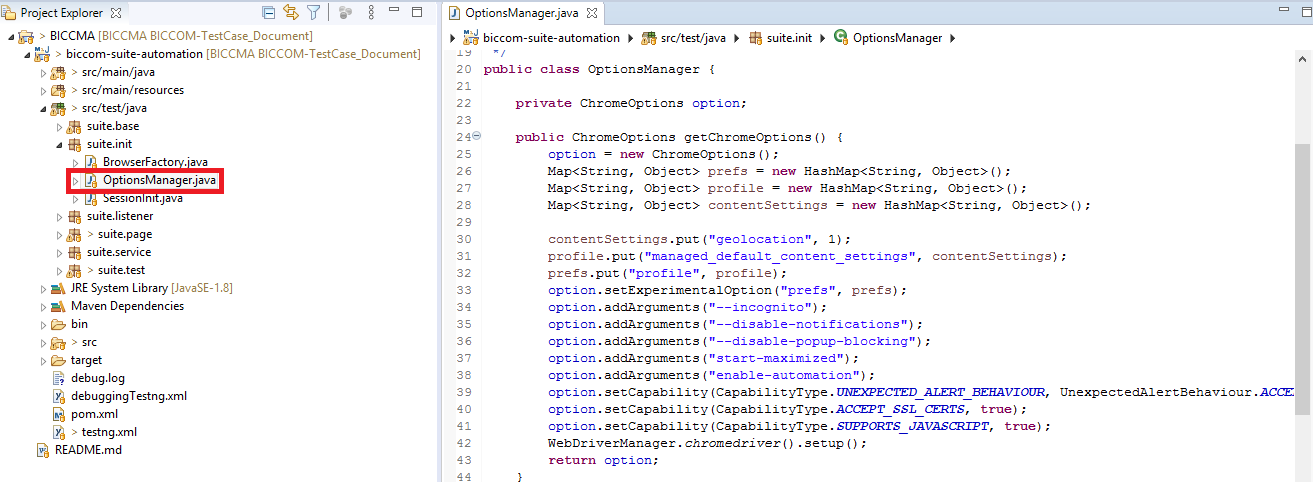
It mainly contains the classes for driver invocation.



**BrowserFactory.java**

This class contains the framework that supports the browser and will provide us the following browser options to run the test cases:

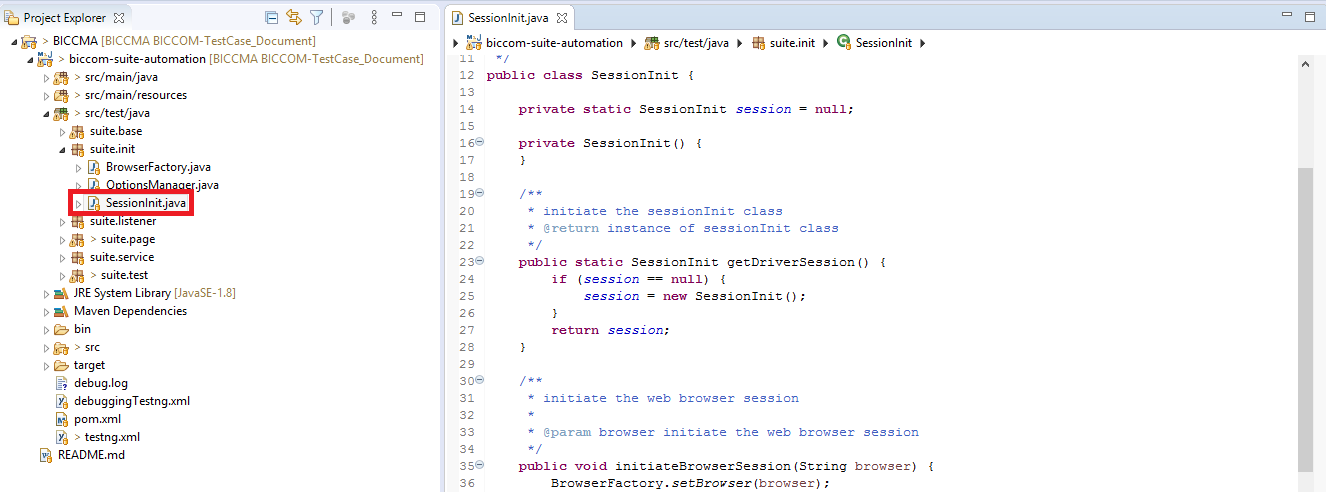
* chrome-ng
* chrome-grid
* chrome-headless



