**What are the essential qualities of a good test report?**

* **Brevity –**
  + A report should be short and concise.
  + It should clearly state the proportion of total no. of success or failure.
* **Trackability –**
  + Captures all the footprints that could lead to the root cause of a failure.
* **Traceability –**
  + It must provide the ability to review the following.
    - Historical data for test cases and failures.
    - Age of a particular failure.
* **Sharable –**
  + It should support a format which you can share through email or integrate with CI tools like Jenkins/Bamboo.
* **Test coverage –**
  + It should highlight the test coverage for the following.
    - Test coverage of the module under test.
    - Test coverage of the application under test.
* [**TestNG HTML Report Generation**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#h1)**.**
* [**JUnit-Style Report Generation**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#h2)**.**
  + [**Simple JUnit Reports**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#h2.1)**, and**
  + [**Convert JUnit to HTML Reports**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#h2.2)**.**
* [**Generating Extent HTML Reports**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#h3)**.**

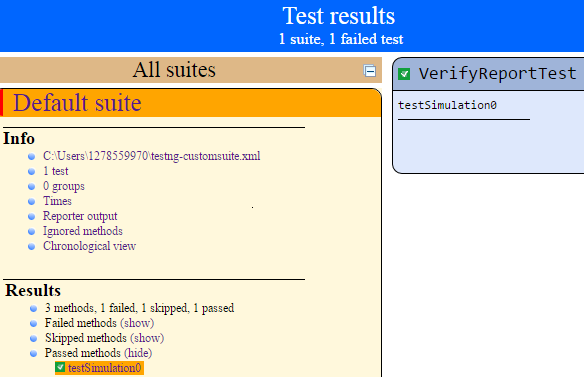
Selenium Webdriver doesn’t have a built-in reporting feature but there are plugins like the TestNG and JUnit which can add this functionality.

### 1- Generate Reports Using TestNG.

TestNG library brings a very convenient reporting feature. Once you execute the tests, TestNG generates a test output folder at the root of the project. It combines two kinds of reports.

#### Detailed Report.

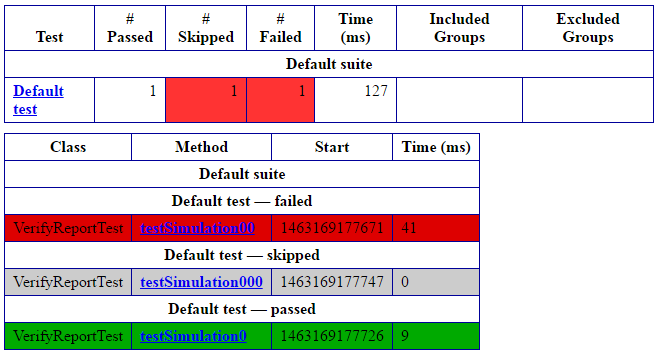
You can find this report in the <***index.html***> file. It combines the detailed information like the errors, test groups, execution time, step-by-step logs and TestNG XML file.

[](http://www.techbeamers.com/wp-content/uploads/2016/05/Generate-Reports-index.html-in-TestNG.png)

**Generate Reports (index.html) in TestNG.**

#### Summary Report.

It is the trimmed version and informs about the test pass/fail/skip count. You can see it from the <***emailable-report.html***> file. It’s an email friendly report which you can embed and share with the stakeholders.

[](http://www.techbeamers.com/wp-content/uploads/2016/05/Generate-Reports-emailable-report.html-in-TestNG.png)

**Generate Reports (emailable-report.html) in TestNG.**

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

#### 1.1- Steps To Generate Reports Using TestNG.

##### Step-1) Select The TestNG Reporting Interface.

TestNG supplies two interfaces to implement reporting in Selenium Webdriver. You can apply any of these two in your projects.

1. Testers commonly use the <**[ITestListener](http://testng.org/javadocs/org/testng/ITestListener.html" \t "_blank)**> Interface.
2. Another is the <***IReporter***> Interface which is least popular.

