

How to test mobile devices using Selendroid and Appium

(Version 1.1 - August 2015)

[1. Preface](#)

[2. Requirements](#)

[2.1 Java SDK](#)

[2.2 Android SDK](#)

[2.3 Android Virtual Devices](#)

[2.4 Physical Android Devices](#)

[3. Preparing Your Device](#)

[4. Selendroid](#)

[4.1 What is Selendroid?](#)

[4.2 Starting Selendroid](#)

[4.3 Writing a Simple Test for Selendroid](#)

[4.4 Running a Test](#)

[4.5 Starting Devices At Runtime](#)

[4.5.1 Selecting AVDs by Properties](#)

[4.5.2 Selecting Physical Devices By Properties](#)

[4.5.3 Performing a Test On Multiple Emulators](#)

[4.6 Selendroid and SiteGenesis](#)

[4.8 Advantages](#)

[5. Appium](#)

[5.1 What is Appium?](#)

[5.2 Installing the Appium Server](#)

[5.3 Creating Your First Appium Test Case](#)

[5.3.1 Setup of a Simple Appium Project](#)

[5.3.2 Using Appium and Maven](#)

[5.3.3 Writing Your First Appium Test Case](#)

[5.4 Test Execution](#)

[5.5 Pitfalls](#)

[5.6 Creating An Appium Test Case For SiteGenesis](#)

[6. Local Mobile Emulators](#)

[6.1 Android Virtual Device \(AVD\) Manager](#)

[6.2 Genymotion](#)

[6.3 Other Alternatives](#)

[7. Mobile Device Clouds](#)

[7.1 AWS Device Farm](#)

[7.2 Keynote](#)

[7.3 Xamarin](#)

[7.4 Sauce Labs](#)

[7.5 BrowserStack](#)

[7.6 Perfectomobile](#)

[8. SiteGenesis Community Testsuite Wrappert](#)

[8.1 BrowserStack Standalone](#)

[8.2 Appium Standalone](#)

[8.3 Appium Wrapper](#)

[8.4 ChromeWebDriver Wrapper](#)

[9. Known Problems](#)

[9.1 Trouble Creating Screenshot With Appium](#)

[9.2 Clicking On Elements Does Not Work](#)

[9.3 Appium Buildpath Order](#)

[9.4 Windows Driver](#)

[10. Sources](#)

1. Preface

Testing native apps and the mobile web on smartphones becomes more and more important these days. To get rid of all the different platforms and application types which exists it is highly recommended to use a test automation framework like *Selendroid*¹ or *Appium*². Both frameworks are available under the *Apache License Version 2.0*. The source code can be obtained from *GitHub*.

The following text shall guide the reader through the process of installing Selendroid/Appium with all dependencies under a linux environment and finally create some test cases with the *XIt Script Developer*.

2. Requirements

Before you can start writing mobile test cases with Selendroid or Appium it is necessary to install certain dependencies. First and foremost you need to have some *Software Development Kits (SDK)*, which allows you to write tests and executes them. Additionally you need to provide some test devices. These can be either virtual devices or physical hardware devices. Make sure you fulfill all those requirements below before you proceed.

2.1 Java SDK

A *Java SDK*³ (v. 1.7 and above) needs to be installed and configured. Make sure that the environment variable *JAVA_HOME* is pointing to the right directory, i.e., do not let it point to the *Java Runtime Environment* only. If you need help installing java consult the oracle online help at https://www.java.com/en/download/help/download_options.xml, which provides further information.

¹ <http://selendroid.io/>

² <http://appium.io/>

³ <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

2.2 Android SDK

To run tests against Android devices requires you to install the *Android-SDK*⁴. Follow the steps below to setup your Android environment:

1. download the Android SDK (*SDK Tools Only*) for your operating system (<http://developer.android.com/sdk/index.html>)
2. extract the content of the downloaded archive to a any location of your choosing - the root of your user home directory is a recommended place
3. add *ANDROID_HOME* to your *PATH*

```
export ANDROID_HOME=/install_dir/android-sdk-linux
export PATH=${PATH}:$ANDROID_HOME/tools
export PATH=${PATH}:$ANDROID_HOME/platform-tools
```

4. start the *Android SDK Manager* by opening a console and typing **android** - if nothing happens your path is not configured correctly
5. in the following window you can select which platforms and tools you want to install - start by selecting the *Tools* checkbox, to download them all
6. select a checkbox beneath an Android Version - make sure that the test automation framework you want to use supports this API, for example Selendroid supports API 10 - 19
7. select the checkbox *Android Support Library* from the *Extras* folder
8. click on *Install packages...*
9. accept the license and click on *Install*

The checked packages will be downloaded, which may take a while depending on your connection. After the download is completed you can start adding virtual devices to your system or use real physical hardware to run your tests.

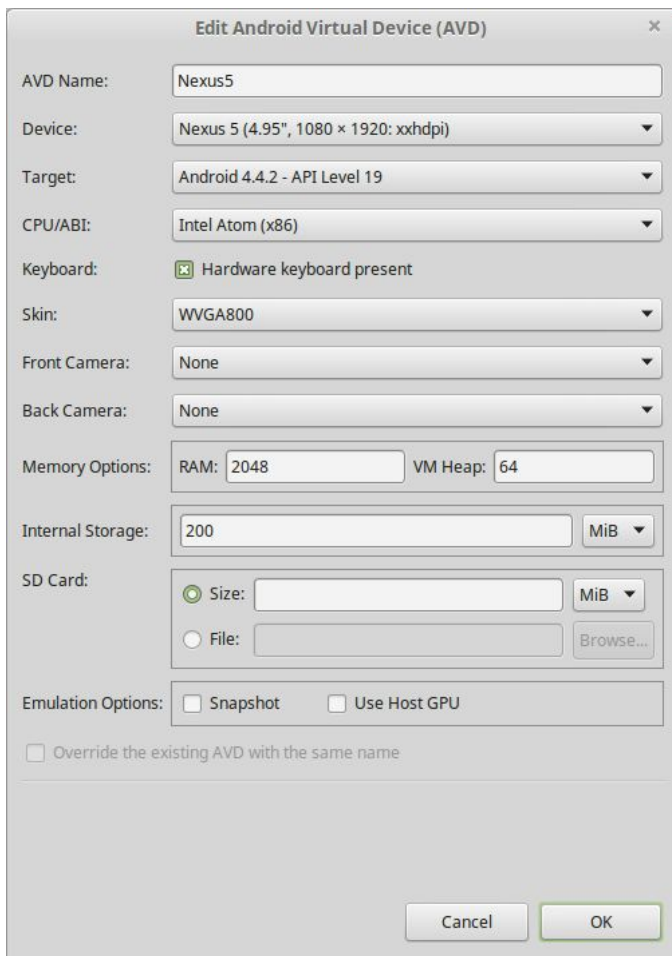
⁴ <http://developer.android.com/sdk/index.html>

2.3 Android Virtual Devices

Instead of running mobile tests on real hardware you can also use an emulator. Emulators offers some advantages compared to physical devices. You can create as many emulators as you like, each of them emulates a different Android device with different settings. An emulator for Android is often called *Android Virtual Device (AVD)*. If you have installed the Android SDK with it tools you can create your own AVD in no time.

To create a new AVD run the command `android` in a console to start the Android SDK Manager. Next go to the toolbar and click *Tools->Manage AVDs....* The *Android Virtual Device (AVD) Manager* should open in another window. Here you can create different AVDs and even specify your own Device Definitions. This allows you to create virtual devices with hardware specs that does not exist in real life. When creating an AVD pay attention to select a supported API version under *Target*, to make sure your test automation framework will recognize the device. All devices you create will be displayed in the AVD Manager. If you rather want to work on console level you can also use the command `android list avd` to get a list of all installed emulators.

IMPORTANT: do NOT activate the “Use Host GPU” option - it might cause some trouble while using the frameworks.



Edit Android Virtual Device (AVD)

AVD Name: Nexus5

Device: Nexus 5 (4.95", 1080 × 1920: xxhdpi)

Target: Android 4.4.2 - API Level 19

CPU/ABI: Intel Atom (x86)

Keyboard: ☒ Hardware keyboard present

Skin: WVGA800

Front Camera: None

Back Camera: None

Memory Options: RAM: 2048 VM Heap: 64

Internal Storage: 200 MiB

SD Card: ☒ Size: MiB ☐ File: Browse...

Emulation Options: ☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

Cancel OK

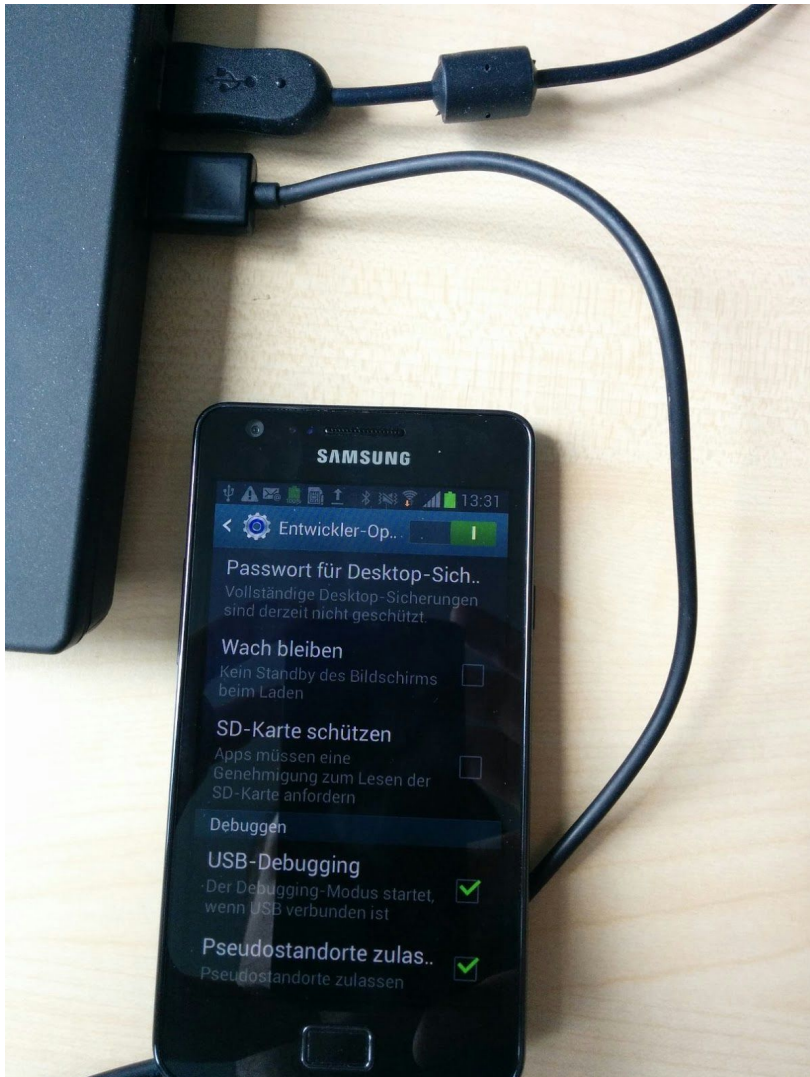
2.4 Physical Android Devices

If you don't want to rely on emulators you can also use real devices. Using physical hardware requires you to enable *USB debugging* on your phone in the **Developer Options** dialog. If you don't know where this options are follow this link <http://developer.android.com/tools/device.html#setting-up>.

