

# Selcukes

Ramesh Babu Prudhvi

Version 1.6.0, 2022-04-28T02:54:02Z

# Table of Contents

1. Selcukes Core	1
1.1. Overview	1
1.2. Features	1
1.3. Getting Started	2
1.4. Browser Tests	3
1.5. Desktop Tests	4
1.6. Mobile Tests	5
2. WebDriver Binaries	7
2.1. Motivation	7
2.2. Setup	8
2.3. Driver Management	8
2.4. Advanced Configuration	10
3. Selcukes DataBind	11
3.1. Setup	11
3.2. Data Mapper	11
4. Selcukes Extent Reports	15
4.1. Features	15
4.2. Setup	15
4.3. Usage	15
5. Selcukes Notifier	17
5.1. Motivation	17
5.2. Setup	17
5.3. Usage	17
6. Selcukes Snapshot	19
6.1. Setup	19
6.2. Usage	19
7. Selcukes Excel Runner	21
7.1. Setup	21
8. Selcukes Reports	22
8.1. Setup	22
9. Selcukes Configuration	23
9.1. Environment Properties	23
9.2. Runtime Properties	23
9.3. Logger Properties	24
10. Community	25
11. Support	26

# Chapter 1. Selcukes Core

Selcukes Core can help your team to build high-quality and highly scalable automated tests.

## 1.1. Overview

### Open Source

As an open source company, we're passionately engaged in numerous projects, initiatives and standards where we represent the needs and requirements of our many customers and partners.

### Unified Framework API

All features that we provide- use the same syntax. Once you learn how to write tests for the web, you can start immediately writing for mobile, desktop, or REST.

### Test Everything

Web, including responsive, iOS, Android, Desktop, and REST services.

### Integrations

Seamlessly integrate the framework with your existing tools and processes. Execute tests in the clouds, distributed and publish test results in reporting solutions.

## 1.2. Features

1. Easy add new logic to your tests without causing regression issues. Control the whole execution workflow - change browsers or reuse them. Retry your failing tests to make sure that there is a real problem.
2. Automate UI actions or user scenarios on real devices or emulators with Selcukes features from test creation to execution.
3. Cross-platform screenshot and video recording on test failure


What can be tested?	Selcukes	Benefits
<b>Web automation cross browser</b> - Chrome, Firefox, Edge, IE, Opera, Chrome Headless, Firefox Headless.		The standard for web automation. Open source. Large community. All major cloud platforms support it. Browser vendors support.
<b>Mobile Automation</b> - iOS, Android native, hybrid and mobile web apps.		The standard for mobile automation. Open source. Large community. Compatible with major cloud providers. Can run in Selenium Grid.
<b>Desktop Automation</b> - WPF, WinForms and Universal applications.	WinAppDriver	Officially supported by Microsoft. Can run in Selenium Grid.
<b>API Automation</b> - REST web services.	RestAssured	The standard for API automation in Java. Open source. Large community.
	Java	Open source. Run tests on Windows, Linux and MacOS.
	TestNG, JUnit	Rich set of assert libraries. Open source. Large community.
	Support all major cloud providers - CrossBrowserTesting, SauceLabs, Browserstack (mobile,web)	
	Easily integrated with all CI systems	

Figure 1. Technologies

## 1.3. Getting Started

What you need to start

### 1.3.1. Java 11

Test Automation is a development activity, so you will need some familiarity with Java. Selcukes uses Java 11, so make sure you have a [JDK 11 or later](#) installed.

### 1.3.2. IDE

You will need a modern IDE to work with Java. We recommend IntelliJ (you can download the Community Edition, which is free, [from here](#). But Eclipse will work fine as well.

### 1.3.3. A Build Tool

You will also need a build tool, either Gradle or Maven, to run your tests and generate your reports. Make sure you have either [Gradle 3.x or higher](#) or [Maven 3.3.x or higher](#) installed.

### 1.3.4. Quick Start

The quickest way to create a new project is to take one of the starter projects on Github. You can find the starter project for TestNG at [selcukes-java-skeleton](#).

You can clone this repository:

```
git clone https://github.com/selcukes/selcukes-java-skeleton.git
cd selcukes-java-skeleton
```

Or simply [download a zip file from here](#).

If you are using Maven, you can also create a new project using one of the Selcukes [Maven Archetypes](#).

