

Quick and Simple Selenium Automation framework setup guide



Contents:

1. Downloading the repository
2. Importing project into Eclipse
3. Framework structure overview
4. Additional downloads
5. Setting up Grid
6. Setting up CI (Jenkins)
7. Running tests

Purpose:

1. Help QA Engineers get basic idea of automation framework structure
2. Provide some useful examples for future use
3. Help understand that automation can be easy and fun

1. Downloading the framework

Step I:

Go to: <https://github.com/bbatsalenka/automation-starter-kit>

Step II:

Download ZIP as shown on the screenshot:

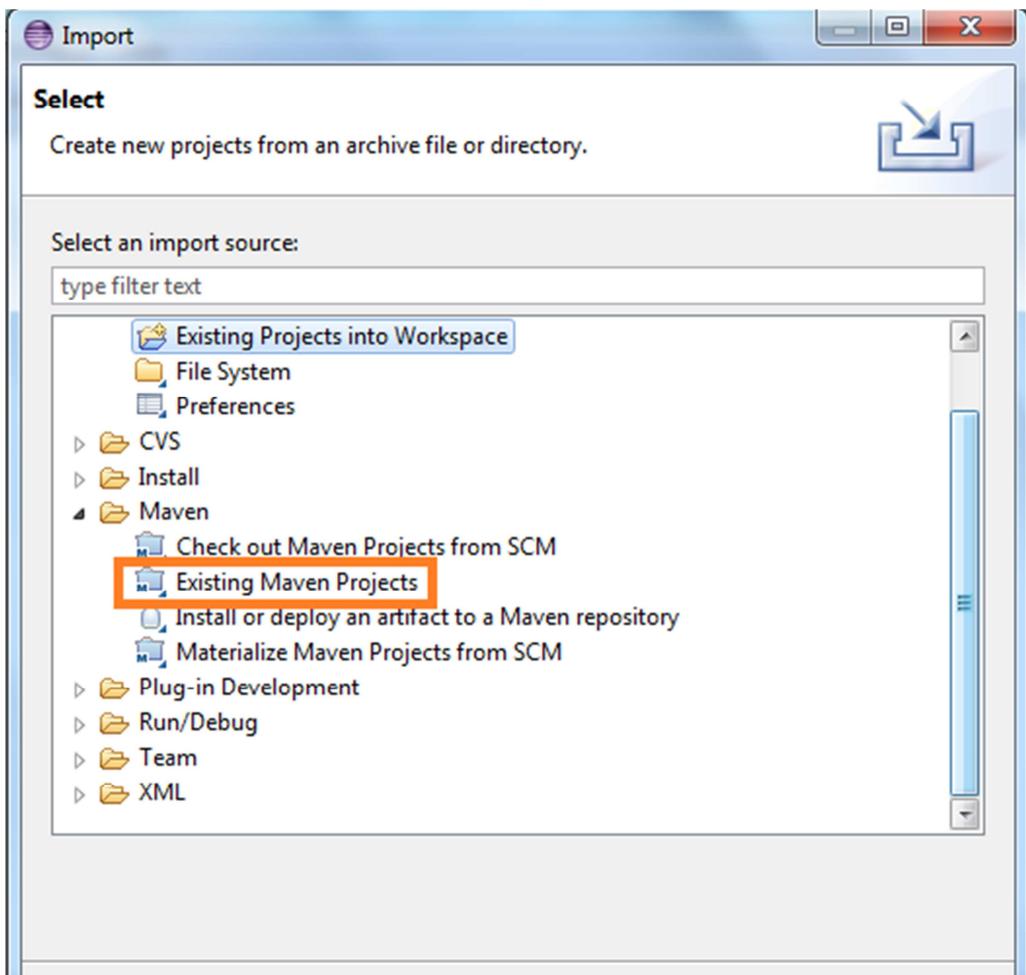
The screenshot shows a GitHub repository page for 'bbatsalenka/automation-starter-kit'. On the left, there's a list of files with their first commit details. A yellow arrow points from the 'src' file entry to the 'Download ZIP' button on the right. The right side features a sidebar with network and settings links, an HTTPS clone URL, a 'Clone in Desktop' button, and the highlighted 'Download ZIP' button.

File	First Commit	Time Ago
screenshots	first commit	37 minutes ago
src	first commit	37 minutes ago
target	first commit	37 minutes ago
.classpath	first commit	37 minutes ago
.project	first commit	37 minutes ago
chrome_tests.xml	first commit	37 minutes ago
environment.properties	first commit	37 minutes ago
firefox_tests.xml	first commit	37 minutes ago
ie_tests.xml	first commit	37 minutes ago
pom.xml	first commit	37 minutes ago
user.properties	first commit	37 minutes ago

We recommend adding a README to this repository to help give people an overview of your project. [Add a README](#)

2. Importing project into Eclipse

Step I: Import downloaded framework as “Existing Maven Projects” (see screenshot) in Eclipse