Connect the device to your computer through usb. It may be necessary (depending on your Android version) to accept the fingerprint for the computer you connect to. In this case a new dialog pops up asking

you whether to trust the computer or decline the connection. Just click “Ok” and you are fine. Also make sure that no screen-lock is activated.

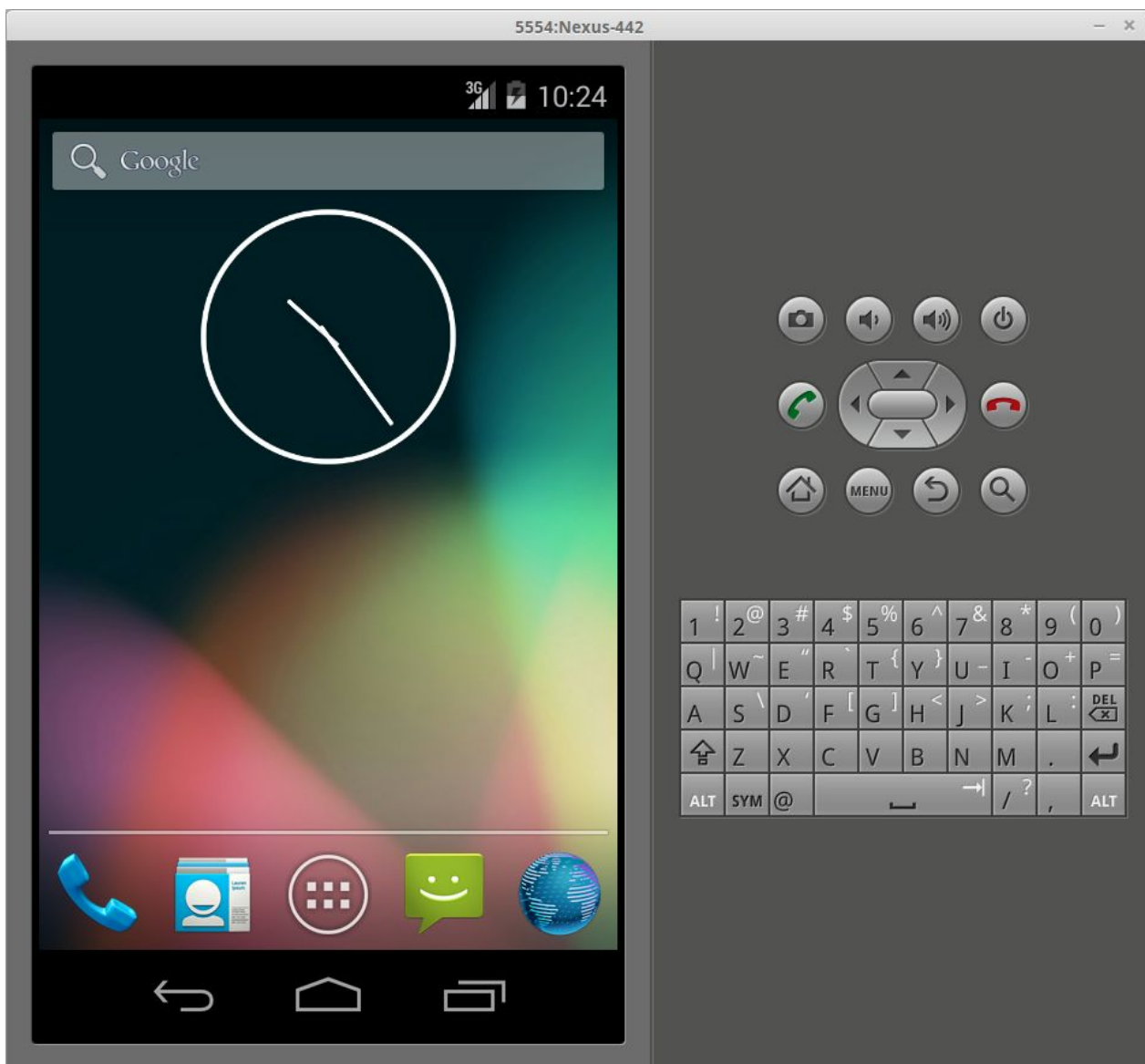
You can verify that your device is correctly attached to your computer typing the command `adb devices` in a console. The output presents a list of all external devices connected to the computer.



3. Preparing Your Device

After installing and configuring the Java- and Android SDK, you are ready to start writing tests with Selendroid or Appium. As already mentioned you can decide whether you want to execute your tests on real hardware or an emulator. Do you want to see how your test case behaves on a real device - then just follow the steps [above](#). Or do you rather prefer an emulator? All you have to do is start emulation by executing the command `emulator -avd Nexus-442` (replace Nexus-442 with the actual name of your previously created device). A new window should appear which acts like an Android device.

IMPORTANT: start all your AVDs at least once and deactivate the Android screen-lock. Otherwise there could be problems later while testing.



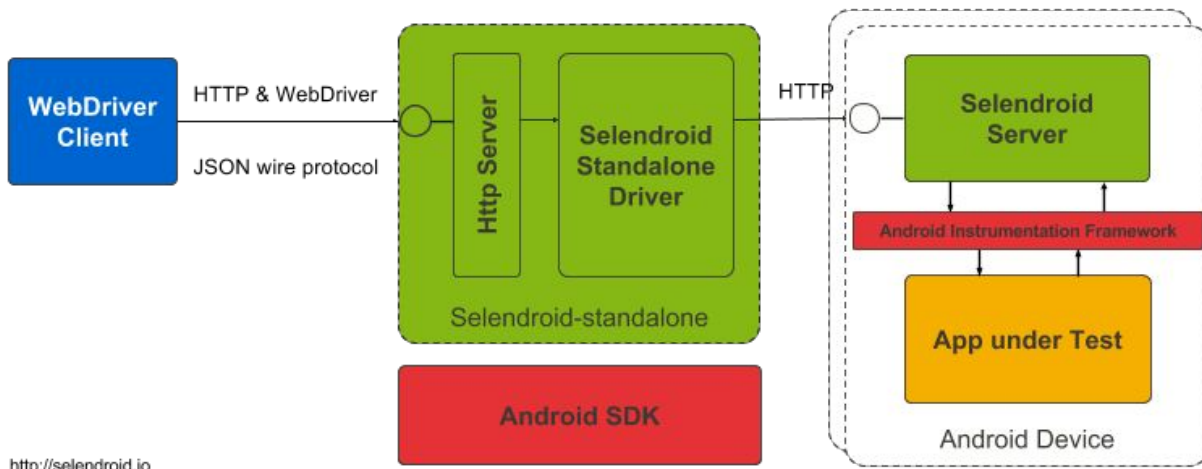
4. Selendroid

4.1 What is Selendroid?

Selendroid is a test automation framework, which can be used to test *native* and *hybrid* apps, and the *mobile web* as well (mobile web refers to accessing the internet through a mobile browser). The framework relies on the *Android Instrumentation Framework*⁵ and is therefore limited to test android devices only. Selendroid consists of four major components:

1. *Selendroid-Client* - java client library based on *Selenium WebDrivers*
2. *Selendroid-Server* - which is running on the android device
3. *AndroidDriver-App* - special app to test the mobile web (not in the picture below)
4. *Selendroid-Standalone* - proxy between Selendroid-Client and Selendroid-Server

⁵ http://developer.android.com/tools/testing/testing_android.html



Communication between the WebDriver Client and Selendroid-Standalone is based on the *JSON wire protocol*⁶. The main function of the Selendroid-Standalone is to handle requests from clients and forward them to the devices. This also includes transferring apps from the computer to the android device. It does not matter if an emulator or a real device is tested.

⁶ <https://code.google.com/p/selenium/wiki/JsonWireProtocol>

4.2 Starting Selendroid

Running Selendroid is straightforward: just download the [Selendroid-Standalone.jar](#) and you are good to go. Navigate to the folder with the jar-file and run following command:

```
java -jar selendroid-standalone-0.15.0-with-dependencies.jar
```

Executing the command will start a http server listening on port 4444. This server will scan continuously your computer for mobile device connections. Therefore it checks all your created AVDs while also watching for new plugged in devices.

Every action the server performs (like detecting new devices, running tests etc.) will be logged to the console and to disk. You can also take a look at <http://localhost:4444/wd/hub/status> to see all connected devices and some other important information. Watch out: if your device is not listed under “supportedDevices” it may be possible that it uses a not supported Android API version. In that case you cannot use it for testing purpose.

```
{
  "status": 0,
  "value": {
    "supportedApps": [
      {
        "appId": "io.selendroid.androiddriver:0.15.0",
        "mainActivity": "io.selendroid.androiddriver.WebViewActivity",
        "basePackage": "io.selendroid.androiddriver"
      }
    ],
    "os": {
      "arch": "amd64",
      "name": "Linux",
      "version": "3.13.0-37-generic"
    },
    "build": {
      "browserName": "selendroid",
      "version": "0.15.0",
      "supportedDevices": [
        {
          "screenSize": "(480, 800)",
          "platformVersion": "16",
          "model": "GT-I9100",
          "emulator": false,
          "serial": "00199148057dae"
        },
        {
          "screenSize": "(240, 320)",
          "platformVersion": "19",
          "emulator": true,
          "avdName": "My-Magic-Phone"
        },
        {
          "screenSize": "(480, 800)",
          "platformVersion": "19",
          "emulator": true,
          "avdName": "Nexus-442"
        },
        {
          "screenSize": "(240, 320)",
          "platformVersion": "19",
          "emulator": true,
          "avdName": "Nexus5-442"
        }
      ]
    }
  }
}
```

If you prefer to run a Selendroid-Standalone from your code all you have to do is create a new default `SelendroidConfiguration` and launch a server with it. Take a look at the code snippet below, which shows you how it works:

```
public void startSelendroidServer()
{
    SelendroidConfiguration config = new SelendroidConfiguration();

    //selendroidServer is an instance variable
    selendroidServer = new SelendroidLauncher(config);
    selendroidServer.launchSelendroid();
}
```

As you can see it is really simple to start Selendroid-Standalone from your code. Of course you can change the server-configuration to your needs, like specifying on which port the is listening, the log-level or some general behaviour, e.g., emulator- and session timeout.