### 1.3.5. So far so good?

The starter project comes with a demo test that you can run. From the command line, run either

```
$ mvn clean verify
```

or

```
$ gradle clean test
```

## 1.4. Browser Tests

Add `selcukes.yaml` file in resource folder and updated below options as required

```
web:
  remote: false
  browserName: CHROME
  headless: true
  serviceUrl: "http://localhost:4444"
```

Create Sample Browser Test as follows

```

public class WebTest {

    DriverManager<RemoteWebDriver> driverManager;
    WebDriver driver;

    @BeforeMethod
    void beforeTest() {
        driverManager = new DriverManager<>();
        driver = driverManager.createDriver(DeviceType.BROWSER);
    }

    @Test
    public void webTest() {
        WebPage page = new WebPage(driver);
        page.enableDriverEvents();
        page.open("https://www.google.com/");
        Assert.assertEquals(page.title(), "Google");
    }

    @AfterMethod
    void afterTest() {
        driverManager.getManager().destroyDriver();
    }
}

```

## 1.5. Desktop Tests

Add `selcukes.yaml` file in resource folder and updated below options as required

```

windows:
  serviceUrl: "http://127.0.0.1:4723"
  winApp-path: "C:/Program Files (x86)/Windows Application Driver/WinAppDriver.exe"
  app: "C:\\Windows\\System32\\notepad.exe"

```

Create Sample WindowDriver Test for Notepad application as follows

```

public class NotepadTest {
    DriverManager<WindowsDriver> driverManager;

    @BeforeTest
    public void beforeTest() {
        driverManager = new DriverManager<>();
    }

    @Test
    public void notepadTest() {
        WindowsDriver driver = driverManager.createDriver(DeviceType.DESKTOP);
        WinPage page = new WinPage(driver);
        page.write(By.className("Edit"), "This is sample");
    }

    @AfterTest
    public void afterTest() {
        driverManager.getManager().destroyDriver();
    }
}

```

## 1.6. Mobile Tests

Add `selcukes.yaml` file in resource folder and updated below options as required

```

mobile:
  serviceUrl: "http://127.0.0.1:4723"
  app: "ApiDemos-debug.apk"

```

Create Sample AndroidDriver Test for `ApiDemos-debug.apk` as follows

```

public class MobileTest {
    DriverManager<AppiumDriver> driverManager;

    @BeforeTest
    void beforeTest() {
        driverManager = new DriverManager<>();
    }

    @Test(enabled = false)
    public void remoteTest() {
        AppiumDriver driver = driverManager.createDriver(DeviceType.MOBILE);
        MobilePage page = new MobilePage(driver);
        page.click(By.xpath("//android.widget.TextView[contains(@text,'Views')]"));
    }

    @AfterTest
    void afterTest() {
        driverManager.getManager().destroyDriver();
    }
}

```



# Chapter 2. WebDriver Binaries

**WebDriver Binaries** is an open-source Java library that automatically downloads and configures the binary drivers (e.g., chromedriver, geckodriver, msedgedriver, etc.) required by Selenium WebDriver.

## 2.1. Motivation

**Selenium WebDriver** is a library that allows controlling web browsers programmatically. It provides a cross-browser API that can be used to drive web browsers (e.g., Chrome, Edge, or Firefox, among others) using different programming languages (e.g., Java, JavaScript, Python, C#, or Ruby). The primary use of Selenium WebDriver is implementing automated tests for web applications.

Selenium WebDriver carries out the automation using the native support of each browser. For this reason, we need to place a binary file called *driver* between the test using the Selenium WebDriver API and the browser to be controlled. Examples of drivers for major web browsers nowadays are **chromedriver** (for Chrome), **geckodriver** (for Firefox), or **msedgedriver** (for Edge). As you can see in the following picture, the communication between the WebDriver API and the driver binary is done using a standard protocol called **W3C WebDriver** (formerly the so-called *JSON Wire Protocol*). Then, the communication between the driver and the browser is done using the native capabilities of each browser.