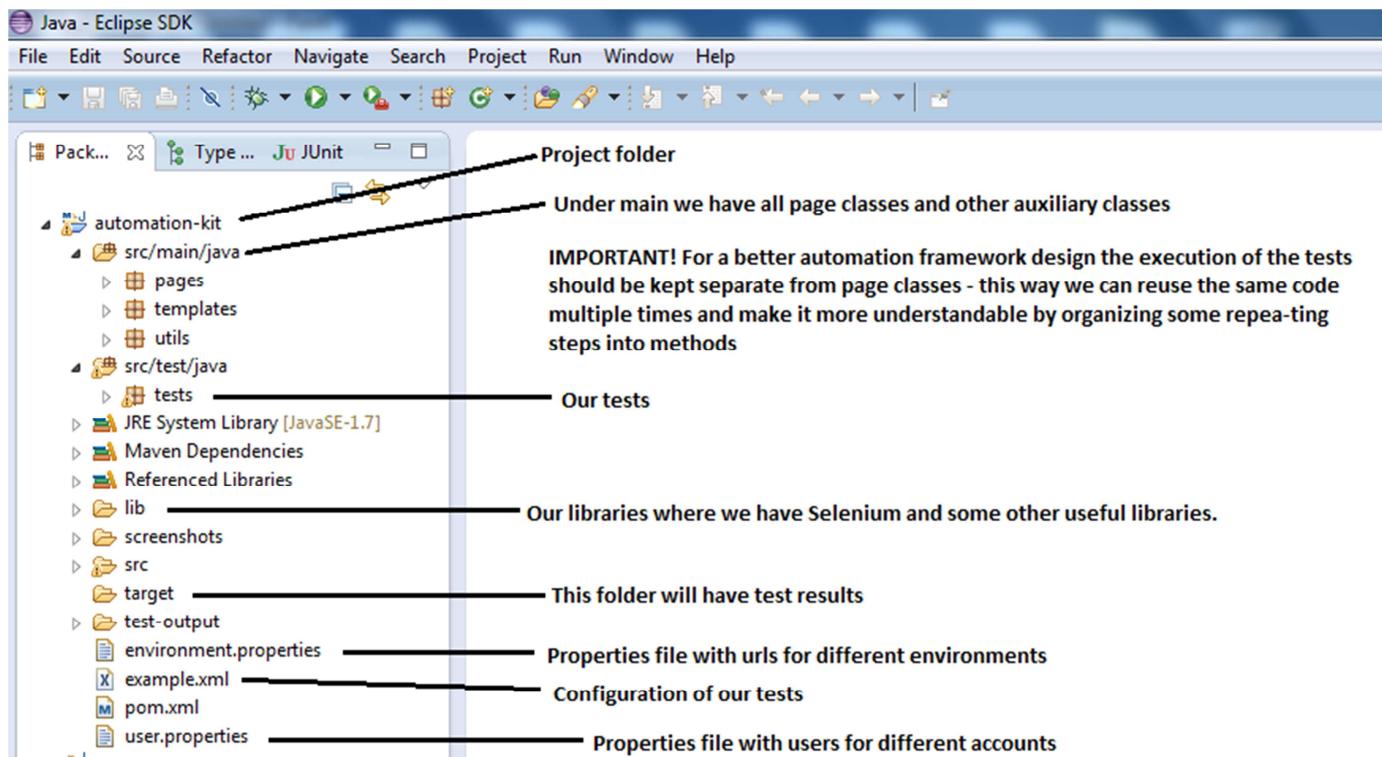
Step II: if there are issues in the project related to java 1.7 (as on the screenshot), use the solution provided by Eclipse – Change project

```
public static DesiredCapabilities getBrowserInstance(String browserName) {
    switch (browserName) {
        case "fir":
            Desire
        Firefox
        capab
        return
    }
    case "ie":
        System
        Desire
        return
    }
    case "ch":
        Desire
        System
        return capability;
}
default: {
    DesiredCapabilities capability = DesiredCapabilities.firefox();
    return capability;
}
```

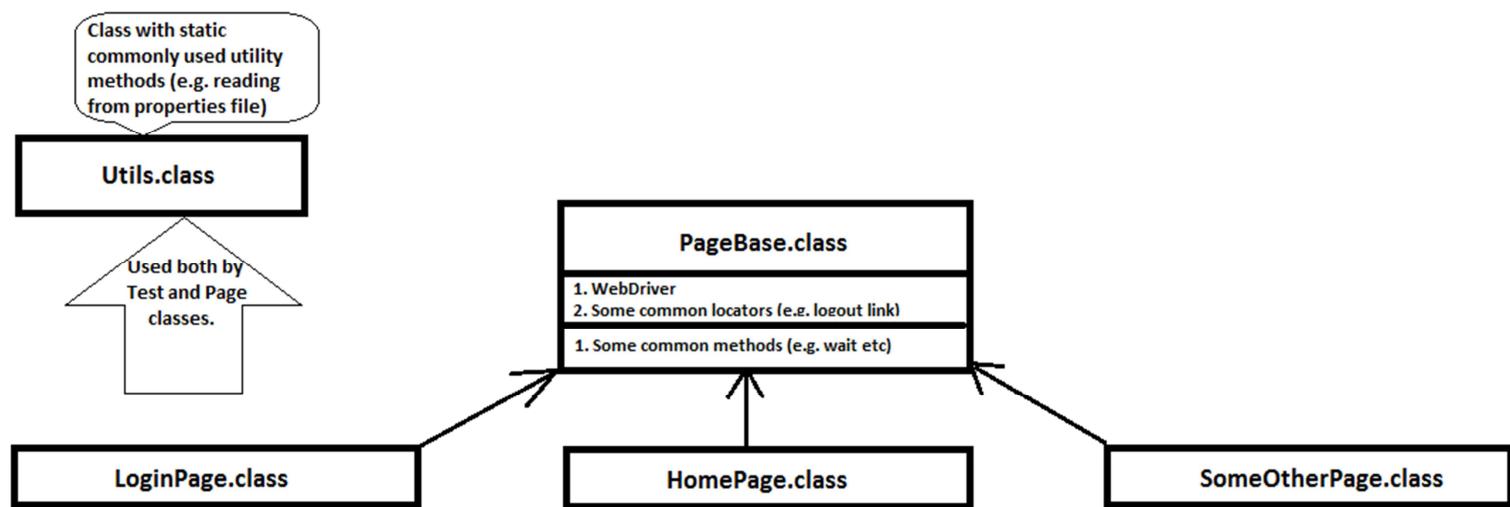
A screenshot of the Eclipse IDE showing a Java code editor for a file named 'Utils.java'. The code defines a static method 'getBrowserInstance' that takes a string parameter 'browserName' and returns a 'DesiredCapabilities' object. It uses a switch statement to handle three cases: 'fir', 'ie', and 'ch'. The 'fir' case contains incomplete code for 'Firefox' and 'capab'. The 'ie' case contains incomplete code for 'System' and 'Desire'. The 'ch' case contains incomplete code for 'Desire' and 'System'. The 'default' block sets 'capability' to a 'DesiredCapabilities' object created with 'firefox()' and then returns it. A context menu is open at the start of the 'switch' block. The 'Change project compliance and JRE to 1.7' option is highlighted with a red box. Other visible options in the menu include 'Rename in file (Ctrl+2, R)' and 'Rename in workspace (Alt+Shift+R)'. A tooltip on the right side of the menu explains: 'Set project compiler compliance settings to 1.7' and 'Set project JRE build path entry to 'JavaSE-1.7''. At the bottom of the code editor, a message says 'Press 'Tab' from proposal table or click for focus'.