4.3 Writing a Simple Test for Selendroid

You can write Selendroid test cases in every programming language for which Selenium offers client bindings. The following test code is written in Java and demonstrates the simple usage of Selendroid. First, create a new Java Project in the IDE of your choice. Add a new class with the name **SelendroidTest** to the project which contains the following code:


```

import io.selendroid.client.SelendroidDriver;
import io.selendroid.common.SelendroidCapabilities;
import io.selendroid.standalone.SelendroidLauncher;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;

public class SelendroidTest {

    SelendroidLauncher selendroidServer;

    WebDriver driver;

    @Before
    public void begin() throws Exception
    {
        DesiredCapabilities caps = SelendroidCapabilities.android();
        driver = new SelendroidDriver(caps);
    }

    @Test
    public void test()
    {
        driver.get("http://m.ebay.de");
        WebElement element = driver.findElement(By.id("kw"));
        element.sendKeys("Nexus 5");
        element.submit();
    }

    @After
    public void end()
    {
        if(driver != null)
        {
            driver.quit();
        }
    }
}

```

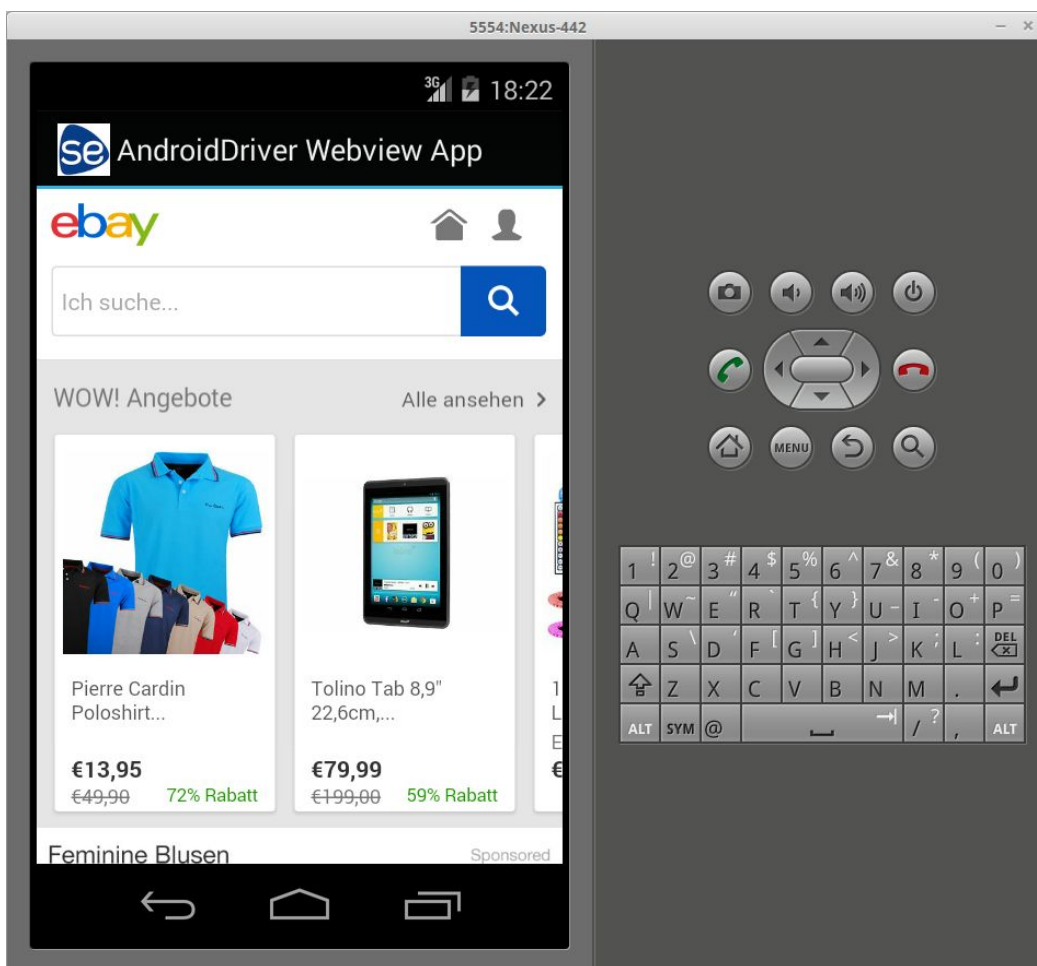
The class contains three methods that are executed before, during and after the test. In `begin()` a new webdriver is created with the capability to communicate with Android devices. The method `test()` provides the code which will be tested. In this case it opens a mobile webbrowser, visit the website <http://m.ebay.de>, get a handle to the search field and searches for the text "Nexus 5". After the payload of the test `end()` is called and the browser is closed.

Before you can run the code with Selendroid you need to add [selendroid-client-0.15.0.jar](#) and [selendroid-standalone-0.15.0-with-dependencies.jar](#) to your project build path.

4.4 Running a Test

To run the code above you need to a.) start an AVD as explained in [3. Preparing your device](#) or b.) connect your smartphone to the computer as explained in [2.4 Physical Android Devices](#).

Take care: if you have real hardware connected and an AVD running, Selendroid will pick up one object randomly to perform the test. Make sure Selendroid-Standalone is started (as described in [4.2 Starting Selendroid](#)) and recognizes your device. All that is left to do is to compile and run the code and watch your device performing the test.



4.5 Starting Devices At Runtime

Testing your code in an emulator as presented in [4.4 Running a test](#) has one major drawback: you need to start/stop your AVD manually before the test, every time you want to use another device. Imagine you have a test case and around ten devices which you all wanted to test. It would be a really tedious job to type always `emulator -avd device#xx` just to start a new device and afterwards execute the actual test. Selendroid offers a convenient way to solve this specific problem and saves you a lot of typing. Instead of starting emulators by hand you can let Selendroid start/stop them for you dynamically (do not confuse

starting an AVD with creating one!). If you have just one emulator/device or it does not matter on which device you are performing a test you can stick with the code from [4.3 Writing a simple test for Selendroid](#). But if you want to test certain devices you need to supply additional information to Selendroid. These information are provided through *properties* during the creation of the webdriver. Selendroid-Standalone will then try to select a device which matches all the requirements. Take a look at the table below to see all the properties you can set for mobile web testing.

There are also some pitfalls by selecting devices which you need to take care of. A device must support ALL the specified properties to return a match, i.e., if your device has the Android target API 19 but the property is telling Selendroid-Standalone to look for devices with API 18, no match will be found and the test will crash. On the other hand, if your properties match various results, Selendroid will pick the first one returned.

Sadly there is no way to select an emulator by name or ID. Only physical devices can be accessed by a serial number.

Property	Description
EMULATOR	boolean value, indicating whether to use an emulator or a real device
PLATFORM_VERSION	the Android target API version
LOCALE	which localization should be used
SCREEN_SIZE	the display resolution
MODEL	specify the model of an emulator
API_TARGET_TYPE	let you specify the API library needed on a device
DISPLAY	only for Linux: specify the display on which the emulator window should appear
PRE_SESSION_ADB_COMMANDS	allows the user to provide a list of adb commands (executed before the test)

4.5.1 Selecting AVDs by Properties

Letting Selendroid pick an emulator for your desired properties is straightforward. The example below shows you how to select a device by using some of the properties from the table above. All you have to do is slightly modify the code from [4.3 Writing a simple test for Selendroid](#) in the `begin()` method:

```
// replace this line...
// DesiredCapabilities caps = SelendroidCapabilities.android();

// ... with these lines
DesiredCapabilities caps = DesiredCapabilities.android();
caps.setCapability(SelendroidCapabilities.EMULATOR, true);
caps.setCapability(SelendroidCapabilities.PLATFORM_VERSION,
    DeviceTargetPlatform.ANDROID16);
caps.setCapability(SelendroidCapabilities.MODEL, "Nexus 5");
```

This sample code snippet will select a single Nexus 5 emulator (if available). If multiple devices are found the first returned to Selendroid will be used.

4.5.2 Selecting Physical Devices By Properties

Besides letting you specify which AVD to use you can also tell Selendroid which real device(s) should be tested. To get a list of all devices connected to your computer you can run the command **adb devices**. The output may look like this:

```
List of devices attached
00199148057dae          device
0149bd8205670019       device
```

Every entry has a unique serial number which identifies the device. To tell Selendroid which device to choose you can use the following code:

```
SelendroidCapabilities caps = new SelendroidCapabilities();
caps.setBrowserName("android");
caps.setPlatform(Platform.ANDROID);
caps.setSerial("00199148057dae");

driver = new SelendroidDriver(caps);
```

4.5.3 Performing a Test On Multiple Emulators

Now that you know how to select devices with Selendroid it should be no problem to write more complex code, which selects different devices consecutively and run tests on them (see the example below).

```
@Test
public void test() throws Exception
{
    String[] devices = {"Nexus 4", "Nexus 5"};
    for(String device : devices)
    {
        selectDeviceByModel(device);
        //testing code goes here ...
        if (driver != null)
        {
            driver.quit();
        }
    }
}

public void selectDeviceByModel(String model) throws Exception
{
    DesiredCapabilities caps = DesiredCapabilities.android();
    caps.setCapability(SelendroidCapabilities.EMULATOR, true);
    caps.setCapability(SelendroidCapabilities.MODEL, model);
    driver = new SelendroidDriver(caps);
}
```

4.6 Selendroid and SiteGenesis

The *SiteGenesis Community-TestSuite*⁷ allows users to write test cases for Demandware⁸-based ecommerce stores. Every test case in SiteGenesis is a subclass of **AbstractScriptTestCase**. If you want to extend the testsuite by offering the possibility to perform mobile tests you need a wrapper class. This class is responsible for starting/shutting down the Selendroid server and the creation of the webdriver. Running a mobile test just requires you to extend test cases from the newly created wrapper class.

```
@ScriptName
("tests.search.TSearchArticles_ArticlesOnly")
//public class TSearchArticles_ArticlesOnly extends AbstractScriptTestCase
public class TSearchArticles_ArticlesOnly extends MobileWrapper
{
}
```

In the sample above the class **MobileWrapper** acts as a wrapper class for mobile test cases. A possible implementation of that class may look like this:

⁷ <https://github.com/Xceptance/SiteGenesis-Community-TestSuite>

⁸ <http://www.demandware.com/>


```

public class MobileWrapper extends AbstractScriptTestCase
{
    private static SelendroidLauncher selendroidServer = null;
    private static WebDriver driver = null;

    static
    {
        Runtime.getRuntime().addShutdownHook(
            new Thread(){
                @Override
                public void run(){
                    MobileWrapper.quitDriver();
                    MobileWrapper.stopSelendroidServer();
                }
            }
        );
        startSelendroidServer();
        createAndroidDriver();
    }

    public MobileWrapper()
    {
        setWebDriver(driver);
    }

    public static void createAndroidDriver()
    {
        final DesiredCapabilities caps = SelendroidCapabilities.android();
        try
        {
            driver = new SelendroidDriver(caps);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void startSelendroidServer()
    {
        if(selendroidServer != null)
        {
            selendroidServer.stopSelendroid();
        }

        final SelendroidConfiguration config = new SelendroidConfiguration();
        config.setPort(4444);

        selendroidServer = new SelendroidLauncher(config);
        selendroidServer.launchSelendroid();
    }
}

```



```

public static void quitDriver()
{
    if(driver != null)
    {
        driver.quit();
    }
}

public static void stopSelendroidServer()
{
    if(selendroidServer != null)
    {
        selendroidServer.stopSelendroid();
    }
}
}

```

The purpose of this class is to start/stop a Selendroid-Server and create/delete an Android-Driver. To save time and resources the server and driver are only created/deleted once. The creation will take place in a static initialization block which gets executed when the Java Classloader loads the class. In this block we also add a shutdown handler that will be called when all tests are finished and the *Java Virtual Machine (JVM)* quits.