**OptionsManager.java**

This class contains the browser configurations that will invoke the browser with the following options as :

* Incognito, or
* Full-screen, etc.



**SessionInit.java**

This class is used to initiate or terminate the browser sessions.

* **For example**:

**public** **void** initiateBrowserSession(String browser) {

BrowserFactory.*setBrowser*(browser);

}

This method is used to initiate the browser session, in order to run the test cases.

* For example:

**Public void terminateBrowserSession(WebDriver driver) {**

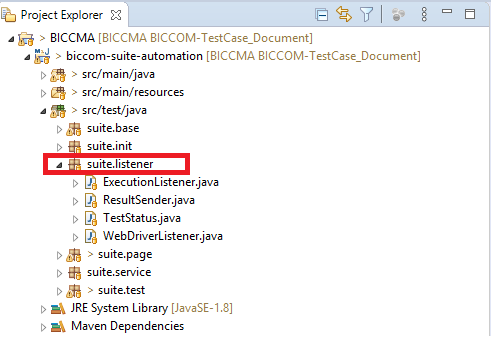
**if (*session* != null) {**

**BrowserFactory.*closeBrowser*(driver);**

***session* = null;**

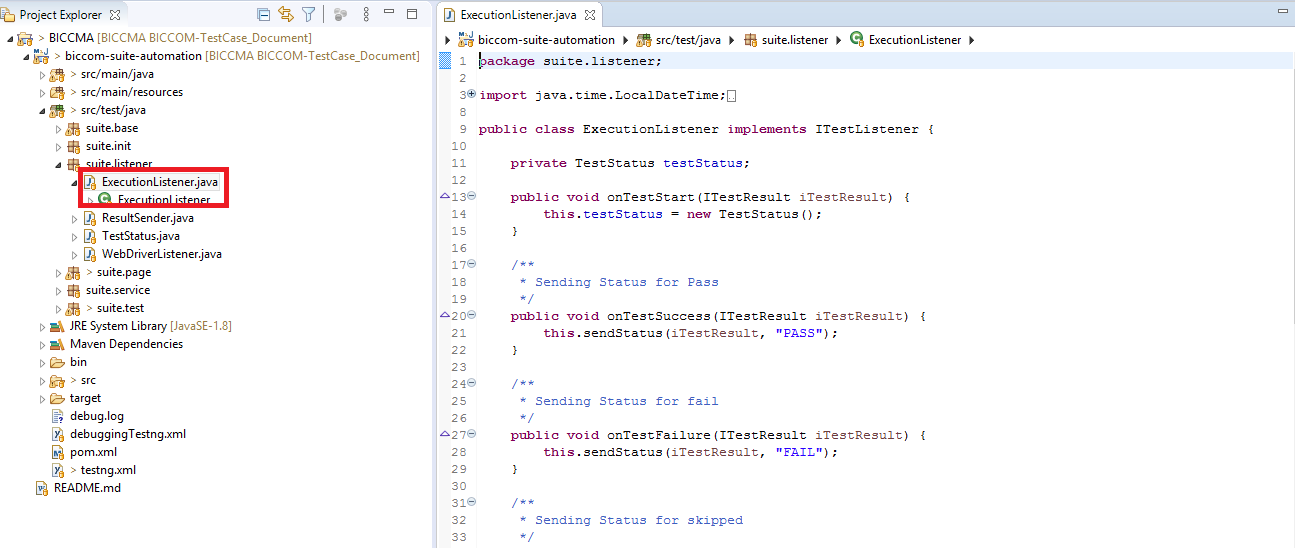
**}**

This method is used to terminate the browser session.