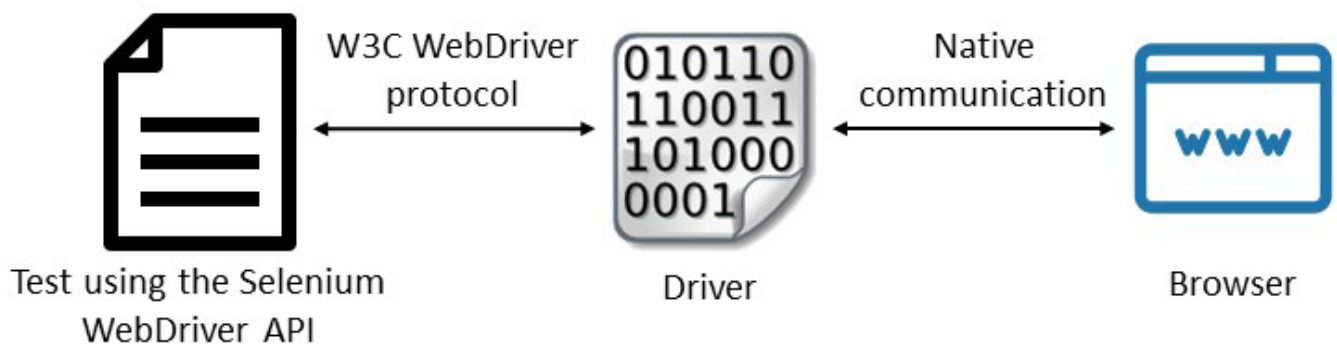


Figure 2. Selenium WebDriver Architecture

From a practical point of view, we need to make a *driver management process* to use Selenium WebDriver. This process consists on:

1. **Download.** Drivers are platform-specific binary files. To download the proper driver, we have to identify the driver type we need (e.g., chromedriver if we want to use Chrome), the operating system (typically, Windows, Linux, or Mac OS), the architecture (typically, 32 or 64 bits), and very important, the driver version. Concerning the version, each driver release is usually compatible with a given browser version(s). For this reason, we need to discover the correct driver version for a specific browser release (typically reading the driver documentation or release notes).
2. **Setup.** Once we have downloaded the driver to our computer, we need to provide a way to locate this driver from our Selenium WebDriver tests. In Java, this setup can be done in two different ways. First, we can add the driver location to our **PATH** environmental variable. Second, we can use *Java system properties* to export the driver path. Each driver path should be

identified using a given system property, as follows:

```
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");
System.setProperty("webdriver.edge.driver", "/path/to/msedgedriver");
System.setProperty("webdriver.opera.driver", "/path/to/operadriver");
System.setProperty("webdriver.ie.driver", "C:/path/to/IEDriverServer.exe");
```

3. Maintenance. Last but not least, we need to warranty the compatibility between driver and browser in time. This step is relevant since modern browsers automatically upgrade themselves (i.e., they are *evergreen* browsers), and for this reason, the compatibility driver-browser is not warranted in the long run. For instance, when a WebDriver test using Chrome faces a driver incompatibility, it reports the following error message: *"this version of chromedriver only supports chrome version N."* As you can see in [StackOverflow](#), this is a recurrent problem for manually managed drivers (chromedriver in this case).

## 2.2. Setup

WebDriver Binaries is primarily used as a Java dependency . We typically use a *build tool* (such as [Maven](#) or [Gradle](#)) to resolve the WebDriverManager dependency. in Maven as follows (notice that it is declared using the `test` scope, since it is typically used in tests classes):

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>webdriver-binaries</artifactId>
  <version>1.6.0</version>
  <scope>test</scope>
</dependency>
```

In the case of a Gradle project, we can declare WebDriverManager as follows (again, for tests):

```
dependencies {
    testImplementation("io.github.selcukes:webdriver-binaries:1.6.0")
}
```

## 2.3. Driver Management

The primary use of WebDriver Binaries is the automation of driver management. For using this feature, you need to select a given driver in the WebDriver Binaries API (e.g., `chromeDriver()` for Chrome) and invoke the method `setup()`. The following example shows a test case using [TestNG](#), Selenium WebDriver, WebDriver Binaries. In this test, we invoke WebDriver Binaries in the setup method for all tests (`@BeforeClass`). This way, the required binary (chromeDriver) will be available for all the WebDriver tests using Chrome in this class.