Because it is necessary to tell the superclass which driver should be used, we need to call `setWebDriver()` in the constructor and pass the driver for each test we like to run.

4.7 Known Problems

At the time of writing selendroid has some issues which have a negative impact on performing tests and may also crash them. The following problems were found by the author during writing this documentation:

Issue #1 : It is not possible to override certain JavaScript functions like window.alert or window.confirm

The reason for that is that Selendroid uses its own messaging system which already overrides those functions. As a result the Selendroid-Server may crash. Selendroid devs know about this issue but it will probably not be fixed soon (<https://github.com/selendroid/selendroid/issues/361>).

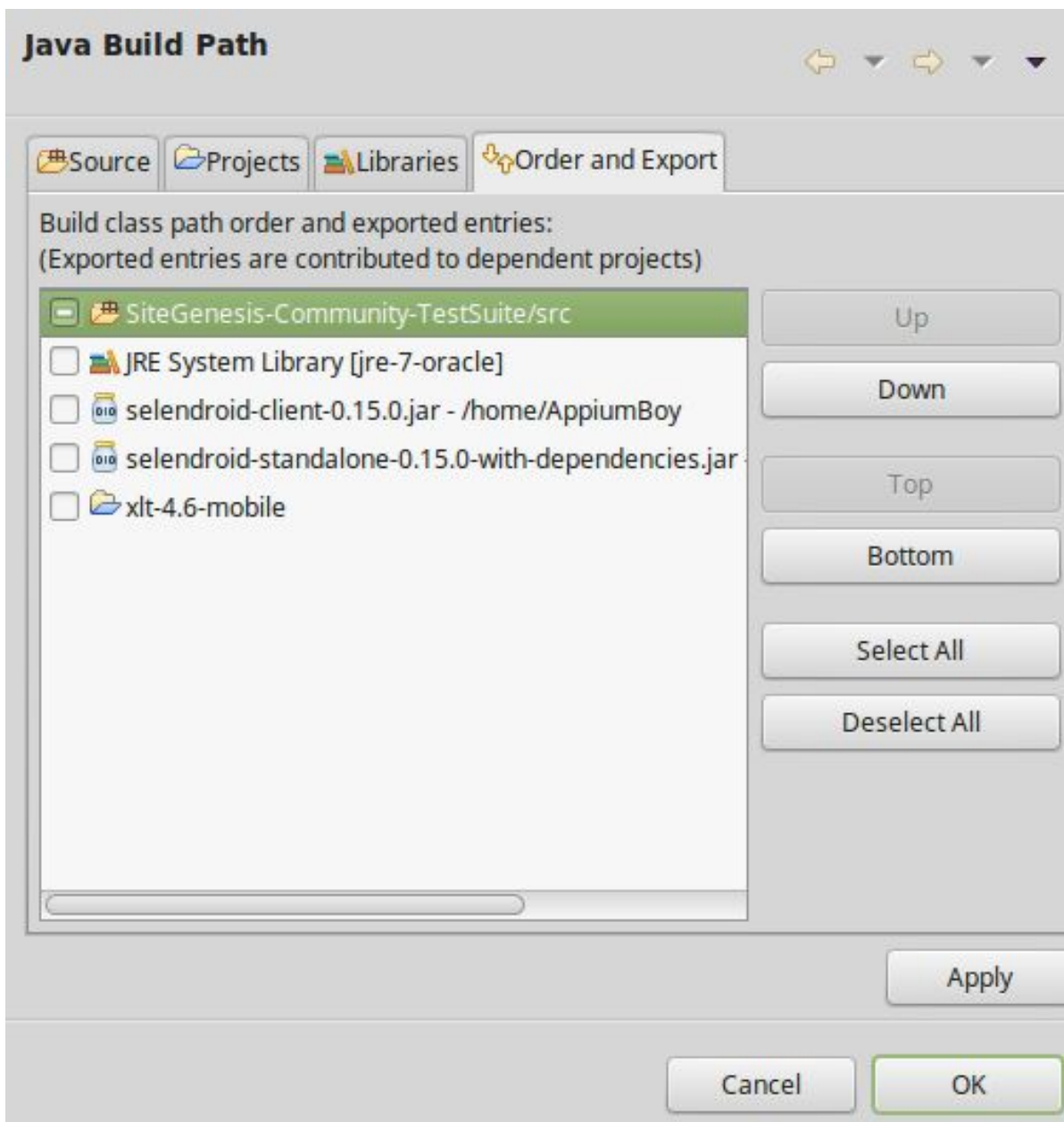
This problem especially affects the XLT-Framework method `assumeOkOnAlertOrConfirm` in the class `WebDriverUtils`.

Issue #2 : The method clear() cannot be used

There are some problems when the selendroid webdriver is created and the Selenium function `clear()` is called. It is not quite clear how and when the error happens (maybe it depends on a certain Android version). For further reading visit <https://github.com/appium/appium/issues/3186> or <https://github.com/selendroid/selendroid/issues/492>.

Issue #3: The webviews gets opened but then closed immediately

If you want to use Selendroid in conjunction with SiteGenesis you need to pay attention to the order of the build path. SiteGenesis uses some libraries which are also required by Selendroid. This can lead to version conflicts or other unexpected behaviour. For example, take a look at the graphic below:



The graphic shows the build classpath order for the authors SiteGenesis project. It uses the libraries required for Selendroid and links to the sources of the XLT-Framework (in the project xlt-4.6-mobile). If you change the order and put the folder named “xlt-4.6.-mobile” above the entry selendroid-standalone-0.15.0-with-dependencies.jar the creation of a webdriver fails. This kind of error is hard to detect because the console log does not tell you exactly what the problem is. In this example all the error log says is that there are some problems populating the webdriver with JSON-Objects. So if there are some strange errors in your project try to change the build order first.

4.8 Advantages

The framework offers many advantages, most of them are related to the functionality to test apps. To get an overview take a quick glimpse at the following list:

- you can test apps without modifying them
- support of multiple devices/simulators simultaneously
- gesture support (touch, swipe, drag ...)
- can handle hot-plug hardware

- supports Android API 10 (2.3.3 GingerBread) to 19 (4.4 KitKat)
- provides some built in UI inspector tools

5. Appium

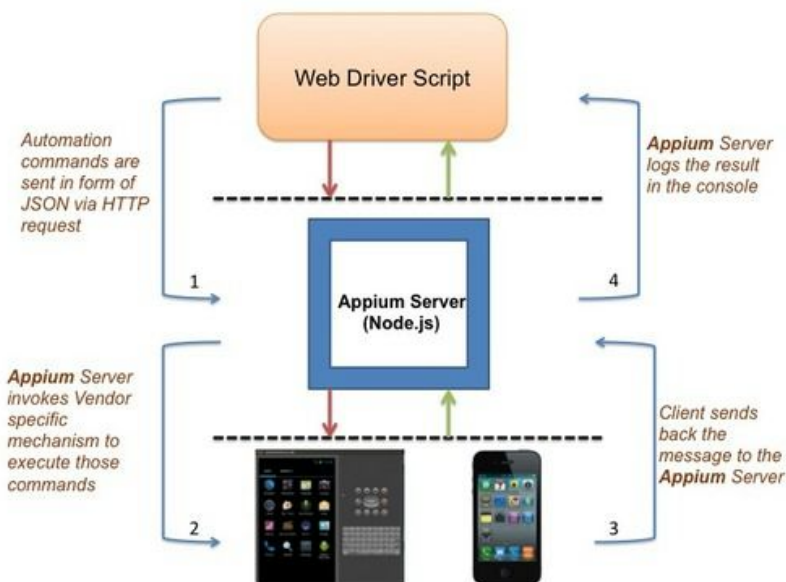
5.1 What is Appium?

Appium is another open source test automation framework for testing the mobile web, native- and hybrid apps. The framework was designed to support cross-platform tests. That means a user can write tests for Android, iOS or even FirefoxOS using the same API. This works because Appium is using different vendor-provided frameworks for different platforms:

- iOS: Apple's [UIAutomation](#)
- Android 4.2+: Google's [UiAutomator](#)
- Android 2.3+: Google's [Instrumentation](#)

Those vendor-provided frameworks are wrapped in the WebDriver API so that the user is not bound to certain framework restrictions. For example, if the user wants to use UIAutomation he is forced to write his tests in JavaScript. By encapsulating the frameworks the user write his tests in every language which is WebDriver-compatible.

The way that Appium works is very similar to Selendroid. A client sends commands to an Appium Server, which translates these commands into vendor specific instructions and forwards them to an emulator or real device. The device executes the instructions and replies to the server with the result of the performed actions. The server takes the result, print it on screen and send it back to the original client.



One important thing to mention is that Appium only runs for Android Devices using API 17 and above. All devices connected using a lower API version will be run using Selendroid.

5.2 Installing the Appium Server

Before you install Appium make sure that you fulfill all requirements from [2. Requirements](#) and prepared your AVDs like described in [3. Preparing your device](#).