# suite.listener

This package contains the listener configurations, which is used for monitoring, logging, and reporting purposes of test automation suite.



**ExecutionListener.java**

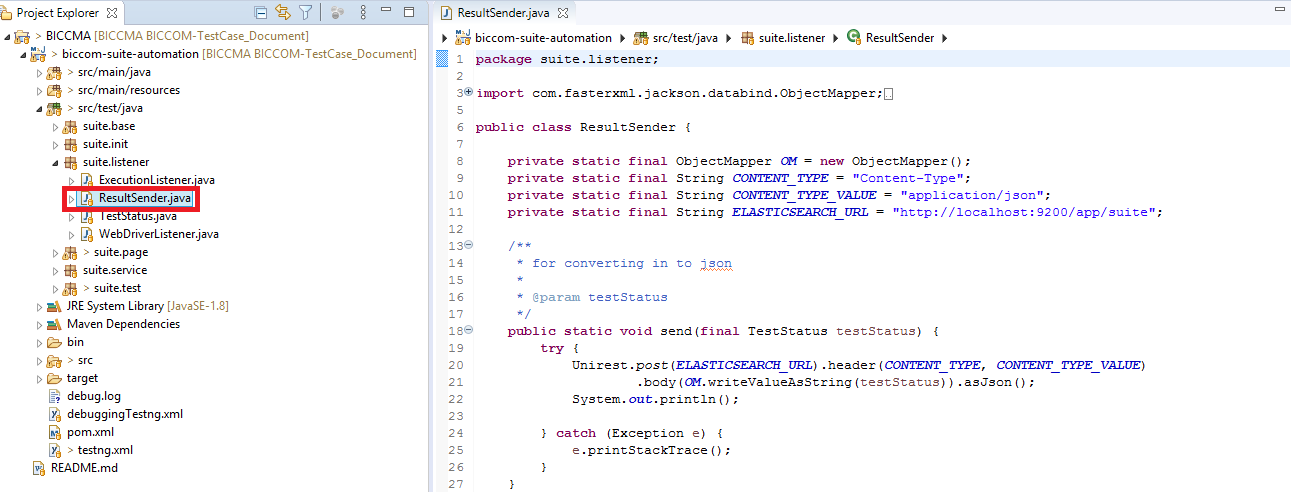
This class manages to send the test result status data into reports, i.e. whether the test cases are passed/fail/skipped.