```

public class BrowserTest {
    WebDriver driver;

    @BeforeClass
    static void setupClass() {
        WebDriverBinary.chromeDriver().setup();
    }

    @BeforeTest
    void setupTest() {
        driver = new ChromeDriver();
    }

    @AfterTest
    void teardown() {
        driver.quit();
    }

    @Test
    void test() {
        driver.get("https://google.com");
        String title = driver.getTitle();
        Assert.assertEquals(title, "Google");
    }
}

```

WebDriver Binaries provides a set of *binaries* for Chrome, Firefox, Edge, Opera, Chromium, and Internet Explorer. The basic use of these binary is the following:

```

WebDriverBinary.chromeDriver().setup();
WebDriverBinary.firefoxDriver().setup();
WebDriverBinary.ieDriver().setup();
WebDriverBinary.edgeDriver().setup();
WebDriverBinary.edgeDriver().setup();
WebDriverBinary.operaDriver().setup();

```

### 2.3.1. Resolution Algorithm

WebDriver Binaries executes a *resolution algorithm* when calling to `setup()` in a given manager. The most relevant parts of this algorithm are the following:

1. WebDriverBinary tries to find the browser version. To this aim, WebDriverBinary uses internally a knowledge database called commands' database. This database is a collection of shell commands used to discover the version of a given browser in the different operating systems (e.g., `google-chrome --version` for Chrome in Linux).
2. Using the browser version, it tries to find the proper driver version. This process is different for each browser. In Chrome and Edge, their respective drivers (chromedriver and msedgedriver)

maintainers also publish resources to identify the suitable driver version for a given major browser release. For instance, to find out the version of chromedriver required for Chrome 100, we need to read the following [file](#). Unfortunately, this information is not available in other browsers (e.g., Firefox and Opera) or older versions of Chrome and Firefox. For this reason, WebDriverManager uses another knowledge database called versions database. This database maps the browser releases with the known compatible driver versions.

3. Once the driver version is discovered, WebDriverManager downloads this driver to a local cache (located at `%temp%webdriver` by default). These drivers are reused in subsequent calls.
4. Finally, WebDriverManager exports the driver path using Java system properties (e.g., `webdriver.chrome.driver` in the case of the Chrome manager).

This process automated the first two stages of the driver management previously introduced, i.e., download and setup. To support the third stage (i.e., maintenance), WebDriverManager implements *resolution cache*. This cache (called by default `resolution.properties` and stored in the root of the driver cache) is a file that stores the relationship between the resolved driver and browser versions. This relationship is valid during a given *time-to-live* (TTL). The default value for this TTL is 1 hour for browsers and 1 day for drivers. In other words, the discovered browser version is valid for 1 hour, and the driver version is considered correct for 1 day. This mechanism improves the performance dramatically since the second (and following) calls to the resolution algorithm for the same browser are resolved using only local resources (i.e., without using the shell nor requesting external services).

## 2.4. Advanced Configuration

WebDriver Binaries provides different ways of configuration. First, by using its *Java API*. To that aim, each manager (e.g., `chromeDriver()`, `firefoxDriver()`, etc., allows to concatenate different methods of this API to specify custom options or preferences. For example (the explanation of these methods and the other possibilities are explained in the tables at the end of this section):

```
WebDriverBinary.firefoxDriver().version("v0.26.0").setup();
WebDriverBinary.chromeDriver().targetPath("temp").setup();
WebDriverBinary.firefoxDriver().arch32().setup();
```

# Chapter 3. Selcukes DataBind

Selcukes `DataBind` helps to parse Json and Yml files

## 3.1. Setup

Selcukes DataBind used as a Java dependency in Maven as follows

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>selcukes-databind</artifactId>
  <version>1.6.0</version>
</dependency>
```

In the case of a Gradle project, we can declare Selcukes DataBind as follows:

```
dependencies {
    implementation("io.github.selcukes:selcukes-databind:1.6.0")
}
```

## 3.2. Data Mapper

DataMapper object helps to read and write Json or yaml/yml files with `@DataFile` annotation.

### 3.2.1. Resolution Algorithm

DataMapper looks for matching data files in test resource folder by converting POJO class name to a SnakeCase json or yaml/yml file.