compliance and JRE to 1.7

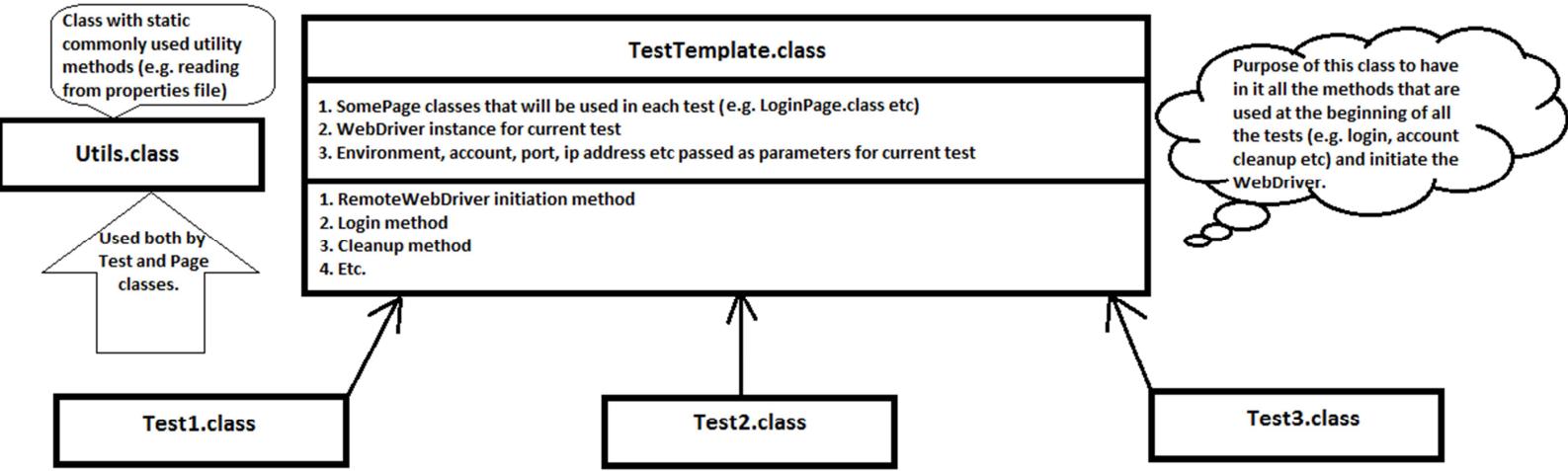
3. Framework structure overview



Below is inheritance diagram of the framework Page classes.



Here is the diagram of the Test classes hierarchy:



Each page extends the PageBase.class that has some common locators and methods in it (e.g. logout or terms of use link, waitForPageLoaded() method etc.).

```
1 package pages;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.WebElement;
5 import org.openqa.selenium.support.FindBy;
6 import org.openqa.selenium.support.PageFactory;
7
8 // each page extends this class that has some common methods and WebElements
9 public abstract class PageBase {
10
11     protected WebDriver driver;
12     protected static final int DEFAULT_WAIT_4_ELEMENT = 25;
13     protected static final int DEFAULT_WAIT_4_PAGE = 30;
14
15     // PageBase constructor that initiates WebElement
16     public PageBase(WebDriver driver) {
17         this.driver = driver;
18         PageFactory.initElements(driver, this);
19     }
20 }
```

Here is an example of LoginPage.class that extends PageBase.class:

Screenshot of an IDE showing the LoginPage.java code. The code defines a class that extends PageBase and initializes WebElements using PageFactory.

```

1 package pages;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.WebElement;
5 import org.openqa.selenium.support.FindBy;
6 import org.openqa.selenium.support.PageFactory;
7
8 public class LoginPage extends PageBase {
9     public LoginPage(WebDriver driver) {
10         super(driver);
11         PageFactory.initElements(driver, this);
12     }
13
14     @FindBy(name = "userid")
15     private WebElement userNameInputField;
16
17     @FindBy(name = "pswrd")
18     private WebElement passwordInputField;
19
20     @FindBy(xpath = "//input[@value='Login']")
21     private WebElement loginButton;
22
23     @FindBy(xpath = "//input[@value='Cancel']")
24     private WebElement cancelButton;
25 }

```

Annotations and comments:

- Line 1: Not visible here: LoginPage.class also has login method which is used by multiple tests for login and accepts 2 parameters for login.
- Line 8: LoginPage.class extends PageBase.class to have access to it's WebElements and methods
- Line 11: In the constructor WebDriver is passed to PageBase.class to instantiate it's WebElements and also LoginPage.class elements are instantiated in it's constructor method.
- Line 14-15: WebElements that are located on LoginPage. Notice: Terms of Use link is not among it's elements as it's PageBase.class element - good example of reducing code.
- Line 26: * There is no need to add all the WebElements from the web page to the Page.class- only the ones that will be potentially used in the tests should be added.