* **Example**: **public void onTestSuccess(ITestResult iTestResult) {**

**this.sendStatus(iTestResult, "PASS");**

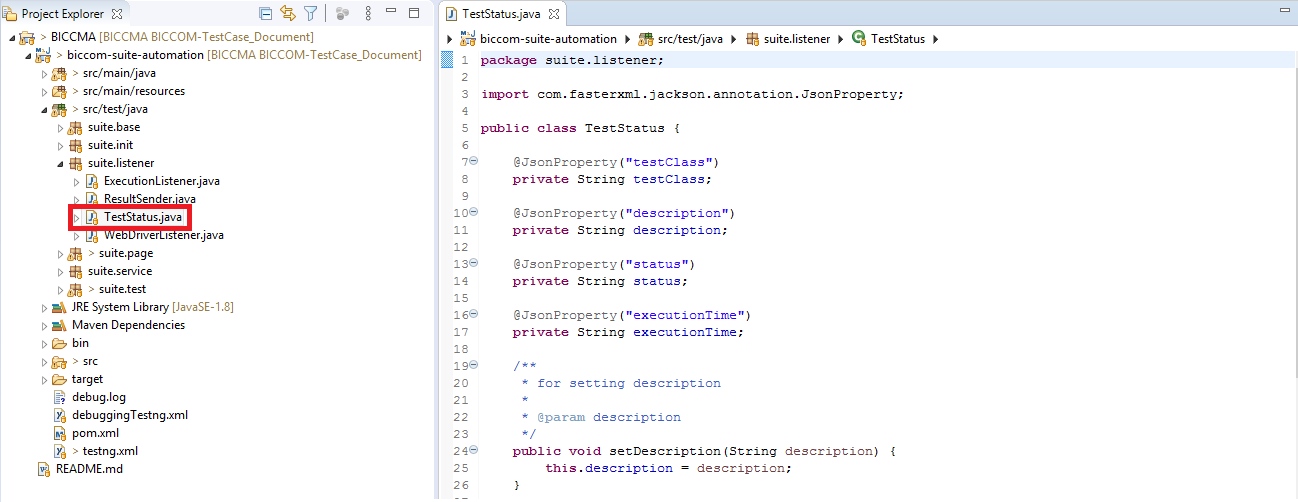
**}**

The above method sends the execution status of a test case as “Pass” only when the test case is successfully executed.



**ResultSender.java**

This class is used to push test execution results (e.g. description, execution time etc) in Elastic Search.



**TestStatus.java**

This class serializes test execution result attributes such as description, execution time etc.

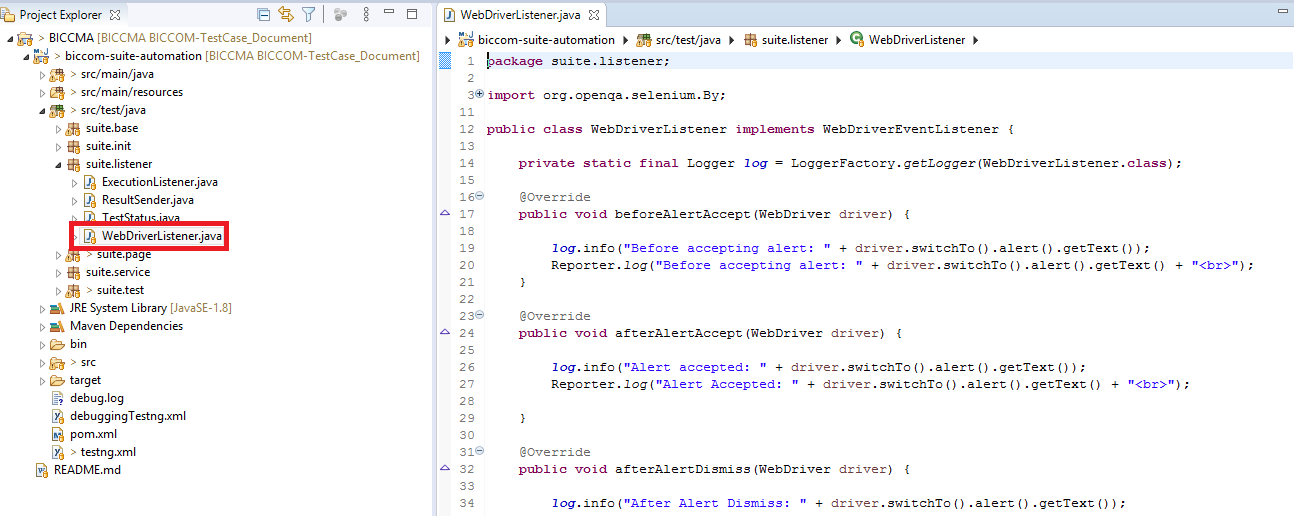
* **For example**:

**public** **void** setExecutionDate(String executionTime) {

**this**.executionTime = executionTime;

}

The above method defines the execution time set for execution of test cases.



**WebDriverListener.java**

This class is used for generation of logs and reports in testing which are based on events/actions.