Appium can be obtained as *Node.js*⁹-package through the *Nodes.js package manager (npm)*¹⁰. Just enter the following commands in your console:

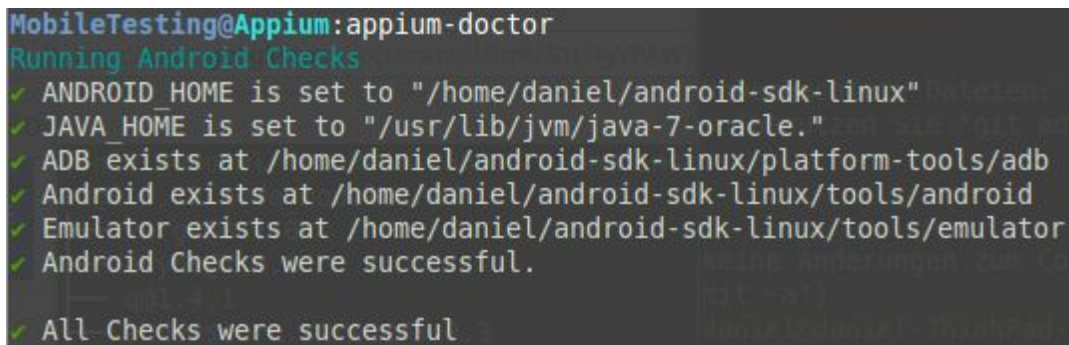
⁹ <https://nodejs.org/>

¹⁰ <https://www.npmjs.com/>

```
npm install -g appium
npm install wd
```

It is important that you do not use `sudo` to install or start Appium otherwise Appium will complain that it cannot run. If you have already installed node.js and are not allowed to install/start something without entering `sudo` you need to uninstall node.js first. Afterwards download the node.js archive for your operating system from <https://nodejs.org/download/>, extract it and configure your PATH. Now you should be able to install/start node-packages without `sudo`.

After the installation you can use the command `appium-doctor` to verify that Appium's dependencies are installed correctly.



```
MobileTesting@Appium:~$ appium-doctor
Running Android Checks
✓ ANDROID_HOME is set to "/home/daniel/android-sdk-linux"
✓ JAVA_HOME is set to "/usr/lib/jvm/java-7-oracle."
✓ ADB exists at /home/daniel/android-sdk-linux/platform-tools/adb
✓ Android exists at /home/daniel/android-sdk-linux/tools/android
✓ Emulator exists at /home/daniel/android-sdk-linux/tools/emulator
✓ Android Checks were successful.

✓ All Checks were successful
```

At the time of writing Appium 1.4 is the latest version. You can check which Appium version is installed by entering `appium --version`.

5.3 Creating Your First Appium Test Case

You are almost ready to create your first Appium Test Case. All that is missing now is the client code for setting up an Appium Driver. Depending on whether you want to create a small and simple project or rather create a project with the build manager Maven¹¹ you can continue reading at [5.3.1 Setup of a Simple Appium Project](#) or [5.3.2 Using Appium and Maven](#).

5.3.1 Setup of a Simple Appium Project

Writing an Appium Application without a build manager is straightforward. At first create a new Java Project - let's call it Appium. Download the latest version of Appium's Client Jar which is available under Appium's Maven repository¹². Also make sure to download the recommended Selenium version for the Appium client. Instead of downloading Selenium and all its dependencies manually you can also just download the whole package *Selenium Standalone Server*¹³. That .jar file contains all additional dependencies which you need to add manually otherwise. Add the Appium's Client Jar and the Selenium Server Jar files to your build path. Now your project is setup and you can continue reading [5.3.3](#).

¹¹ <https://maven.apache.org/>

¹² <http://mvnrepository.com/artifact/io.appium/java-client>

¹³ <http://selenium-release.storage.googleapis.com/2.46/selenium-server-standalone-2.46.0.jar>



5.3.2 Using Appium and Maven

Instead of creating a simple single project and downloading and adding all dependencies manually to the buildpath you can also use Maven as software project manager. In fact all sources and jars are available in the Maven repository from Appium <http://mvnrepository.com/artifact/io.appium>. All you need to do is to create a file *pom.xml* that contains all dependencies required by Appium.

```
<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
    http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.xceptance</groupId>
  <artifactId>appium-test</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>io.appium</groupId>
      <artifactId>java-client</artifactId>
      <version>3.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>2.46.0</version>
    </dependency>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.2.4</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.3.3</version>
    </dependency>
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>17.0</version>
    </dependency>
    <dependency>
      <groupId>cglib</groupId>
      <artifactId>cglib</artifactId>
      <version>3.1</version>
    </dependency>
  </dependencies>
</project>
```

5.3.3 Writing Your First Appium Test Case

If your project setup is completed you can start to write your first test case. It does not matter if you want to use *JUnit*¹⁴ for testing or just write a few lines of code because Appium is decoupled from JUnit since version 3.0. The code below is a quick and dirty example of how to use Appium to test the mobile web.

¹⁴ <http://junit.org/>

```

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.remote.MobileBrowserType;
import io.appium.java_client.remote.MobileCapabilityType;
import io.appium.java_client.remote.MobilePlatform;

import java.net.MalformedURLException;
import java.net.URL;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;

public class Appium
{
    public static void main(String[] args) throws MalformedURLException
    {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps = DesiredCapabilities.android();

        caps.setCapability(MobileCapabilityType.DEVICE_NAME, "Nexus5");
        caps.setCapability(MobileCapabilityType.PLATFORM_NAME, MobilePlatform.ANDROID);
        caps.setCapability(MobileCapabilityType.BROWSER_NAME, MobileBrowserType.BROWSER);

        URL appiumUrl = new URL("http://127.0.0.1:4723/wd/hub");

        WebDriver driver = new AndroidDriver<WebElement>(appiumUrl, caps);
        driver.get("http://m.ebay.de");

        final WebElement element = driver.findElement(By.id("kw"));

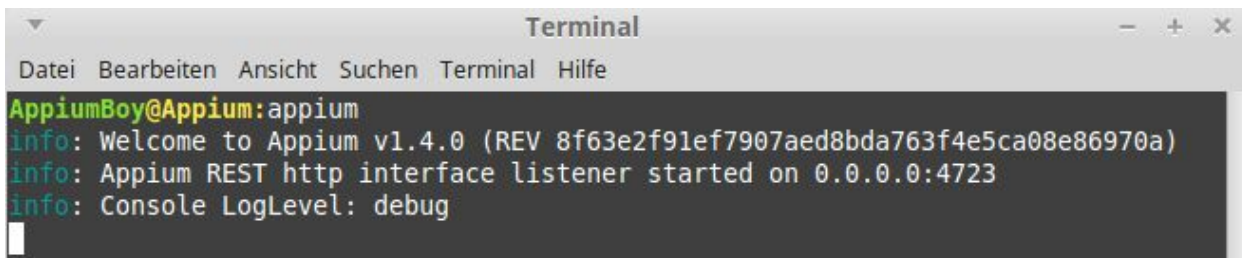
        element.sendKeys("Nexus 5");
        element.submit();

        if(driver != null)
        {
            driver.quit();
        }
    }
}

```

5.4 Test Execution

Before you can start your test you need to start the Appium server by typing **appium** in your console. The Appium server is started and begins to listen:



```

Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
AppiumBoy@Appium:appium
info: Welcome to Appium v1.4.0 (REV 8f63e2f91ef7907aed8bda763f4e5ca08e86970a)
info: Appium REST http interface listener started on 0.0.0.0:4723
info: Console LogLevel: debug

```

Now you can either connect your phone via USB to your computer OR start an AVD (for example, typing **emulator -avd Nexus-442** will start the Virtual Device with the name Nexus-442). Pay attention: One Appium server cannot handle multiple devices at the same time. It is possible to test against multiple devices at the same time but this requires running multiple instances of Appium servers (each listening to

another port). Executing the following two lines will start two separated Appium servers, each on a different port, using the devices specified by the parameter passed in within U.

```
appium -p 4727 -bp 4728 -U "1af519b642187c" --chromedriver-port 9516  
appium -p 4725 -bp 4731 -U "98531da52fc10e" --chromedriver-port 9518
```

Wait until the AVD is started or your connect device gets recognized. You can verify that your device is ready for testing by entering **adb devices** in your console. Again, if you only have a single Appium server started make sure that only one device is listed as output. When anything is prepared you can start your test case and see how Appium executes the commands on your phone.

5.5 Pitfalls

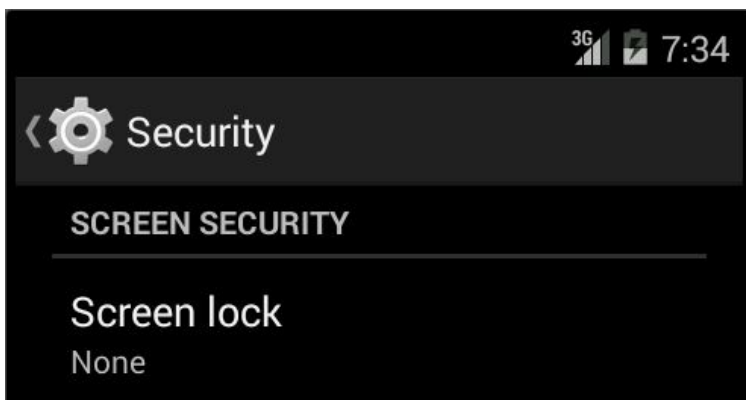
Appium also contains some pitfalls which may give you some headache. If something does not work take a look at the list below.

Issue #1: Phone gets unlocked but nothing happens afterwards

Appium message:

“Failed to start an Appium session, err was: Error: An operation did not complete before its timeout expired. (Original error: timeout: Adb command timed out after 30 seconds)”

You have started Appium, booted your AVD (or connected your phone) and are ready to proceed testing. After starting your test you see that the device is unlocked but nothing happens afterwards. Maybe you see a white window which opens up but closes immediately or you see how the browser is opened but then again, nothing happens. Appium tries to perform further commands (like opening a website) but it cannot proceed. This error is probably related to a well known problem discussed in the Appium developer forum¹⁵. Avoiding this problem requires you to set your phone setting “Screen lock” to “None”. That may not be the best solution regarding that the initial problem still exists (Appium cannot handle phone unlocking properly) but it will help you to continue with mobile testing.



5.6 Creating An Appium Test Case For SiteGenesis

If you are familiar with the *SiteGenesis-Community-TestSuite*¹⁶(SGCTS) then you can of course create your own own mobile test cases which are executed by Appium and test SiteGenesis sites. Integrating Appium into the SGCTS requires you to write an additional wrapper that creates the Appium driver.