To store WebElements as class fields we use PageFactory (the elements are usually initialized in the constructor as is shown on the screenshot above)

```

22     PageFactory.initElements(driver, this);
23 }
24
25 // it makes sense to place this element in PageBase.class as it appears on
26 // all pages that extend PageBase.class
27     @FindBy(linkText = "Terms of Use")
28     private WebElement termsOfUseLink;
29
30     @FindBy(linkText = "Logout")
31     private WebElement logoutLink;
32
33     @FindBy(linkText = "Back to load testing")
34     private WebElement backToLoadTestingLink;
35

```

Annotation:

PageFactory locators

To structure each test and supply some parameters to it, like: environment, user credentials, browser parameters, parameters for the Grid we are using .xml format TestNG file:

```

example.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <suite name="Firefox-Tests" parallel="tests" preserve-order="true" thread-count="4" verbose="5">
3   <listeners>
4     <listener class-name="packageName.ScreenshotOnFailure" />
5   </listeners>
6 <test name="Example"> ━━━━ Test name
7   <parameters>
8     <parameter name="browser" value="firefox" /> ━━━━ Browser parameters
9     <parameter name="environment" value="stage" /> ━━━━ Some other parameters as
10    <parameter name="user" value="stage_11" /> ━━━━ testing environment, users, etc.
11    <parameter name="port" value="5555" />
12    <parameter name="ipAddress" value="localhost" /> ━━━━ Node parameters for Selenium Grid (localhost is
13                               used if the local machine both a hub and a node,
14                               otherwise remote machine ip should be used)
15   </parameters>
16   <classes>
17     <class name="packageName.ExampleTest" /> ━━━━ Class name with
18   </classes>                                         specific tests
19 </test>
</suite> <!-- Suite -->

```

We add each class that has specific tests in it to .xml tests configuration file as a separate `<test>`. Each .xml configuration file has a `ScreenshotOnFailure` listener in it that takes screenshots when a test failure occurs.

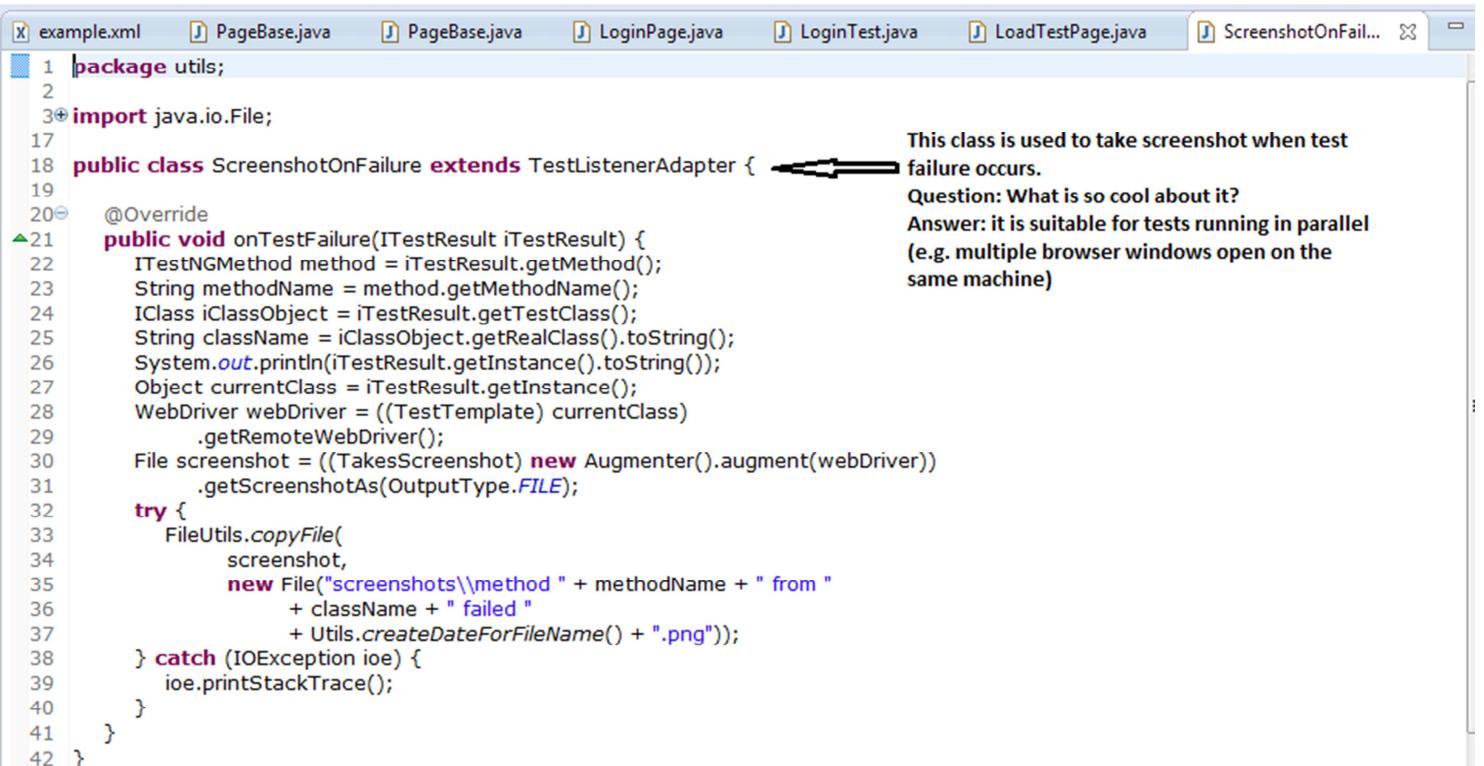
Note: if structure of the framework is followed the `ScreenshotOnFailure` listener works perfectly fine for tests running in parallel and takes screenshot for the right browser when a failure occurs.

```

example.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <suite name="Firefox-Tests" parallel="tests" preserve-order="true" thread-count="4" verbose="5">
3   <listeners>
4     <listener class-name="packageName.ScreenshotOnFailure" /> ━━━━ Listener is one per
5   </listeners>                                         suite
6 <test name="Test I"> ━━━━ Insert each test class with
7   <parameters>                                         specific tests as is shown
8     <parameter name="someParameter" value="someParameter" />
9   </parameters>
10  <classes>
11    <class name="packageName.ExampleTestI" />
12  </classes>
13 </test>
14 <test name="Test II"> ━━━━
15   <parameters>
16     <parameter name="someParameter" value="someParameter" />
17   </parameters>
18   <classes>
19     <class name="packageName.ExampleTestII" />
20   </classes>
21 </test>
22 </suite> <!-- Suite -->