* **For example**:

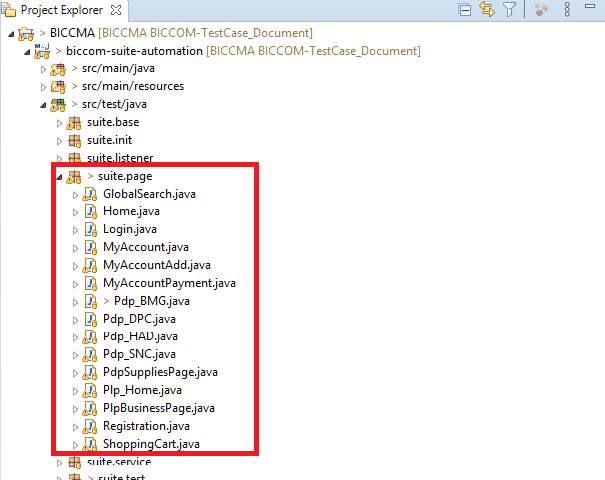
**public** **void** beforeNavigateTo(String url, WebDriver driver) {

***log***.info("Navigating to: " + url);

Reporter.*log*("Navigating to url: " + url + "<br>");

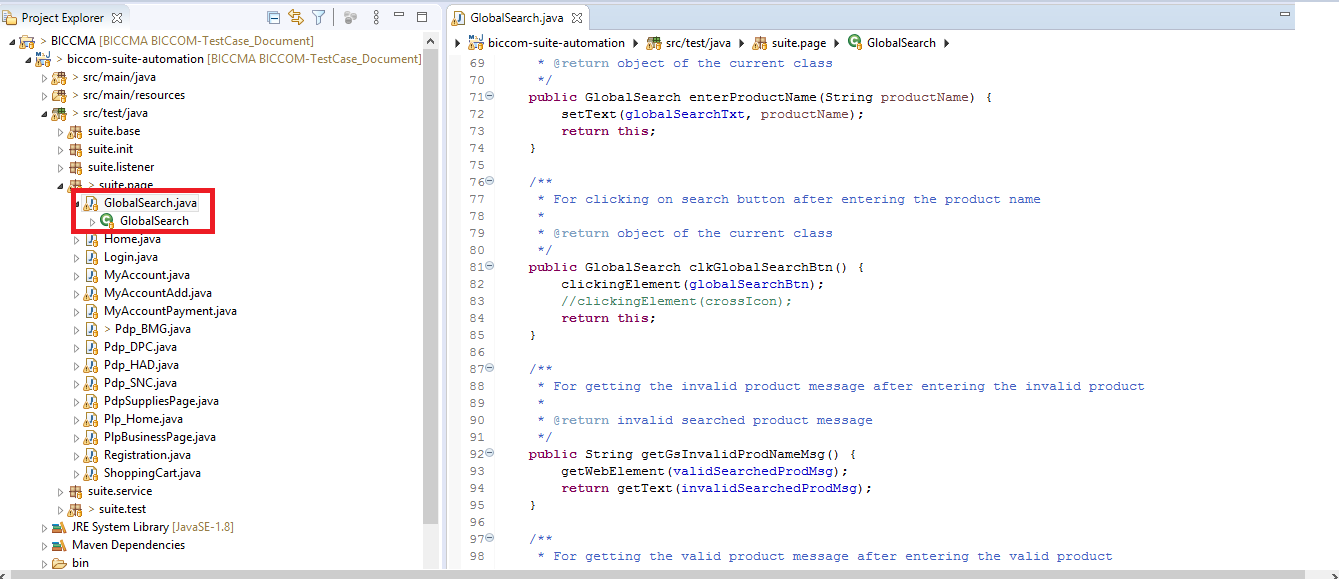
}

The above method generate the logs and reports of the current page before navigating to another page URL.



# suite.page

This sub-package contains the webpage elements and action methods like pressing enter button/ switch to page URL, etc. This sub-package contains ‘.java’ classes.



**GlobalSearch.java**

* This class contains the various locators and methods like clicking on a button, or for getting the page title, or the displayed product name, etc.
* **For example**:

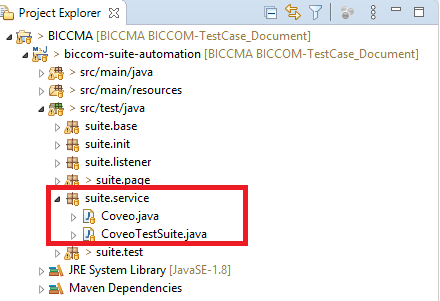
**public** GlobalSearch enterProductName(String productName) {

setText(globalSearchTxt, productName);

