Contents

Mobile WebDriver

On this course we provide an overview of 'mobile' automated execution with WebDriver.

Primarily because:

- our focus is the WebDriver API
- we are really looking at automating browsers rather than native
- 'all' you are really doing is creating a `RemoteWebDriver` connection to a server

History Mobile on WebDriver

Mobile support on WebDriver has changed a lot.

Initially there was `AndroidDriver` which was a small `.apk` which we installed to the Android device. We then ran this and used `AndroidDriver` to connect to the server as though it were a remote grid hub i.e `/wd/hub` in the url.

This is no longer supported by the WebDriver project.

Google themselves created a WebDriver which was part of the sdk - that is no longer supported.

Selendroid is still supported but seems to be mainly used for native applications.

- selendroid.io

The viable option for the future appears to be the 'Appium' project.

Since Appium is a separate project it is really out of scope for this course, but since everyone wants to experiment with mobile. I will cover it in basic outline form.

About Appium

The Appium home page is appium.io

It runs as a local grid server. You connect to it as a `RemoteWebDriver` with capabilities defining the device and browser you want to use.

Appium then connects to the device, uploads an `.apk` to the device and can route your normal WebDriver commands through to the device.

There is an Appium client driver which you can use if you want to perform native app testing. But for 'web' testing you can use the normal RemoteWebDriver connection.

Getting Installed

The install process is a little convoluted.

I've had it work easy on some machines and fail completely on others.

So I break the install into small chunks.

Install the Android SDK

Appium uses the Android SDK, therefore we need to install that first, and it is important to test that is working before we go any further forward.

Download the android sdk from Google:

- developer.android.com/studio/index.html#downloads

If you want the 'super easy version' then download the full Android Studio:

- https://developer.android.com/studio/install.html

I downloaded the SDK tools package installer - I did not download the full android studio.

I installed to `d:\android` to avoid permission problems on windows.

During the install the 'android sdk manager' will run automatically.

At this point - accept all the recommended updates and installs.

The important ones are the updated SDK and the USB driver.

If you are following the Appium install instructions you will immediately head off and install Appium but I'm going to ask you to stop here and make sure that the SDK is working properly otherwise you will find it hard to debug any problems.

SO PAUSE HERE

Make sure that you setup the paths and environment variables for Android.

If the SDK installer hasn't added the environment variables then:

- Add an environment variable `ANDROID_HOME` which points to the location of your sdk, in my case this was `D:\Android\android-sdk`
- Add the tools to your path
  - `%ANDROID_HOME%\tools`
  - `%ANDROID_HOME%\platform-tools`

Check that your version of `adb` is accessible via the command line.

- Open a command line and type `adb version`

On windows I also check `where adb` becuase I have had instances where there were very old versions of the `adb` executable installed in strange and early versions of the path, which caused me issues using Appium and other tools.

```
C:\Users\Alan>adb version
Android Debug Bridge version 1.0.36
Revision fd9e4d07b0f5-android
```

`adb` is the Android Debug Bridge and is the key tool that Appium is going to use:

- developer.android.com/studio/command-line/adb.html

You can find other tools that will be installed on the command line tools page of Google Android SDK:

- developer.android.com/studio/command-line/index.html

Make sure your phone is setup for USB Debugging

The easiest way to do this, with the clearest up to date instructions is to try and use the Chrome Remote Debugger.

- developers.google.com/web/tools/chrome-devtools/debug/remote-debugging/remote-debugging
- The old instructions are also useful:
  - developer.chrome.com/devtools/docs/remote-debugging
- make sure you have developer options enabled on your Android device
  - developer.android.com/studio/run/device.html#device-developer-options
- connect your Android device to your computer
- you should see a prompt to "Allow USB debugging?" on your device

If all went well, then in Chrome go to "chrome://inspect" and hopefully you will see your device there and you'll be able to control it from your computer.

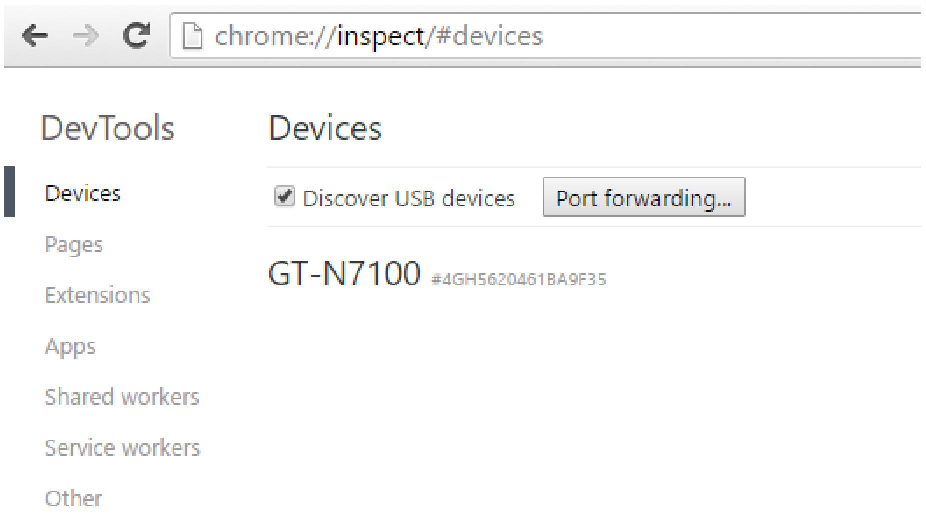


Figure 1: Chrome Inspect

If not then follow the troubleshooting tips from Google:

- developer.chrome.com/devtools/docs/remote-debugging#troubleshooting

I have had issues where:

- the cable was bad and kept dropping the connection
- I had to use different usb ports
- The Android SDK Google USB driver was not installed - use `android sdk` to install it
- I had to 'Revoke USB debugging authorization', then plugin the cable a few times for the dialog to appear

In essence - if you can't get Chrome remote debugging working then don't go any further. This is where you will get the best troubleshooting instructions on line to fix the problem.