In this blog post, we’ll cover the application of the <***ITestListener***> Interface. You’ll have to create a TestNG project in Eclipse. You can refer our below post that teaches to create a basic TestNG project.

[**Learn to create a TestNG project in Eclipse.**](http://www.techbeamers.com/create-testng-test-case-using-selenium/)

##### Step-2) Generate Reports Using The <ITestListener> Interface.

GenerateReport.Java - Implementation of the ITestListener Interface.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | import org.testng.ITestContext;  import org.testng.ITestListener;  import org.testng.ITestResult;    public class GenerateReport implements ITestListener {    @Override  public void onStart(ITestContext arg0) {    System.out.println("+Begin test: " + arg0.getName());    }    @Override  public void onTestStart(ITestResult arg0) {    System.out.println(" Starting test: " + arg0.getName());    }    @Override  public void onTestSuccess(ITestResult arg0) {    System.out.println(" Test passed: " + arg0.getName());    }    @Override  public void onTestFailure(ITestResult arg0) {    System.out.println(" Test failed: " + arg0.getName());    }    @Override  public void onTestSkipped(ITestResult arg0) {    System.out.println(" Test ignored: " + arg0.getName());    }    @Override  public void onFinish(ITestContext arg0) {    System.out.println("-End test: " + arg0.getName());    }    @Override  public void onTestFailedButWithinSuccessPercentage(ITestResult arg0) {    // TODO Auto-generated method stub    }    } |

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

##### Step-3) Verify The Report Generation Process.

VerifyReportTest.Java - Verify if the test code is producing the report.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import org.testng.Assert;  import org.testng.annotations.Listeners;  import org.testng.annotations.Test;    @Listeners(GenerateReport.class)  public class VerifyReportTest {    @Test  public void testSimulation0() {    Assert.assertTrue(true);    }    @Test  public void testSimulation00() {    Assert.assertTrue(false);    }    // Test case <testSimulation000> depends on the intentionally  // failed test case <testSimulation00>    @Test(dependsOnMethods = "testSimulation00")  public void testSimulation000() {    }    } |

### 2- Generate Reports Using JUnit.

JUnit is another very good tool that can add intuitive reporting feature in your Selenium project. It provides the JUnit <[TestWatcher](http://junit.org/junit4/javadoc/latest/index.html" \t "_blank)> class to introduce reporting ability.

The JUnit’s TestWatcher class has the <***failed()***> and <***succeeded()***> methods which you can override. The JVM would call them automatically whenever it smells a pass or failure.

#### 2.1- Steps To Generate Reports In JUnit Style.

We’ve just summarized the summary of the steps that you can use to generate reports using the JUnit plugin.

1. Create a new Java class (name it as ***JUnitTestReporter***) that applies the JUnit rules with the help of the TestWatcher() class.
2. Override the <***succeeded()***> method so that the names of passed tests could get displayed at the console with the <***Passed***> status.
3. Override the <***failed()***> method so that the names of the failed tests could appear at the console with the <***Failed***> status.
4. Create a sample test class as <***JUnitSampleTest***> which must extend the <***JUnitTestReporter***> class to utilize the overridden <***succeeded()***> and <***failed()***> methods.

Now we’ll explain how simple it is to work with JUnit so that you can quickly generate the summary of the test execution.

#### 2.2- Simple Report Generation Using JUnit.

##### 2.2.1- Generate Reports By Creating A JUnit Test Watcher Class.

Create a simple project in Eclipse and add the below file <***JUnitTestReporter.Java***> to your project. This file will display the report into the Eclipse console.