```

Below is the ScreenshotOnFailure.class declaration. When a screenshot is taken it has its name as the method in which the failure occurred with date and time.



```
1 package utils;
2
3+ import java.io.File;
4
5 public class ScreenshotOnFailure extends TestListenerAdapter {
6     // ...
7
8     @Override
9     public void onTestFailure(ITestResult iTestResult) {
10         ITestNGMethod method = iTestResult.getMethod();
11         String methodName = method.getMethodName();
12         IClass iClassObject = iTestResult.getTestClass();
13         String className = iClassObject.getRealClass().toString();
14         System.out.println(iTestResult.getInstance().toString());
15         Object currentClass = iTestResult.getInstance();
16         WebDriver webDriver = ((TestTemplate) currentClass)
17             .getRemoteWebDriver();
18         File screenshot = ((TakesScreenshot) new Augmenter().augment(webDriver))
19             .getScreenshotAs(OutputType.FILE);
20         try {
21             FileUtils.copyFile(
22                 screenshot,
23                 new File("screenshots\\method " + methodName + " from "
24                     + className + " failed "
25                     + Utils.createDateForFileName() + ".png"));
26         } catch (IOException ioe) {
27             ioe.printStackTrace();
28         }
29     }
30 }
```

This class is used to take screenshot when test failure occurs.
Question: What is so cool about it?
Answer: it is suitable for tests running in parallel (e.g. multiple browser windows open on the same machine)

To run the tests from CI tool like Jenkins we will need to provide Maven commands to it, like:

-Dtests=ie_tests.xml -Dreceiver=youremail@domain.com test

ie_tests.xml – is the .xml file with specific tests that you want to run.

For this we need to configure our pom.xml file (it's already configured and here is just a brief overview):

We supply .xml file names to maven-surefire-plugin to run all the tests contained in this .xml file

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.15</version>
<configuration>
  <testFailureIgnore>true</testFailureIgnore>
  <suiteXmlFiles>
    <suiteXmlFile>${tests}</suiteXmlFile>
  </suiteXmlFiles>
</configuration>
</plugin>
```

We are using maven-postman-plugin to send tests-run report and screenshots (if any) to the specified email address.

Note: you will need to change the “From” property to the email address from which you want to receive emails.

```
<plugin>
<groupId>ch.fortysix</groupId>
<artifactId>maven-postman-plugin</artifactId>
<version>0.1.6</version>
<executions>
  <execution>
    <id>send_an_email</id>
    <phase>test</phase>
    <goals>
      <goal>send-mail</goal>
    </goals>
    <configuration>
      <mailhost>smtp.gmail.com</mailhost>
      <mailport>465</mailport>
      <mailssl>true</mailssl>
      <mailAltConfig>true</mailAltConfig>
      <mailuser>bagditester@gmail.com</mailuser>
      <mailpassword>████████</mailpassword>
      <from>bagditester@gmail.com</from>
      <receivers>
        <receiver>${receiver}</receiver>
      </receivers>
      <fileSets>
        <fileSet>
          <directory>${basedir}/target/surefire-reports</directory>
          <includes>
            <include>emailable-report.html</include>
```

This you will need to change for your email address

This property you will supply as part of the maven command