Usually SGCTS test cases extends **AbstractScriptTestCase**. So what you gonna need to do is write a further wrapper that extends **AbstractScriptTestCase**. The following class could be used for mobile testing:

¹⁵ <https://github.com/appium/appium/issues/2908>

¹⁶ <https://github.com/Xceptance/SiteGenesis-Community-TestSuite>

```

public class AppiumWrapper extends AbstractScriptTestCase
{
    private static final AppiumDriver<WebElement> driver;

    static
    {
        Runtime.getRuntime().addShutdownHook
        (
            new Thread(){
                @Override
                public void run()
                {
                    AppiumWrapper.quitDriver();
                }
            }
        );

        DesiredCapabilities caps = new DesiredCapabilities();
        caps = DesiredCapabilities.android();
        caps.setCapability(MobileCapabilityType.DEVICE_NAME, "Nexus5");
        caps.setCapability(MobileCapabilityType.PLATFORM_NAME, MobilePlatform.ANDROID);
        caps.setCapability(MobileCapabilityType.BROWSER_NAME, MobileBrowserType.BROWSER);

        URL appiumUrl = null;

        try
        {
            appiumUrl = new URL("http://127.0.0.1:4723/wd/hub");
        }
        catch (MalformedURLException e)
        {
            e.printStackTrace();
        }

        driver = new AndroidDriver<>(appiumUrl, caps);
    }

    public AppiumWrapper()
    {
        setWebDriver(driver);
    }

    public static void quitDriver()
    {
        System.out.println("Quit Driver");
        if(driver != null)
        {
            driver.quit();
        }
    }
}

```

All that is left to do now is let your current test case extends the newly created AppiumWrapper:

```

@ScriptName
("tests.checkout.phone.TCheckoutOrder_AsGuest")
//public class TCheckoutOrder_AsGuest extends AbstractScriptTestCase
public class TCheckoutOrder_AsGuest extends AppiumWrapper
{
}

```

6. Local Mobile Emulators

The following section shows you which tools you can use for creating local emulators.

6.1 Android Virtual Device (AVD) Manager

As already mentioned in [2.3 Android Virtual Devices](#) you can create emulators using the Android Virtual Device Manager that comes with the Android SDK. Creating devices is straightforward: just click on the button “Create ...” and select which device, API version and skin you want to use. Keep in mind that before you can actually use an emulator, you need to install the required API version for the specific device.



Pros

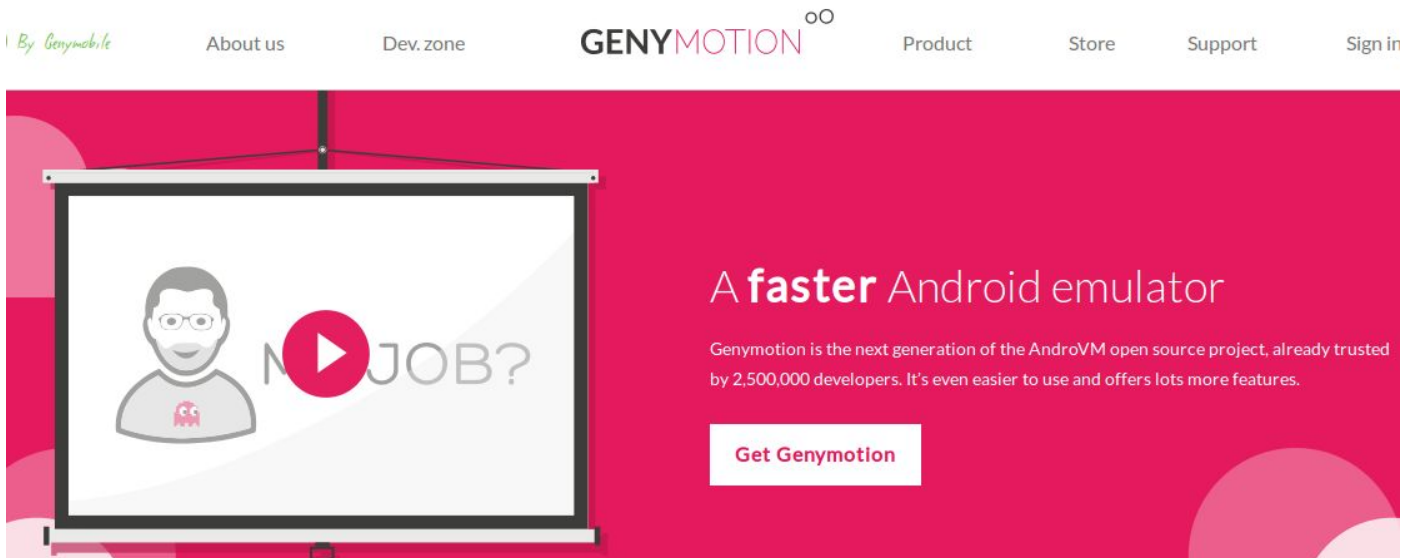
- ✓ it is free
- ✓ create as many devices as you want
- ✓ contained in the Android SDK
- ✓ you can create your own devices

Cons

- ✗ not able to process multiple test cases in a row on the same simulator
- ✗ you need to download/setup the API's by yourself
- ✗ no pre-configured devices
- ✗ if you do not want create your own imaginary hardware, you are limited to just a few standard devices (Nexus)
- ✗ VM's may took a huge amount of space

6.2 Genymotion

Genymotion is an Android emulator that is available at <https://www.genymotion.com/#/> and lets you create several android devices. As a private user you can own a copy for personal use only. That kind of copy has some limitations (missing screencast, no clone and reset ...). If you do not want these limitations and are willing to pay 24.99 € per month you can buy a business license for Genymotion with all features.



Pros

- ✓ can run multiple test cases in a row on the same device without crashing (in contrast to AVD)
- ✓ it is "Insanely fast and fully featured"¹⁷ (compared to physical devices)
- ✓ brings its own ADB tools

Cons

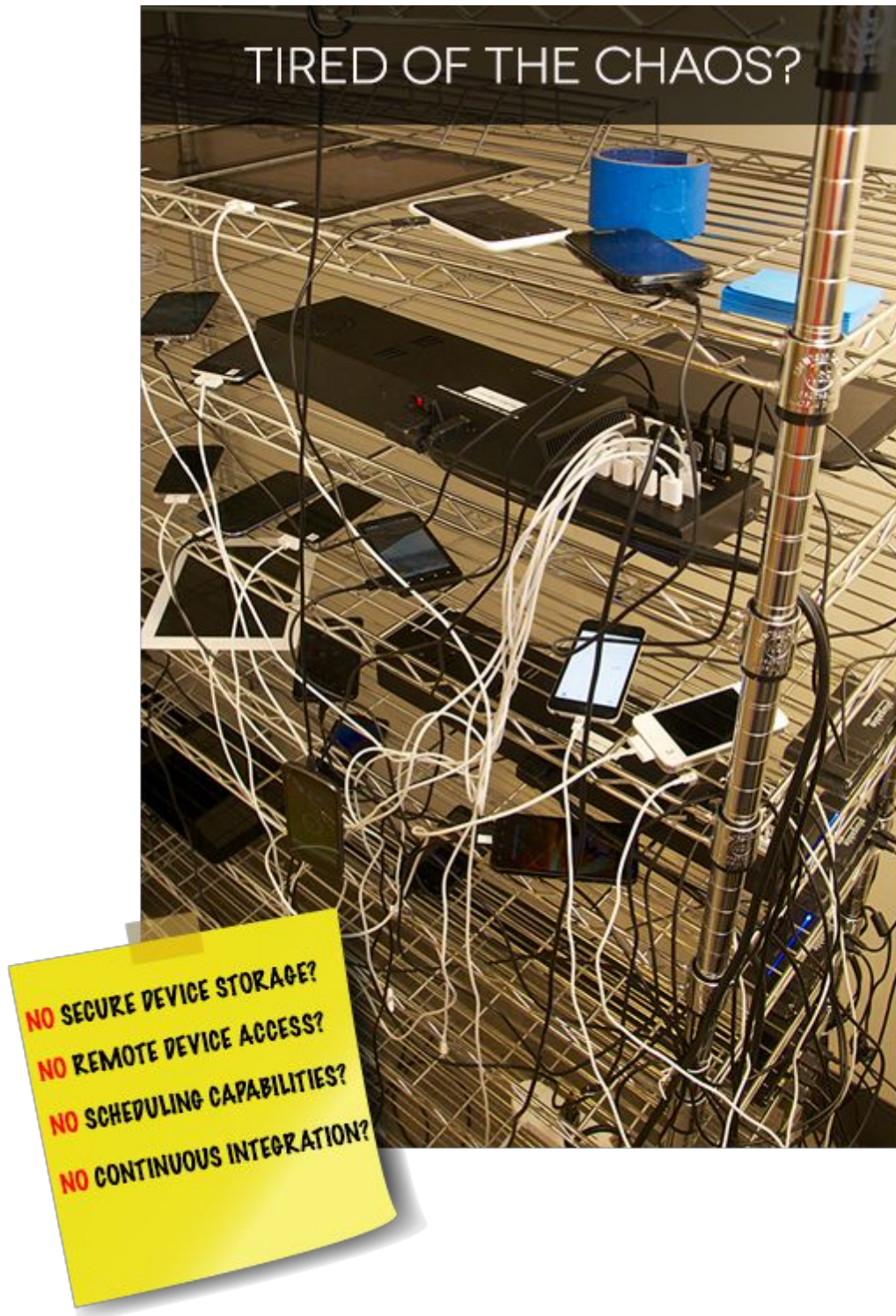
- ✗ monthly fee (business license)

6.3 Other Alternatives

There are a few more alternatives you can use to run an Android emulator locally but they all seem to have certain limitations or flaws (operating system, android version, still in a beta version...), so they are probably not suitable for testing. Anyway, if you want to check some of them out take a look at the following links: [You Wave](#), [Jar Of Beans](#) and [Andyroid](#).

¹⁷ Quote from the genymotion website

7. Mobile Device Clouds



Some drawbacks of using local emulators are that you need to manage all devices by yourself, install the required API's and the extra stress for your CPU. If you do not want to set up emulators or real devices yourself you could also rent them online. A lot of vendors offer such devices for executing mobile tests. Of course, all these solutions are System as a Service solutions, so none of them is free. The following section give you an overview about some cloud mobile renting services from which you may choose.

7.1 AWS Device Farm

Amazon offers you to test your Android and Fire OS apps in their cloud. All tests are executed on real devices in parallel. You can either choose to perform your the test on a smartphone or on a tablet.