- For POJO class `TestUsers.java`
- Matching test data files are `test_users.json` or `test_users.yml` or `test_users.yaml`

`@DataFile` annotation also takes additional attributes as follows

- `fileName` : Specify custom data file like 'mydata.json'
- `filePath` : path where data file is located. 'src/main/resources'
- `rootFolder` : root directory

### 3.2.2. Read data files

Let's first look at the `test_users.json` file we'll be reading

```
{
  "users": [
    {
      "username": "MyName",
      "password": "things"
    },
    {
      "username": "TestName",
      "password": "eggs"
    }
  ]
}
```

Then, let's define the POJO class with `@DataFile` annotation

```
@Data
@DataFile
static class TestUsers {
    List<User> users;
}
```

Finally, let's create our User class:

```
@Data
static class User {
    private String username;
    private String password;
}
```

We're going to use `DataMapper` to read our JSON file into an `TestUsers` object, so let's set that up now

```
final TestUsers testUsers = DataMapper.parse(TestUsers.class);
```

We'll find that our `TestUsers` object is populated from the file, including the list of `User`.

Here is full example code to demonstrate to parse and read `test_users.json` file

```

public class DataMapperTest {
    @DataProvider
    public Iterator<Object[]> getTestUsers() {
        final TestUsers testUsers = DataMapper.parse(TestUsers.class);
        final List<Object[]> data = new ArrayList<>();
        testUsers.getUsers()
            .forEach(user -> data.add(new Object[]{user}));
        return data.iterator();
    }

    @Test(dataProvider = "getTestUsers")
    public void jsonTest(User user) {
        Assert.assertFalse(user.getUsername().isBlank());
        System.out.println("Username[" + user.getUsername() + "] Password[" + user
            .getPassword() + "]");
    }

    @Data
    @DataFile
    static class TestUsers {
        List<User> users;
    }

    @Data
    static class User {
        private String username;
        private String password;
    }
}

```

### 3.2.3. Update data files

We're also going to use DataMapper to write a TestSample out to a file. This time we will try using yaml file.

Let's quickly look at the `test_sample.yml` file we'll be writing the values

```

users:
  user1:
    username: "Ramesh"
    password: "4177472e-23a3-4426-893f-8a794af7189c"
  user2:
    username: "Babu"
    password: "40aafad2-1d24-4d6c-85e2-b7630dc17c57"

```

Then, let's define the POJO class with `@DataFile` annotation

```

@Data
@DataFile(fileName = "test_sample.yml")
static class TestSample {
    Map<String, Map<String, String>> users;
}

```

Let's read our TestSample and update values:

```

UUID uuid = UUID.randomUUID();
TestSample testSample = DataMapper.parse(TestSample.class);
testSample.getUsers().get("user1").put("password", uuid.toString());

```

Let's write our updated TestSample values:

```

DataMapper.write(testSample);

```

Here is full example code to demonstrate to update and write values to `test_sample.yml`

```

public class DataMapperWriteTest {
    @SneakyThrows
    @Test
    public void testClass() {
        UUID uuid = UUID.randomUUID();
        TestSample testSample = DataMapper.parse(TestSample.class);
        testSample.getUsers().get("user1").put("password", uuid.toString());
        DataMapper.write(testSample);
    }

    @Data
    @DataFile(fileName = "test_sample.yml")
    static class TestSample {
        Map<String, Map<String, String>> users;
    }
}

```



# Chapter 4. Selcukes Extent Reports

[Selcukes Extent Reports](#) is used to generate Extent reports for Cucumber JVM.

## 4.1. Features

1. Generates Emailable extent report
2. Full page screenshots attached to report as base64 format
3. Supports to add JUL based info logs to report

```
logger.info()-> "Navigated to Home page..")
```

## 4.2. Setup

Selcukes Extent Reports used as a Java dependency in Maven as follows

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>selcukes-extent-reports</artifactId>
  <version>1.6.0</version>
</dependency>
```

In the case of a Gradle project, we can declare Selcukes Extent Reports as follows:

```
dependencies {
    implementation("io.github.selcukes:selcukes-extent-reports:1.6.0")
}
```

## 4.3. Usage

Create extent.properties file in src/test/resources

```
extent.reporter.spark.start=true
extent.reporter.spark.out=target/extent-reports/Sample.html
systeminfo.Author=Ramesh
timestamp.report=false
thumbnail.report=false
```

Add Selcukes Extent Reports plugin to cucumber runner as follows

```

@CucumberOptions(tags = "@tag1", plugin = {
    "io.github.selcukes.extent.report.SelcukesExtentAdapter:",
    "html:target/cucumber-reports/cucumber.html", "json:target/cucumber-
reports/cucumber.json"