**return** **this**;

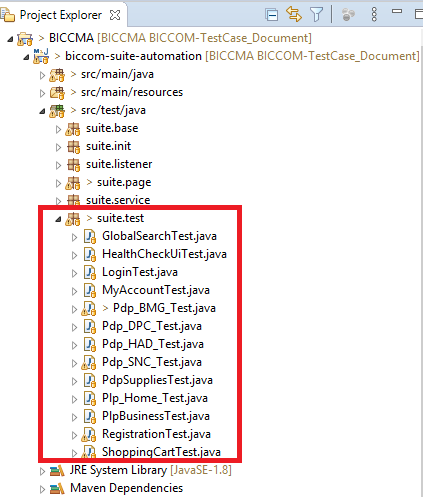
}

The above method is used to enter a product name to be searched for under the Global Search text box.



# suite.service:

This sub-package contains all the API level test cases.

****

# suite.test

This sub-package contains all the automation test cases. It contains all the Test classes.



**GlobalSearchTest.java**

This class contains the Test methods for the Global Search. This class contains the methods defined step by step to perform a particular task, for example: to globally search a product. Similar is the case for other classes defined within the suite.test package.

* **For example**:

@Test(description = "Verify that global search displays the search results when valid product is entered.", groups = {"globalsearch" })

**public** **void** gs\_1() {

glbsch = **new** GlobalSearch(getDriver());

glbsch.getGlobalSearchPage(getPageURL()).

enterProductName(GlobalSearchTestData.***GLOBAL\_SEARCH\_ENTER\_PRODUCT\_NAME***).

clkGlobalSearchBtn();

String actual = glbsch.getGsValidProdNameMsg();

Assert.*assertEquals*(actual,GlobalSearchTestData.***GLOBAL\_SEARCH\_VALID\_PRODUCT\_NAME***);

}

The above test method belongs to the Test class, in which we have automated several execution steps as:

**Step-i:** Get the page URL

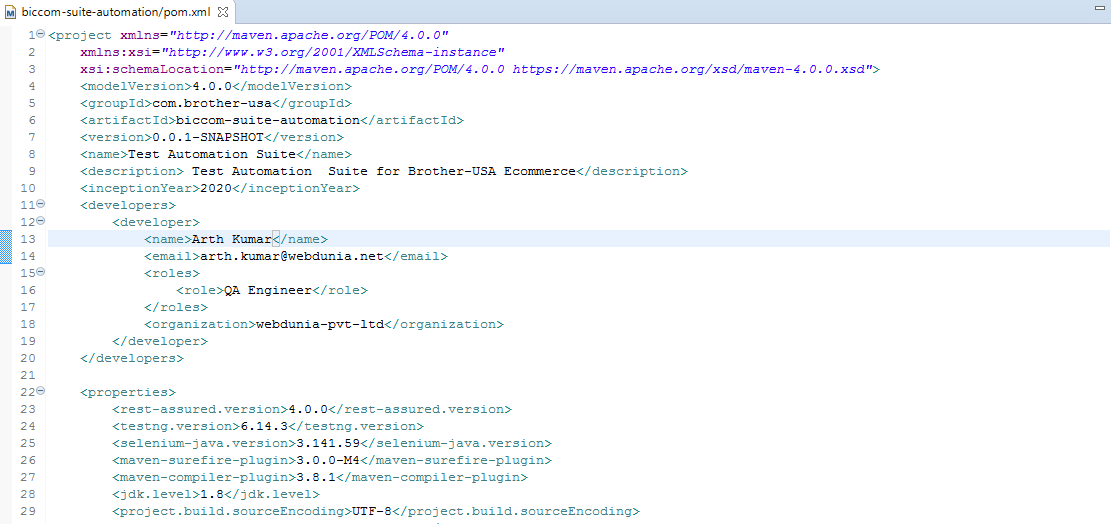
**Step-ii**: Enter the product name in the Global Search text box.