```

<includes>
    <include>emailable-report.html</include> ← Name of the report
</includes>
</fileSet>
<fileSet>
    <directory>${basedir}/screenshots</directory> ← Screenshots location
    <includes>
        <include>**/*.{png}</include>
    </includes>
</fileSet>
</fileSets>
<subject>Important subject</subject> ← Email subject
<failonerror>true</failonerror>
<htmlMessage>
<![CDATA[
<p>Attached is tests run report.</p>
<br>
<p>Have a nice day.</p>
]]>
</htmlMessage>
</configuration>
</execution>
</executions>
</plugin>

```

In our tests we also use .properties files to store parameters for our tests (e.g. "environment"="stage", "user"="testuser", "password"="password") – the advantage of it is just to supply a value to the method `Utils.getValueFromPropertiesFile(String variable, String filename)` and the method will return the value of requested variable.



4. Additional downloads

Additionally you will need to download the following:

- i. Selenium-standalone-server
- ii. Chromedriver.exe
- iii. IEDriverServer.exe
- iv. Jenkins CI tool

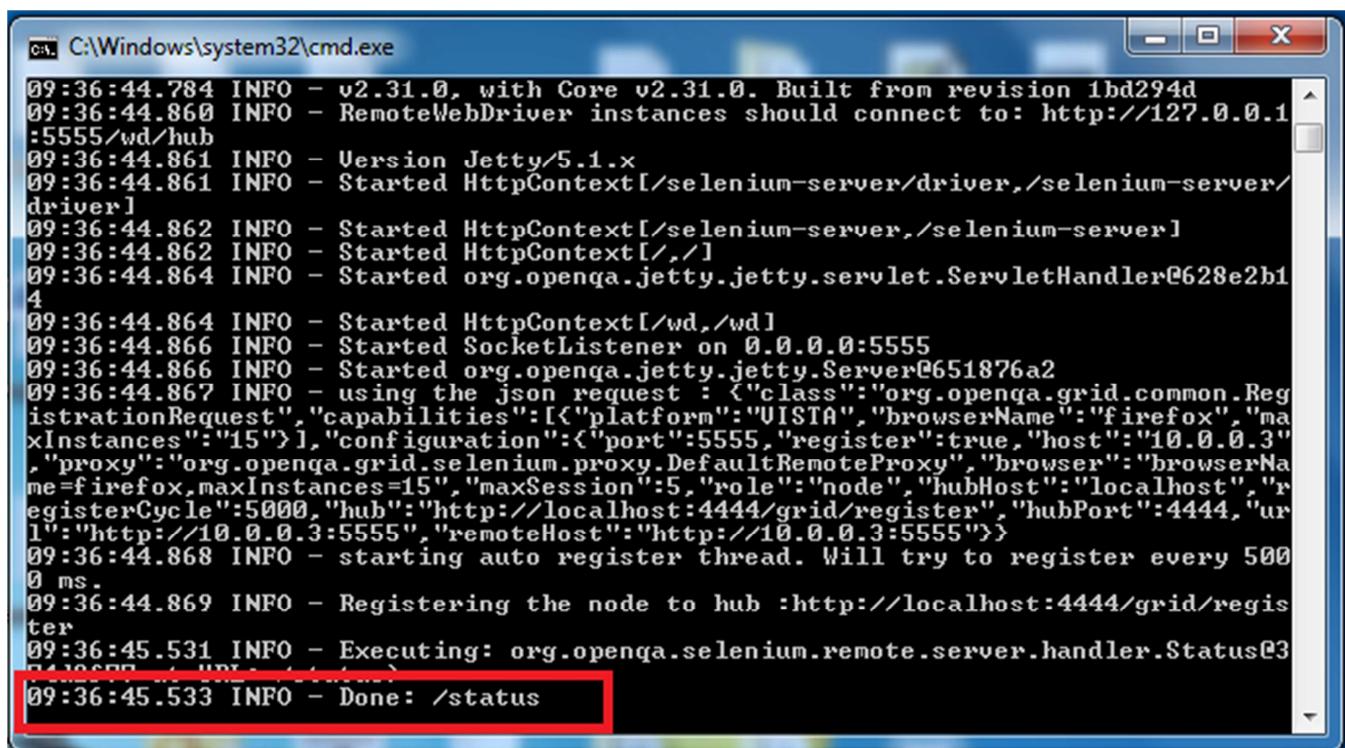
5. Setting up Grid:

Since you will have the hub of your Grid integrated with your Jenkins CI you will need only to start the nodes either on your local machine or on remote ones so that you could run tests on them.

To configure the node you will need to create a .cmd file with the following command in it:

```
cd to the location of selenium-server-standalone-version.jar  
java -jar selenium-server-standalone-2.31.0.jar -role node -hub  
http://localhost:4444/grid/register  
\ -browser browserName=firefox,platform=WINDOWS  
\ -browser browserName=internet  
explorer,version=9,platform=WINDOWS  
\ -browser browserName=chrome,platform=WINDOWS  
If you are not using any nodes, then the ip address or hostname  
should be the one of local machine (you do not need to start the hub  
as it's a part of Selenium Grid plugin on Jenkins – the setup will be  
explained further)
```

After you start the node, make sure your node is connected to the hub, look for the following line:



The screenshot shows a Windows Command Prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The window contains a log of information from a Selenium node. The log includes several 'INFO' messages indicating the node's startup, connection to a hub at 'http://127.0.0.1:5555/wd/hub', and its registration with the hub. A specific line of interest is highlighted with a red rectangle: '09:36:45.533 INFO - Done: /status'. This line indicates that the node has successfully registered with the hub.

```
09:36:44.784 INFO - v2.31.0, with Core v2.31.0. Built from revision 1bd294d
09:36:44.860 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:5555/wd/hub
09:36:44.861 INFO - Version Jetty/5.1.x
09:36:44.861 INFO - Started HttpContext[/selenium-server/driver,/selenium-server/driver]
09:36:44.862 INFO - Started HttpContext[/selenium-server,/selenium-server]
09:36:44.862 INFO - Started HttpContext[/,/]
09:36:44.864 INFO - Started org.openqa.jetty.servlet.ServletHandler@628e2b14
09:36:44.864 INFO - Started HttpContext[/wd,/wd]
09:36:44.866 INFO - Started SocketListener on 0.0.0.0:5555
09:36:44.866 INFO - Started org.openqa.jetty.Server@651876a2
09:36:44.867 INFO - using the json request : {"class":"org.openqa.grid.common.RegistrationRequest","capabilities":[{"platform":"VISTA","browserName":"firefox","maxInstances":15}],"configuration":{"port":5555,"register":true,"host":"10.0.0.3","proxy":"org.openqa.grid.selenium.proxy.DefaultRemoteProxy","browser":"browserName=firefox,maxInstances=15","maxSession":5,"role":"node","hubHost":"localhost","registerCycle":5000,"hub":"http://localhost:4444/grid/register","hubPort":4444,"url":"http://10.0.0.3:5555","remoteHost":"http://10.0.0.3:5555"}}
09:36:44.868 INFO - starting auto register thread. Will try to register every 500 ms.
09:36:44.869 INFO - Registering the node to hub :http://localhost:4444/grid/register
09:36:45.531 INFO - Executing: org.openqa.selenium.remote.server.handler.Status@3f410c8b
09:36:45.533 INFO - Done: /status
```

You will need to place the previously downloaded .exe and .jar files (Selenium-standalone-server, Chromedriver.exe, IEDriverServer.exe) on the C:\ drives of your nodes or some other location, then you will need to change it in the getBrowserInstance() method. Also you will need to have java on your nodes:

```

public static DesiredCapabilities getBrowserInstance(String browserName) {
    switch (browserName) {
        case "firefox": {
            DesiredCapabilities capability = DesiredCapabilities.firefox();
            FirefoxProfile firefoxProfile = new FirefoxProfile();
            capability.setCapability(FirefoxDriver.PROFILE, firefoxProfile);
            return capability;
        }
        case "ie": {
            System.setProperty("webdriver.ie.driver", "C:\\IEDriverServer.exe");
            DesiredCapabilities capability = DesiredCapabilities
                .internetExplorer(); If you place the drivers for browsers somewhere
            return capability; else - you will need to change the location in this
        }
        case "chrome": {
            DesiredCapabilities capability = DesiredCapabilities.chrome();
            System.setProperty("webdriver.chrome.driver",
                "c:\\chromedriver.exe");
            return capability;
        }
        default: {
            DesiredCapabilities capability = DesiredCapabilities.firefox();
            return capability;
        }
    }
}

```

6. Setting up Jenkins:

You will need to download Jenkins CI and install it on your hub machine.

After it's installed you should type in you browser: localhost:8080 and if the installation is successful you will see the Jenkins dashboard.

As a prerequisite for further Jenkins setup you will need to have Maven and Java installed and set environmental variables for them.

i. Configuring Jenkins:

- a. Go to Manage Jenkins -> Configure System

Jenkins

New Job People Build History Selenium Grid **Manage Jenkins** search ENABLE AUTO REFRESH

Manage Jenkins

Configure System Configure global settings and paths.

Configure Global Security Secure Jenkins; define who is allowed to access/use the system.

Reload Configuration from Disk Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins Add, remove, disable or enable plugins that can extend the functionality of Jenkins. (**updates available**)

System Information Displays various environmental information to assist trouble-shooting.

System Log System log captures output from java.util.logging output related to Jenkins

b. Enter JDK information:

1. Enter JDK information

Maven Configuration

Default settings provider: Use default maven settings
Default global settings provider: Use default maven global settings

JDK

JDK installations	Name: <input type="text" value="JDK"/>	JAVA_HOME: <input type="text" value="C:\Program Files\Java\jdk1.7.0_25"/>
<input type="checkbox"/> Install automatically		
<input type="button" value="Add JDK"/>		
List of JDK installations on this system		

Ant

Ant installations:

List of Ant installations on this system

Maven

c. Enter Maven information:

**1. Enter Maven information.
Note: Maven should be installed on the machine**

Maven

Maven installations	Name: <input type="text" value="Maven"/>	MAVEN_HOME: <input type="text" value="C:\maven"/>
<input type="checkbox"/> Install automatically		
<input type="button" value="Add Maven"/>		
List of Maven installations on this system		

Maven Project Configuration

Global MAVEN_OPTS:

Local Maven Repository: Default (~/.m2/repository)

Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

Jenkins Location

Jenkins URL:

Please set a valid host name, instead of localhost

System Admin e-mail address:

- d. Click “Save” at the bottom after the above steps are done.
- ii. Install Selenium Grid plugin – it’s very convenient as it runs as hub and you do not need to start the hub separately you will also be able to see nodes registered to the hub.

a. Go to “Manage Jenkins” -> “Manage Plugins”

The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links like New Job, People, Build History, Selenium Grid, and Manage Jenkins. The Manage Jenkins link is highlighted with a red box. The main content area has a heading 'Manage Jenkins' and several management options: Configure System, Configure Global Security, Reload Configuration from Disk, Manage Plugins (which is also highlighted with a red box), and System Information. A message at the top indicates a new Jenkins version is available for download.

b. Click “Available” tab and type “Selenium” into search box

The screenshot shows the Jenkins Plugin Manager. The 'Available' tab is selected and highlighted with a red box. A search bar at the top right contains the text 'Filter: selenium'. Below the tabs, there are four buttons: Update, Available (highlighted with a red box), Installed, and Advanced. The main area displays a table of available plugins, with one row for 'Selenium Grid Plugin' highlighted with a red box. The table columns are Name, Version, and a checkbox for installation.

Name	Version
Selenium AES Plugin	0.5
Seleniumhq Plugin	0.4
seleniumphtmlreport Plugin	0.94
TestLink Plugin	3.10
Nerrvana Plugin	1.02.06
Selenium Builder Plugin	1.9
SeleniumMC Plugin	

1. Go to "Available"
2. Type in "Selenium" in search box
3. Download and install Selenium Grid plugin

c. Download and install the plugin

d. After the plugin is installed you will see a “Se” sign on the dashboard:

The screenshot shows the Jenkins dashboard. On the left sidebar, under the 'Manage Jenkins' section, there is a 'Selenium Grid' item with a checkmark icon. A black arrow points from the text 'After successful installationg will see the Grid sign on the dashboard' to this 'Selenium Grid' item. The main content area displays a table of build jobs, with the first job listed as 'Selenium Grid'. The table columns include 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The 'Last Success' column shows '20 hr - #66'. The 'Last Failure' column shows '1 mo 14 days - #18'. The 'Last Duration' column shows '3 min 59 sec'. The 'Name' column shows 'Selenium Grid' with a sun icon.

III. Create a new Job

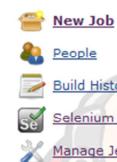
a. Go to “Manage Jenkins” -> “New job”

The screenshot shows the 'Manage Jenkins' page. At the top left, there are two buttons: 'New Job' and 'Manage Jenkins'. Both buttons are highlighted with orange boxes and have black arrows pointing towards them from the text '1. Go to "Manage Jenkins"' and '2. Go to "New Job"' respectively. Below these buttons, there is a warning message about a new Jenkins version (1.544) available for download. The main content area contains several configuration links: 'Configure System', 'Configure Global Security', 'Reload Configuration from Disk', and 'Manage Plugins'. There are also 'Setup Security' and 'Dismiss' buttons at the bottom right.