})

```

Update Cucumber Hooks file as follows

```

public class CucumberHooks {

    WebDriver driver;

    public CucumberHooks(WebDriver driver) {
        this.driver = driver;
    }

    @Before
    public void beforeTest(Scenario scenario) {
        getReport().start() //Initialise Extent Report and start recording logRecord
        .initSnapshot(driver); //Initialise Full page screenshot
        logger.info(() -> "Starting Scenario .." + scenario.getName());
        getReport().attachAndRestart(); // Attach INFO logs and restart logRecord
    }

    @BeforeStep
    public void beforeStep() {
        logger.info(() -> "Before Step");
        getReport().attachAndRestart(); // Attach INFO logs and restart logRecord
    }

    @AfterStep
    public void afterStep() {
        getReport().attachAndRestart(); // Attach INFO logs and restart logRecord
        getReport().attachScreenshot(); //Attach Full page screenshot
    }

    @After
    public void afterTest(Scenario scenario) {
        logger.info(() -> "Completed Scenario .." + scenario.getName());
        getReport().attachAndClear(); // Attach INFO logs and clear logRecord
    }
}

```

# Chapter 5. Selcukes Notifier

Selcukes Notifier helps to send notifications using Slack and Microsoft Teams.

## 5.1. Motivation

Currently, we moved to On premises environments. To view HTML report generated by automation scripts we need to connect to the different VPN's to open the report and check the related screenshots to see which test case has failed and why. So, why not raise an alert as soon as the test case fails? And why not attach a screenshot along with that alert so that all the stakeholders can actually see what has failed in the application?

## 5.2. Setup

Selcukes Notifier used as a Java dependency in Maven as follows

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>selcukes-notifier</artifactId>
  <version>1.6.0</version>
</dependency>
```

In the case of a Gradle project, we can declare Selcukes Notifier as follows:

```
dependencies {
    implementation("io.github.selcukes:selcukes-notifier:1.6.0")
}
```

## 5.3. Usage

Add selcukes.yaml and update below config as follows

```
notifier:
  notification: false
  type: slack
  webhook-token: WEBHOOKXXXX
  api-token: APIXXXX
  channel: selcukes
```

Create test class as follows

```
public class NotifierTest {  
    @Test  
    public void testNotifications() {  
        NotifierFactory.getNotifier()  
            .scenarioName("This is sample scenario")  
            .scenarioStatus("FAILED")  
            .stepDetails("This is sample test step")  
            .errorMessage("NullPointerException")  
            .path("")  
            .pushNotification();  
    }  
}
```

# Chapter 6. Selcukes Snapshot

[Selcukes Snapshot](#) is a utility library written in Java for creating full page screenshots using Selenium WebDriver.

## 6.1. Setup

Selcukes Snapshot used as a Java dependency in Maven as follows

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>selcukes-snapshot</artifactId>
  <version>1.6.0</version>
</dependency>
```

In the case of a Gradle project, we can declare Selcukes Snapshot as follows:

```
dependencies {
    implementation("io.github.selcukes:selcukes-snapshot:1.6.0")
}
```

## 6.2. Usage

```

@CustomLog
public class SnapshotTest {
    WebDriver driver;

    @BeforeClass
    static void setupClass() {
        WebDriverBinary.chromeDriver().setup();
    }

    @BeforeTest
    void setupTest() {
        driver = new ChromeDriver();
    }

    @AfterTest
    void teardown() {
        driver.quit();
    }

    @Test
    public void fullPageScreenshotTest() {
        final String url = "https://techyworks.blogspot.com/";

        String browser = driver.getClass().getSimpleName().replace("Driver", "");
        logger.info(() -> String.format("Initiated %s browser", browser));
        driver.get(url);
        logger.info(() -> "Navigated to " + url);
        Snapshot snapshot = new SnapshotImpl(driver);
        String screenshotFilePath = snapshot
            .withText("Browser: " + browser + "\nThis sample Text Message\nMake it
simple Make it simple Make it simple Make it simple Make it simple")
            .shootPage();
        logger.info(() -> String.format("Captured full page screenshot for %s browser
and placed at %s ",
            browser, screenshotFilePath));
    }
}

```

# Chapter 7. Selcukes Excel Runner

Selcukes Excel Runner is an Excel driven cucumber runner which helps to

1. Execute cucumber scenarios in required order
2. Execute only specific examples in a Scenario Outline

## 7.1. Setup

Selcukes Excel Runner used as a Java dependency in Maven as follows

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>selcukes-excel-runner</artifactId>
  <version>1.6.0</version>
</dependency>
```