JUnitTestReporter.Java- Generate Reports by creating a JUnit Test Watcher Class.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | import org.junit.Rule;  import org.junit.rules.TestRule;  import org.junit.rules.TestWatcher;  import org.junit.runner.Description;  import org.junit.runners.model.Statement;    public class JUnitTestReporter {    @Rule  public TestRule junitWatcher = new TestWatcher() {    @Override  public Statement apply(Statement base, Description description) {  return super.apply(base, description);  }    @Override  protected void succeeded(Description description) {  System.out.println(description.getDisplayName() + " "  + "Test Passed!");  }    @Override  protected void failed(Throwable e, Description description) {  System.out.println(description.getDisplayName() + " "  + e.getClass().getSimpleName());  }  };  } |

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

##### 2.2.1- Prepare A Sample JUnit Test Suite To Verify Report Generation.

Also, add the below file <***JUnitSampleTest.Java***> to your project. It’ll be the main test file which you’ll execute from the Eclipse.

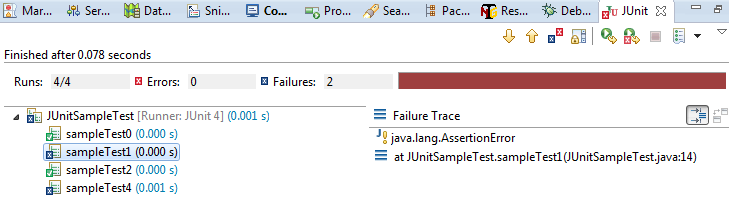
JUnitSampleTest.Java- Prepare a Sample JUnit Test Suite for Verifying Reports.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import static org.junit.Assert.assertTrue;    import org.junit.Test;    public class JUnitSampleTest extends JUnitTestReporter {    @Test  public void sampleTest0() {  assertTrue(1 < 2);  }    @Test  public void sampleTest1() {  assertTrue(1 > 2);  }    @Test  public void sampleTest2() {  assertTrue(1 < 2);  }    @Test  public void sampleTest4() {  assertTrue(1 > 2);  }  } |

When you run the above JUnit tests from Eclipse, it’ll show you the test results in Eclipse as captured in the below screenshot. The downside to this technique is that it doesn’t save the output in an HTML file.

[](http://www.techbeamers.com/wp-content/uploads/2016/05/Generate-Reports-Using-JUnit-Plugin.png)

**Generate Reports Using JUnit Plugin.**

But you can generate an HTML report by modifying the above code. We’ve given the technique to produce the test summary in the next section. It saves the report in an HTML file.

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

#### 2.3- HTML Report Generation Using JUnit.

We’ll create a new JUnit class that would enable the HTML report generation. This class will also override the TestWatcher methods to implement the desired features.

* We’ll define two static members; one is a File object and the second one is BufferWriter’s handle that will help us adding the test execution summary to the report file.
* We’ll use the following JUnit annotations.
  + ***@BeforeClass –*** It’ll help us define the setup() method. It’ll create/open the HTML report file as per the situation.
  + ***@AfterClass –*** We’ll use it for the purpose of cleanup. It’ll also update the HTML report file to add the HTML footer and close all the open handles.
* We’ll override the following two methods.
  + **<succeeded()> method –** It’ll write the names and status of the test cases passed during execution.
  + **<failed()> method –** It’ll log the names and status of the test cases failed during execution.

Now, you’ll find the source code of the <***JUnitHTMLReporter***> class.