b. Select “Build a free-style software project”

Jenkins

Jenkins > All >



Build Queue	
No builds in the queue.	
#	Status
1	Idle
2	Idle

Job name:

Build a free-style software project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Build a maven2/3 project
Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Build multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Monitor an external job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

Copy existing Job
Copy from:

1. Enter job name
2. Select "Build a free-style software project"

OK

- c. Click “Execute concurrent builds if necessary” and select “None” for version control (unless you want to use a repository for your tests)

Jenkins

Jenkins > new_job > configuration



Project name: new_job

Description:

[Raw HTML] [Preview](#)

Discard Old Builds

This build is parameterized

Disable Build (No new builds will be executed until the project is re-enabled.)

Execute concurrent builds if necessary

1. Select "Execute concurrent builds..."
2. Select "None" - unless you are going to use source control (e.g. GitHub, Bitbucket)

[Advanced...](#)

Advanced Project Options

Source Code Management

CVS

CVS Projectset

None

Subversion

- d. Click “Add build step”

Source Code Management

CVS

CVS Projectset

None

Subversion

Build Triggers

Build after other projects are built

Build periodically

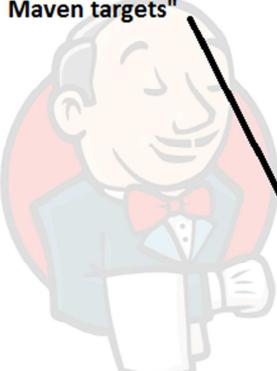
Poll SCM

Build

Post-Build Actions

Save Apply

e. Select “Invoke top-level Maven targets”



1. Select "Invoke top-level Maven targets"

Source Code Management

- CVS
- CVS Projectset
- None
- Subversion

Build Triggers

- Build after other projects are built
- Build periodically
- Poll SCM

Build

Add build step ▾

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke top-level Maven targets**

Save Apply

f. Follow the steps from the screenshot below:



Poll SCM

Build

Invoke top-level Maven targets

Maven Version: Maven

Goals: -Dtests=firefox_tests.xml -Dreceiver=youremail@domain.com

POM: C:\

Properties

JVM Options

Use private Maven repository

Settings file: Use default maven settings

Global Settings file: Use default maven global settings

1. Select the Maven setup you made before
2. Enter Maven command to run particular suite of test and enter email where you want to receive the test-results
3. Enter the location of your POM file (should be the project directory)

Delete

g. Follow the steps from screenshot below:



CVS

CVS Projectset

None

Subversion

Build Triggers

Build after other projects are built

Build periodically

Schedule: 0 8 * * *

1. Click "Build periodically"
2. Type in schedule box the time you want to run the tests every day (e.g. 0 8 * * * for 8 am each day)

Poll SCM

Build

Add build step ▾

Post-build Actions

Add post-build action ▾

h. Click “Save” at the bottom

7. Running tests

If you followed all the steps from above you will see a new job created on Jenkins dashboard:

The screenshot shows the Jenkins dashboard with a sidebar on the left containing links like 'New Job', 'People', 'Build History', 'Selenium Grid', and 'Manage Jenkins'. The main area displays a table of build jobs. One job, 'new job', is highlighted with a red box around its name in the 'Name' column. Below the table, a legend indicates icons for Success (blue), Warning (yellow), and Failure (red). A large black arrow points upwards from the text 'The new job you created' towards the 'new job' entry in the table.

All	S	W	Name ↓	Last Success	Last Failure	Last Duration
	●	○		25 min - #71	1 mo 19 days - #18	3 min 58 sec
	●	○		54 min - #416	1 mo 16 days - #366	29 min
	●	○		20 min - #59	1 mo 19 days - #8	1 min 38 sec
	●	○	new job	N/A	N/A	N/A

Start the node on your local machine or on your node by clicking the .cmd file you created before and make sure node is connected.

The job is setup to run at a certain time but you can also start it by clicking the “triangle” sign in the right corner:

This screenshot is similar to the one above, showing the Jenkins dashboard with the 'new job' entry. A large black arrow points from the text 'Click it to start the tests' towards the 'start' icon (a triangle) for the 'new job' entry in the table. The table data is identical to the previous screenshot.

All	S	W	Name ↓	Last Success	Last Failure	Last Duration
	●	○		25 min - #71	1 mo 19 days - #18	3 min 58 sec
	●	○		54 min - #416	1 mo 16 days - #366	29 min
	●	○		20 min - #59	1 mo 19 days - #8	1 min 38 sec
	●	○	new job	N/A	N/A	N/A

After the tests run, you will get a report with screenshots (if there are any failures- the tests in the starter kit will have some failures on purpose) to the email you've provided to the Maven command in Jenkins.

Test	Methods Passed	Scenarios Passed	# skipped	# failed	Total Time	Included Groups	Excluded Groups
Test III	2	2	0	1	18.1 seconds		
Test II	2	2	0	1	21.2 seconds		
Test I	1	1	0	0	22.5 seconds		
Total	5	5	0	2	22.5 seconds		

Class	Method	# of Scenarios	Start	Time (ms)
Test III — failed				
tests.VerifyPagesLoadFine	verifyLoadTestingToolsPageLoadsFine	1	1387905896087	1307
Test III — passed				
tests.VerifyPagesLoadFine	veifyLoadTestPageLoadsFine	1	1387905894039	1101
	verifyPerformancePageLoadsFine	1	1387905895144	942
Test II — failed				

All the tests in the starter kit have thorough comments of all the steps performed and you can basically start creating your automation framework by modifying the example tests- this way you will follow the structure of current framework.

*Disclaimer: all the tests in the repository for this Guide were configured using the following:

selenium-server-standalone-2.31.0.jar, firefox 22, ie 9, chrome 31.0.1650.63 m

And finally you will ask why automation is easy and fun? Well, first of all once the structure of your framework is created you will need just to add new tests and reuse most of the code you have in your page classes, the fun part is that while your tests are running and the job is being done – you can have a cup of coffee meanwhile ☺.