Website	http://aws.amazon.com/de/device-farm/
Suitable for	native apps, hybrid apps
Physical Dev.	yes (> 200)
Virtual Dev.	no
Support	Appium, Calabash
Pricing	0,17 USD per device/min. or a monthly flat for 250 USD
Information	<p>Can I use AWS to test the mobile web as well?</p> <p>Amazon: Hi, thanks for the question. Today, AWS Device Farm supports testing of native and hybrid apps. For web applications, this means you can test the app within a thin container (WebView or similar) or in an instrumented build of a browser. (17th July 2015)</p>

Additional Links:

<http://docs.aws.amazon.com/devicefarm/latest/developerguide/welcome.html>

<https://aws.amazon.com/de/blogs/aws/aws-device-farm-test-mobile-apps-on-real-devices/>

<https://forums.aws.amazon.com/thread.jspa?messageID=648896>

7.2 Keynote

Keynote is a company which is specialised in performance monitoring. They are the (self-claimed) business leader in cloud-based testing and offer customers real devices in a cloud for testing.

Website	http://www.keynote.com/solutions/testing/mobile-testing
Suitable for	native apps, hybrid apps, mobile web
Physical Dev.	yes (~ 300)
Virtual Dev.	no
Support	Appium
Pricing	between 0 - 750 \$

Information	<p>DeviceAnywhere Cloud which contains the largest mobile device library available.</p> <p>Easy to combine with Appium. Just create a device and the server sends you back an Appium-ID to use during testing.</p> <p>Requires a valid credit card for sign up</p>
--------------------	--

Additional Links:

<http://www.heise.de/developer/meldung/Keynote-Mobile-Testing-mit-Appium-Framework-gekoppelt.html>

<http://www.keynote.com/solutions/testing/appium-integration>

7.3 Xamarin

Website	http://xamarin.com/test-cloud
Suitable for	native apps, hybrid apps, mobile web
Physical Dev.	yes (> 1000)
Virtual Dev.	no
Support	Calabash
Pricing	between 1,000 - 12,000 \$
Information	Probably too expensive for customers

7.4 Sauce Labs

Website	https://saucelabs.com/mobile/
Suitable for	native apps, hybrid apps, mobile web
Physical Dev.	no
Virtual Dev.	yes (number unknown)
Support	Appium, Selendroid
Pricing	between 12 - 149 \$
Information	<p>Works very reliable</p> <p>Super slow (~ 10 minutes)</p> <p>bloated</p>

7.5 BrowserStack

Website	https://www.browserstack.com/automate
Suitable for	(hybrid apps), mobile web
Physical Dev.	yes (iOS)
Virtual Dev.	yes (> 50)
Support	Appium
Pricing	between 29 - 99 \$ (1 user/1 vm)
Information	Capability Configurator iOS and MAC support iOS has sometimes problems to start Android uses the deprecated AndroidDriver

7.6 Perfectomobile

Website	http://www.perfectomobile.com/
----------------	---

Perfectomobile offers you to rent real mobile devices hosted by organisations and/or people. You can choose from various vendors and models and if your desired device is available a connection is established. You even get the phone number for the device so that you can do phone calls and write or read SMS. That is pretty awesome because you can get instantly hundreds of new phone numbers to register yourself somewhere.

The downside of the Perfectomobile service is that customer is responsible for clearing his own data after each session. That means when you're done using the phone and disconnect, another user may re-use your previously used phone and has access to all the data you stored on which was not properly deleted. So it is up to you to wipe all data after the phone usage.

8. SiteGenesis Community Testsuite Wrappert

Below you will find different code-snippets, that demonstrates how to run test cases against mobile emulators/real devices. Some snippets are wrapper classes that allow you to run your SiteGenesis Community Testsuite test scenarios as mobile test cases.

8.1 BrowserStack Standalone

Requires:

- ❑ selenium-server-standalone-2.46.0.jar

```
public class BrowserStack
{
    public static final String USER = "your-username";
    public static final String KEY = "your-password";
    public static final String URL = "http://" + USER + ":" + KEY + "@hub.browserstack.com/wd/hub";

    public static void main(String args[]) throws Exception
    {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setCapability("browserName", "iPhone");
        caps.setCapability("platform", "MAC");
        caps.setCapability("device", "iPhone 5");

        WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
        driver.get("http://www.google.com");
        WebElement searchField = driver.findElement(By.name("q"));

        searchField.sendKeys("Testing on BrowserStack");
        searchField.submit();

        System.out.println(driver.getTitle());

        driver.quit();
    }
}
```

8.2 Appium Standalone

Requires:

- ❑ running emulator (device name "Nexus5")
- ❑ running Appium service
- ❑ java-client-3.0.0.jar
- ❑ selenium-server-standalone-2.46.0.jar

```
public class Appium30
{
    public static void main(String[] args) throws MalformedURLException
    {
        caps = new DesiredCapabilities();
        caps = DesiredCapabilities.android();
        caps.setCapability(MobileCapabilityType.DEVICE_NAME, "Nexus5");
        caps.setCapability(MobileCapabilityType.PLATFORM_NAME, MobilePlatform.ANDROID);
        caps.setCapability(MobileCapabilityType.BROWSER_NAME, MobileBrowserType.BROWSER);

        URL appiumUrl = new URL("http://127.0.0.1:4723/wd/hub");
        WebDriver driver = new AndroidDriver<WebElement>(appiumUrl, caps);
        driver.get("http://m.ebay.de");
    }
}
```



```

final WebElement element = driver.findElement(By.id("kw"));

element.sendKeys("Nexus 5");
element.submit();

if(driver != null)
{
    driver.quit();
}
}
}

```

8.3 Appium Wrapper

Requires:

- ❑ running emulator (device name "Nexus5")
- ❑ running Appium service
- ❑ SGCTS test case which extends AppiumWrapper
- ❑ java-client-3.0.0.jar
- ❑ selenium-server-standalone-2.46.0.jar

```

public class AppiumWrapper extends AbstractScriptTestCase
{
    private static final WebDriver driver;

    static
    {
        Runtime.getRuntime().addShutdownHook
        (
            new Thread()
            {
                @Override
                public void run()
                {
                    AppiumWrapper.quitDriver();
                }
            }
        );

        DesiredCapabilities caps = new DesiredCapabilities();
        caps = DesiredCapabilities.android();

        caps.setCapability(MobileCapabilityType.DEVICE_NAME, "Nexus5");
        caps.setCapability(MobileCapabilityType.PLATFORM_NAME, MobilePlatform.ANDROID);
        caps.setCapability(MobileCapabilityType.BROWSER_NAME, MobileBrowserType.BROWSER);

        URL appiumUrl = null;

        try
        {
            appiumUrl = new URL("http://127.0.0.1:4723/wd/hub");
        }
        catch (MalformedURLException e)
        {
            e.printStackTrace();
        }
        driver = new RemoteWebDriver(appiumUrl, caps);
    }

    public AppiumWrapper()
    {
        setWebDriver(driver);
    }

    public static void quitDriver()
    {

```

```

        System.out.println("Quit Driver");
        if(driver != null)
        {
            driver.quit();
        }
    }
}

```

8.4 ChromeWebDriver Wrapper

Requires:

- ❑ Chrome (Browser)
- ❑ ChromeDriver¹⁸
- ❑ SGCTS test case which extends ChromeWebDriver
- ❑ selenium-server-standalone-2.46.0.jar

```

public class ChromeWebDriverWrapper extends AbstractScriptTestCase
{
    private static WebDriver driver;

    static
    {
        Runtime.getRuntime().addShutdownHook(new Thread(){
            @Override
            public void run() {
                ChromeWebDriverWrapper.quit();
            }
        });

        System.setProperty("webdriver.chrome.driver", "/home/Appium/Downloads/chromedriver");

        Map<String, String> phone = new HashMap<>();
        phone.put("deviceName", "Google Nexus 5");

        Map<String, Object> chromeOptions = new HashMap<>();
        chromeOptions.put("mobileEmulation", phone);

        DesiredCapabilities caps = DesiredCapabilities.chrome();
        caps.setCapability(ChromeOptions.CAPABILITY, chromeOptions);

        driver = new ChromeDriver(caps);
    }

    public ChromeWebDriverWrapper()
    {
        if(driver != null)
        {
            setWebDriver(driver);
        }
    }

    public static void quit()
    {
        if(driver != null)
        {
            driver.quit();
        }
    }
}

```

¹⁸ <https://sites.google.com/a/chromium.org/chromedriver/downloads>

9. Known Problems

9.1 Trouble Creating Screenshot With Appium

Some drivers doesn't support the XLT screenshot implementation and therefore crash while taking a screenshot. To avoid that problem make sure to explicit set the driver device context to "NATIVE_APP". Make sure that you save the previously assigned value and reset the context to it, otherwise strange behavior may occur.

```
URL url = new URL("http://127.0.0.1:4723/wd/hub");
AppiumDriver<WebElement> driver = new AndroidDriver<>(url, caps);

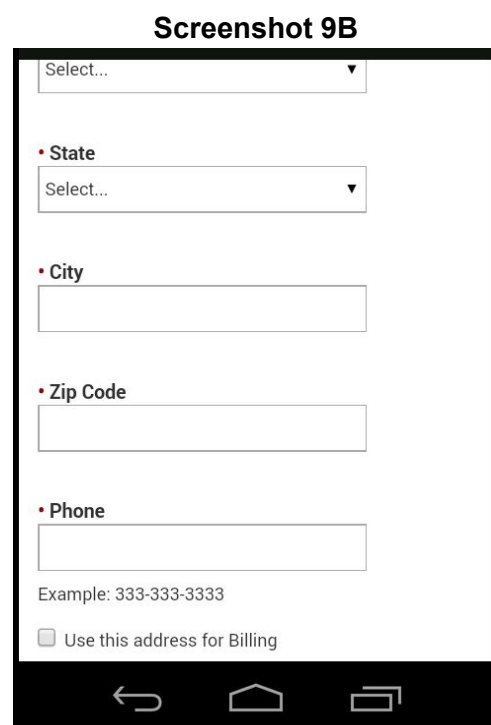
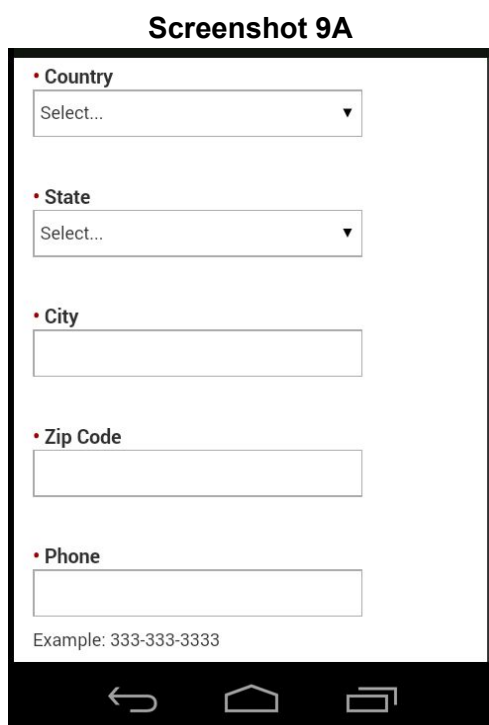
driver.get("https://www.browserstack.com/");
WebElement element = driver.findElement(By.id("pricing-upgrade-link"));
new Actions(driver).moveToElement(element).perform();

String context = driver.getContext();
driver.context("NATIVE_APP");
File scrFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
System.out.println("Screenshot saved as: " + scrFile);

driver.context(context);
driver.quit();
```