In the case of a Gradle project, we can declare Selcukes Excel Runner as follows:

```
dependencies {
    implementation("io.github.selcukes:selcukes-excel-runner:1.6.0")
}
```

# Chapter 8. Selcukes Reports

[Selcukes Reports](#) helps to record video and send notifications on test failures

## 8.1. Setup

Selcukes Reports used as a Java dependency in Maven as follows

```
<dependency>
  <groupId>io.github.selcukes</groupId>
  <artifactId>selcukes-reports</artifactId>
  <version>1.6.0</version>
</dependency>
```

In the case of a Gradle project, we can declare Selcukes Reports as follows:

```
dependencies {
    implementation("io.github.selcukes:selcukes-reports:1.6.0")
}
```



# Chapter 9. Selcukes Configuration

## 9.1. Environment Properties

Selcukes environment properties are created using `selcukes.yaml` file

```
# Selcukes Environment Properties
projectName: Selcukes
env: Dev
proxy: false
baseUrl:
excelRunner:
  filePath: ""
  suiteName: "smoke"
web:
  remote: false
  browserName: CHROME
  headLess: true
  serviceUrl: "http://localhost:4444"
windows:
  serviceUrl: "http://127.0.0.1:4723"
  winApp-path: "C:/Program Files (x86)/Windows Application Driver/WinAppDriver.exe"
  app: "C:\\Windows\\System32\\notepad.exe"
mobile:
  serviceUrl: "http://127.0.0.1:4723"
  app: "ApiDemos-debug.apk"
video:
  recording: false
  recorderType: FFMPEG
  ffmpegPath:
  watermarkStatus: false
notifier:
  notification: false
  type: slack
  webhook-token: WEBHOOKXXXX
  api-token: APIXXXX
  channel: selcukes
```

## 9.2. Runtime Properties

The below `selcukes.properties` provides flexibility to override default environment properties in runtime.

```

selcukes.excel.runner= # true or false. default: true
selcukes.excel.suiteName= # Name of Sheet in Excel Suite File
selcukes.excel.suiteFile= # Excel File path used as excel runner
selcukes.extent.report= # true or false. default: true.
selcukes.features= # comma separated paths to feature files. example:
path/to/example.feature, path/to/other.feature
selcukes.tags= # tag expression. example: @smoke and not @slow
selcukes.glue= # comma separated package names. example:
com.example.glue
selcukes.plugin= # comma separated plugin strings. example: pretty,
json:path/to/report.json
selcukes.reports.path= # path/target
selcukes.timestamp.report= # true or false. default: false

```

## 9.3. Logger Properties

```

# To add the FileHandler, use the following line.
handlers: java.util.logging.FileHandler, java.util.logging.ConsoleHandler

#.level: INFO
.level: INFO

# For example, set the io.github.selcukes.core logger to only log SEVERE
io.github.selcukes.level: ALL
io.github.selcukes.handler: java.util.logging.ConsoleHandler

# Default file output is in user's home directory.
java.util.logging.FileHandler.pattern: target/selcukes.log
java.util.logging.FileHandler.limit: 50000
java.util.logging.FileHandler.count: 1
java.util.logging.FileHandler.formatter:
io.github.selcukes.commons.logging.SelcukesLoggerFormatter
java.util.logging.FileHandler.level: FINE

# Limit the message that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level: FINE
#java.util.logging.ConsoleHandler.formatter : java.util.logging.SimpleFormatter
java.util.logging.ConsoleHandler.formatter:
io.github.selcukes.commons.logging.SelcukesColorFormatter

```

# Chapter 10. Community

There are two ways to try to get community support related to Selcukes. First, questions about it can be discussed in [StackOverflow](#), using the tag *selcukes\_java*. In addition, comments, suggestions, and bug-reporting should be made using the [GitHub issues](#). Finally, if you think Selcukes can be enhanced, consider contributing to the project through a [pull request](#).

# Chapter 11. Support

[Selcukes](#) is part of [OpenCollective](#), an online funding platform for open and transparent communities. You can support the project by contributing as a backer (i.e., a personal [donation](#) or [recurring contribution](#)) or as a [sponsor](#) (i.e., a recurring contribution by a company).