**Step-iii**: Clicking on Global Search Button

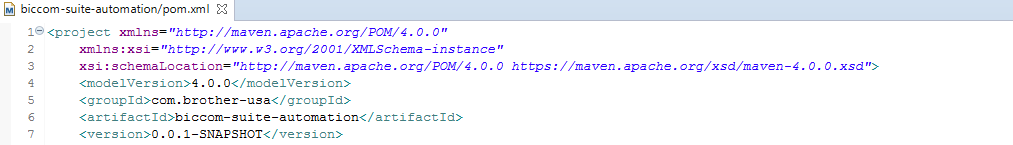
**Step**-iv: Storing the product name displayed after clicking on search button.

**Step**-v: Asserting the product name to match the search results.

**pom.xml**

****

* Any maven project consists of one configurable file called **pom.xml**, which stands for the abbreviation **Project Object Model**.
* This pom.xml will always be located in the root directory of any maven project.
* It contains all the necessary information about the:
* configuration details,
* dependencies included and
* plug-ins included in the project.
* For creating the simple pom.xml file, you need to have following elements:
* **project**: It is the root element of pom.xml file.
* **modelVersion**: It specifies the modelVersion. It should be set to 4.0.0.
* **groupId**:  this is an id which is unique for any project in an organization.
* **artifactId**: It basically defines the name of any project.
* **version:** This element is used to derive the version of any project in order to classify the versions as and when the major changes/implementations are carried during the development phase of a project.
* Following should be the structure of pom.xml for the above mentioned elements:



<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

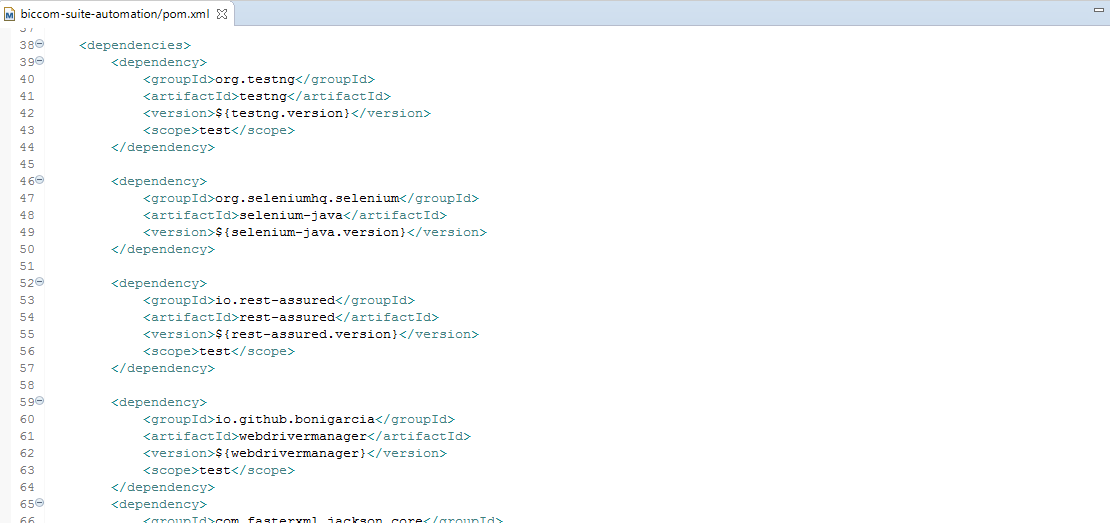
<modelVersion>4.0.0</modelVersion>

<groupId>com.brother-usa</groupId>

<artifactId>biccom-suite-automation</artifactId>

<version>0.0.1-SNAPSHOT</version>

* Dependencies:



* Dependency is another archive—JAR, ZIP, which our current project needs in order to compile, build, test, and/or to run.
* The dependencies are gathered in the pom.xml file, inside the<dependencies> tag.
* When we run a build or execute a maven goal, these dependencies are resolved, and are then loaded from the local repository.
* If they are not present there, then Maven will download them from a remote repository and store them in the local repository.
* We can also manually install the dependencies.