9.2 Clicking On Elements Does Not Work

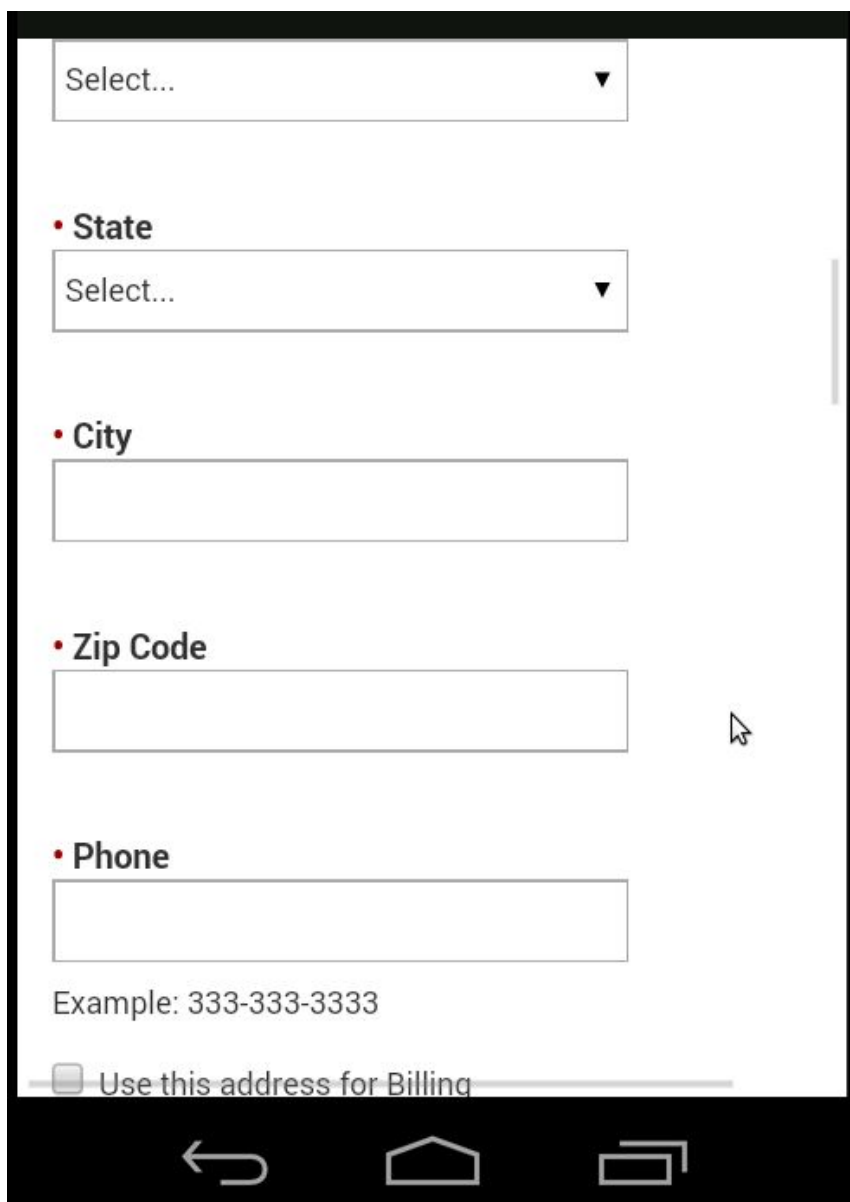
Sometimes clicking on elements does not seem to work. Of course there are several reasons why it may not work as expected, so you should start your investigation by debugging your code and take a look at the return values of some functions (is the element locator correct, was the element found ...). If everything is correct, there might be another, harder to track, error, that's causing this behavior: invisible scrollbars. Let's clarify what this error is by an example: imagine you want to check a checkbox on a website. That box is part of a form and is not displayed in the current viewport (the rectangle which covers the mobile display). Take a look at **screenshot 9A**.



Most webdriver implementations expect that elements to interact with, must lie within the current viewport. If an element lies not within the viewport, the webdriver tries to scroll to the element. In **screenshot 9B** the webdriver scrolls as far as needed to fully display the checkbox in the viewport. As soon as the checkbox is visible the click command is fired. But nothing happens? Now, how did that come?

Some web pages offers horizontal/vertical scrollbars for navigation. However, mobile devices are mostly configured to use as much space as possible to display information in the viewport. So unnecessary elements, like scrollbars, are faded out. They are only displayed when they are required: during scrolling. So what happened in this scenario is this: the webdriver starts scrolling to the checkbox. A scrollbar is displayed for a fraction of a second. The scrollbar is drawn over our checkbox and covers the element below which the webdriver does not recognize. Instead of checking the checkbox by a click, another UI element, the scrollbar, is simply clicked. **Screenshot 9C** demonstrates the explained situation. As workaround you could scroll to each element before you execute a click and wait until a certain threshold is reached (for example, scroll and wait 100 ms). It may not be the best solution

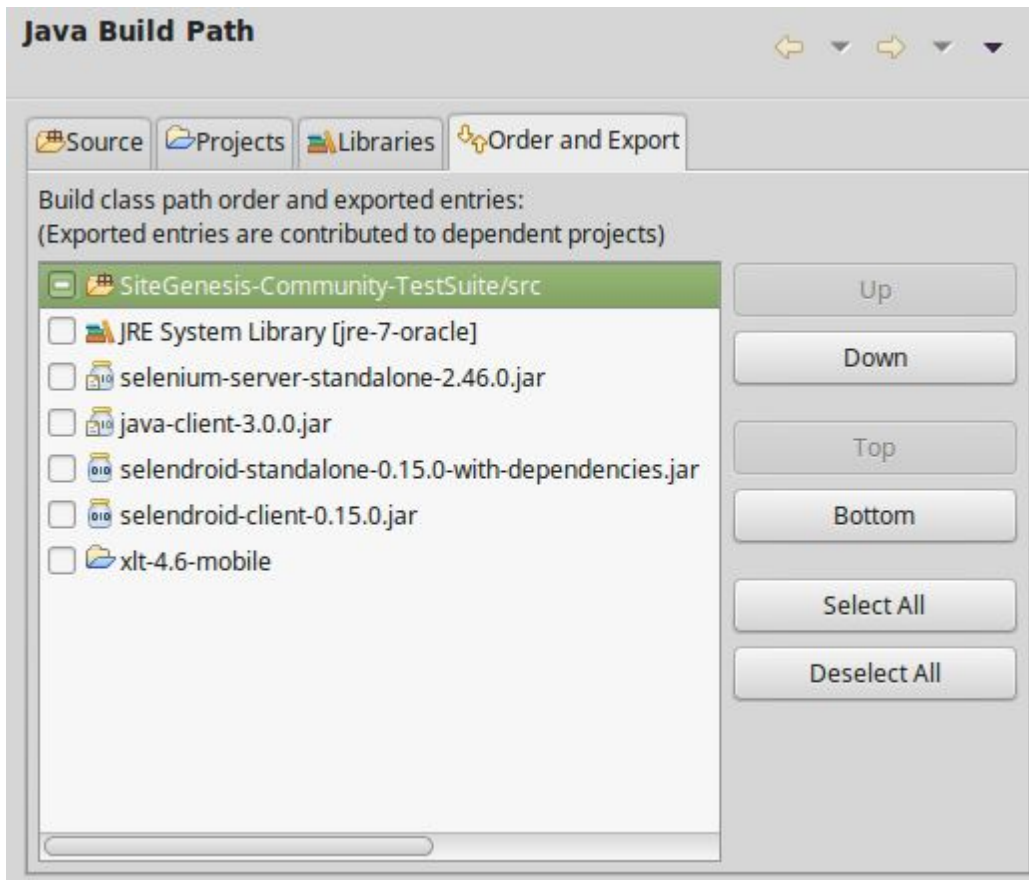
Screenshot 9C



The screenshot shows a mobile web interface with a form. At the top is a dropdown menu labeled "Select...". Below it is a section titled "• State" with another "Select..." dropdown. This is followed by a "• City" label and an empty text input field. Then, a "• Zip Code" label and an empty text input field. Below that is a "• Phone" label and an empty text input field. Under the phone field is the text "Example: 333-333-3333". At the bottom of the form is a checkbox labeled "Use this address for Billing". A vertical scrollbar is visible on the right side of the form, partially overlapping the "Use this address for Billing" checkbox. A mouse cursor is pointing at the scrollbar. The entire form is enclosed in a black border, and at the bottom is a black navigation bar with three white icons: a back arrow, a home icon, and a recent apps icon.

9.3 Appium Buildpath Order

Pay attention on organizing your imports when using SGCTS and Appium. It is recommended that your included libraries have a certain order, otherwise some strange runtime behavior may be detected. Unfortunately, errors related to the buildpath order are hard to track, because error messages that might appear often seems a little vague. In case you have trouble to run your Appium project for unknown reasons you might want to try to adapt the build path order. The screenshot below shows you a possible configuration for using Appium in conjunction with SGCTS.



9.4 Intel HAXM Driver For Windows

When using Microsoft Windows¹⁹ as operating system it may be possible that the performance of your local emulators is really choppy, if you don't have installed the latest Intel²⁰ Hardware Accelerated Execution Manager (HAXM). HAXM is a hypervisor which allows you to take advantage of Intel's Virtualization Technology (Intel VT) and this may give your local emulators a huge performance boost. That's the reason why the Android SDK Manager offers you the *Intel x86 Emulator Accelerator (HAXM installer)* package. So if you suffer under the low performance you may give that package a try.

¹⁹ <http://www.microsoft.com/en-us/windows>

²⁰ <http://www.intel.de/content/www/de/de/homepage.html>

10. Sources

<http://selendroid.io/faq.html>

<http://selendroid.io/architecture.html>

<http://spring.io/guides/gs/android/>

<http://www.guru99.com/introduction-to-selendroid.html>

<https://docs.saucelabs.com/tutorials/appium/>

<http://appium.io/>

<http://appium.io/slate/en/master/?java#appium>

<http://appium.io/introduction.html>

http://nishantverma.gitbooks.io/appium-for-android/content/appium/why_appium.html