JUnitHTMLReporter.java

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77 | import java.awt.Desktop;  import java.io.BufferedWriter;  import java.io.File;  import java.io.FileWriter;  import java.io.IOException;  import java.text.DateFormat;  import java.text.SimpleDateFormat;  import java.util.Date;    import org.junit.AfterClass;  import org.junit.BeforeClass;  import org.junit.Rule;  import org.junit.rules.TestRule;  import org.junit.rules.TestWatcher;  import org.junit.runner.Description;  import org.junit.runners.model.Statement;    public class JUnitHTMLReporter {    static File junitReport;  static BufferedWriter junitWriter;    @BeforeClass  public static void setUp() throws IOException {    String junitReportFile = System.getProperty("user.dir")  + "\\junitReportFile.html";  DateFormat dateFormat = new SimpleDateFormat("dd-MMM-yyyy HH:mm:ss");  Date date = new Date();  junitReport = new File(junitReportFile);  junitWriter = new BufferedWriter(new FileWriter(junitReport, true));  junitWriter.write("<html><body>");  junitWriter.write("<h1>Test Execution Summary - " + dateFormat.format(date)  + "</h1>");    }    @AfterClass  public static void tearDown() throws IOException {    junitWriter.write("</body></html>");  junitWriter.close();  Desktop.getDesktop().browse(junitReport.toURI());    }    @Rule  public TestRule watchman = new TestWatcher() {    @Override  public Statement apply(Statement base, Description description) {  return super.apply(base, description);  }    @Override  protected void succeeded(Description description) {  try {  junitWriter.write(description.getDisplayName() + " "  + "success!");  junitWriter.write("<br/>");  } catch (Exception e1) {  System.out.println(e1.getMessage());  }  }    @Override  protected void failed(Throwable e, Description description) {  try {  junitWriter.write(description.getDisplayName() + " "  + e.getClass().getSimpleName());  junitWriter.write("<br/>");  } catch (Exception e2) {  System.out.println(e2.getMessage());  }  }  };  } |

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

So, we have a ready to use HTML reporter class, all we need now is the JUnit Test class that will extend it and call its methods. The code of this new class is available below.

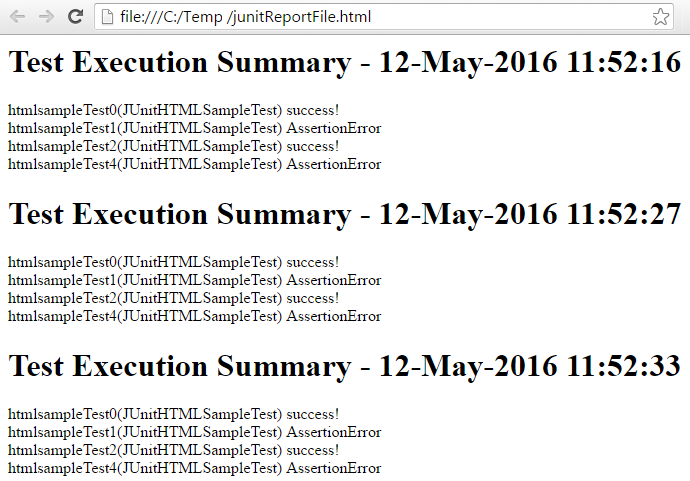
JUnitHTMLSampleTest.java

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import static org.junit.Assert.assertTrue;    import org.junit.Test;    public class JUnitHTMLSampleTest extends JUnitHTMLReporter {    @Test  public void htmlsampleTest0() {  assertTrue(1 < 2);  }    @Test  public void htmlsampleTest1() {  assertTrue(1 > 2);  }    @Test  public void htmlsampleTest2() {  assertTrue(1 < 2);  }    @Test  public void htmlsampleTest4() {  assertTrue(1 > 2);  }  } |

When you run the above class file as JUnit test case, then the <junitReportFile.html> will get generated. It’ll contain the test cases pass/fail info and look like the one as given in the below screenshot.

[](http://www.techbeamers.com/wp-content/uploads/2016/05/Generate-Reports-in-HTML-Format-Using-JUnit.png)

**Generate Reports in HTML Format Using JUnit.**

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

### 3- Generate Reports Using Extent Library.

The third and the last technique for generating some awesome reports is by using the <***Extent Report***> library. It comes with a rich set of features.

* Ability to generate dynamic HTML logs.
* Represents test case status with the help of PIE Charts.
* Generates step-by-step test case summary.
* Ability to filter reports based on test status.
* It maintains execution history.
* It captures details like OS, Memory, Java version and so on.
* You can attach error screenshots within the report.

You can download the Extent library from the below URL.

**[Download Extent Report Jar Files.](http://relevantcodes.com/ExtentReports-for-selenium/" \t "_blank)**

[](http://www.techbeamers.com/wp-content/uploads/2016/05/Generate-Reports-Using-Extent-Library.png)

**Download Extent Library Jar Files.**

Once you have the reporting library, then follow the below steps to use them in a Selenium Webdriver project.

* [***Create or open a demo project in Eclipse***](http://www.techbeamers.com/six-steps-to-setup-selenium-webdriver-project-in-eclipse/).
* Add the Jar files shown in the above picture as external libraries to your project.
* Also, make sure to add the Jars that are under the <***lib***> folder else the exception may occur at run-time.

Now, we are attaching the source code of the sample Java program that we’d created for the Extent report validation. You can use the below Java file as is in your demo project.

ExtentReportTest.Java

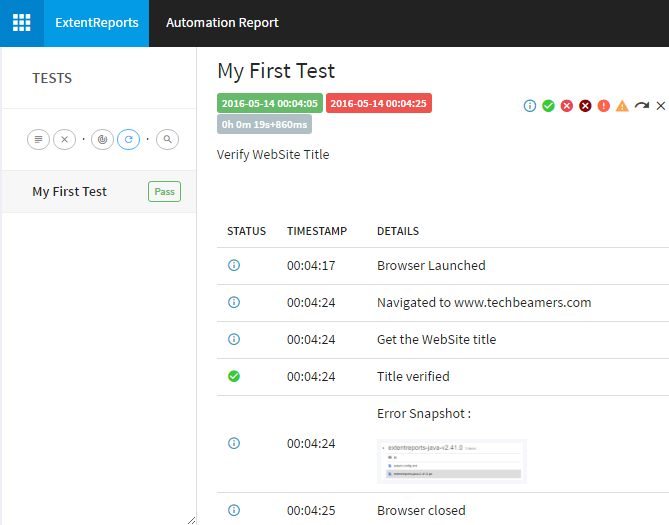
Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.testng.Assert;  import org.testng.annotations.Test;    import com.relevantcodes.extentreports.ExtentReports;  import com.relevantcodes.extentreports.ExtentTest;  import com.relevantcodes.extentreports.LogStatus;    public class ExtentReportTest {    @Test  public void verifySeleniumBlog() {    String extentReportFile = System.getProperty("user.dir")  + "\\extentReportFile.html";  String extentReportImage = System.getProperty("user.dir")  + "\\extentReportImage.png";    // Create object of extent report and specify the report file path.  ExtentReports extent = new ExtentReports(extentReportFile, false);    // Start the test using the ExtentTest class object.  ExtentTest extentTest = extent.startTest("My First Test",  "Verify WebSite Title");    // Launch the FireFox browser.  WebDriver driver = new FirefoxDriver();    driver.manage().window().maximize();    extentTest.log(LogStatus.INFO, "Browser Launched");    // Open application.  driver.get("/");    extentTest.log(LogStatus.INFO, "Navigated to www.techbeamers.com");    // get title.  String title = driver.getTitle();    extentTest.log(LogStatus.INFO, "Get the WebSite title");    // Verify title.  Assert.assertTrue(title.contains("Selenium Webdriver"));    extentTest.log(LogStatus.PASS, "Title verified");    // In case you want to attach screenshot then use below method  // We used a random image but you've to take screenshot at run-time  // and specify the error image path.  extentTest.log(  LogStatus.INFO,  "Error Snapshot : "  + extentTest.addScreenCapture(extentReportImage));    // Close application.  driver.quit();    extentTest.log(LogStatus.INFO, "Browser closed");    // close report.  extent.endTest(extentTest);    // writing everything to document.  extent.flush();  }  } |

[**Back to top**](http://www.techbeamers.com/generate-reports-selenium-webdriver/#top)

Once you run the above code, it’ll launch the specified URL in Firefox and produce an intuitive Extent test report. The final test report would look like the one shown in the below screenshot.

[](http://www.techbeamers.com/wp-content/uploads/2016/05/Live-Extent-Report-Example.png)

**Live Extent Report Example.**

So, that was all we wanted to share in this post. We hope it would enable you to use the three distinct methods to generate reports in Selenium. And, you’ll be able to reap the real benefits of the test automation.