In the past I have had to use `adb` directly:

- `adb kill-server`
- `adb start-server`

Also you can use:

- `adb devices`

```
C:\Users\Alan>adb devices
List of devices attached
4gh5620461ba9f35        device
```

If it says 'device' you should be good to go, if it says 'offline' you have a problem and Chrome Remote Debugging won't work either.

The number shown when you run the command `adb devices` is important, you'll need that to connect from WebDriver. You can also see that number listed in the chrome tab as well.

That is the `deviceName` capability that you'll use to connect.

Install Appium

This should be easy, but again I've found I have to skip the official route.

If you visit the appium.io home page you'll see a big blue 'Download Appium' button. Which I have found (more often than not) downloads an old version of appium.

So visit the appium.io/downloads.html instead. Then download the "Appium Desktop App" suitable for your operating system.

I again choose the custom install and install it to `d:\appium` to avoid any permission problems on windows.

When that installs you should be able to run Appium.

Connecting to Appium from WebDriver

On the Appium GUI click the `[play]` button.

This will start Appium local grid hub.

The diagnostic information it shows is important:

```
> Launching Appium server with command: D:\Appium\node.exe lib\server\
main.js --address 127.0.0.1 --port 4723 --platform-name Android
 --platform-version 23 --automation-name Appium --log-no-color
> info: Welcome to Appium v1.4.16
(REV ae6877eff263066b26328d457bd285c0cc62430d)
> info: Appium REST http interface listener started on 127.0.0.1:4723
> info: [debug] Non-default server args: {"address":"127.0.0.1","logNoColors":true,"platforr
> info: Console LogLevel: debug
```

If there are any errors then deal with them.

This tells you that you need to use "127.0.0.1" as the url for the hub

- `"http://127.0.0.1:4723/wd/hub"`

I would connect to it from WebDriver by using a `RemoteWebDriver`

```
        DesiredCapabilities capabilities = new DesiredCapabilities();

        // the device name can be seen when you do "adb devices"
        capabilities.setCapability("deviceName", "4gh5620461ba9f35");
        capabilities.setCapability("platformName", Platform.ANDROID);
        capabilities.setCapability("app", "Chrome");


        try {
            // add url to environment variables to avoid releasing with source
            String appiumURL = "http://127.0.0.1:4723/wd/hub";
            driver = new RemoteWebDriver(
                    new URL(appiumURL),
                    capabilities);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
```

In the above code I used the value shown by `adb devices` as the `deviceName` capability.

I use `Chrome` as the browser as I have had much less problems configuring it on android that the default browser.

After this, we can use any WebDriver test that we want to run with the `driver`.

Bear in mind that you may well encounter limitations with the Appium driver. Certainly we have encountered issues with frames and JavaScript execution in the past.

If you encounter problems then:

- close/open the Appium GUI
- stop/start the Appium server
- Pay attention to the errors that Appium throws in the GUI and Google them directly.

The GUI shows the full Appium log of what is going on so it is worth reading because it will give you some insight into what Appium is doing and may highlight any issues.

Mac

I've had far fewer issues installing on Mac and generally just follow the instructions on the Android SDK site and the Appium site directly.

Running the Tests

I initally ran the @Test through Chrome on a physical device because Chrome usually provides better compatibility.

I use Chrome on physical devices because at the current time of writing, Appium is not compatible with the Android N images in the SDK which provide Chrome.

Results of running the tests

After running the tests on Chrome the following types of tests failed:

- WindowManageExerciseTest
- WindowsManageExampleTest

The windows management tests fail because Appium does not allow resizing and moving of windows - on mobile, that concept does not exist so the tests fail. Most of the Windows management commands will fail on remote driver execution.

- CookiesExercisesTestWorkWithExtraSync

I did not expect these to fail given that other cookie tests passed, so we need to investigate these to see why they did not work. As you'll see in the investigation, this was a driver issue, but my tests were brittle in checking a condition so I made an amendment in the test code to make them check number of cookies in terms of a 'domain' SeleniumSimplified Search Engine Cookies', rather than in terms of the physical browser 'All Cookies'. I explain this in the follow on video showing this issue and fix.

- ManipulateExerciseSubmitFileTest

I do not expect the file upload tests to work on a Remote WebDriver since we have coded it for local demonstration.

- BasicTestsRefactored

I suspect some tests failed because of timing issues. The tests currently use a lot of external sites, and I should probably change them so that they don't use compendiumdev.co.uk and seleniumsimplified.com as these can sometimes laod slowly or have items on the page that don't render effectively on mobile. This is not a 'test' or mobile issue, it just means I have to use a better system under test.

Re-running these tests showed that they passed. So this was an environmental issue, not a test or mobile issue.

- UserInteractionsExercisesTest

My initial thought was that the user interactions failed because sometimes user interactions can be very device specific and perhaps they didn't work effectively on Chrome Mobile. But investigation (and you can see this in the later video) showed that this was not the case and the @Test failure was a result of earlier browser workaround code.

- BasicDataDrivenTest

- CsvDrivenTest

I did not expect these test to fail, so had to investigate why. The reason was 'mobile' specific, but the workaround was pretty simple and you'll see the investigation approach I used.

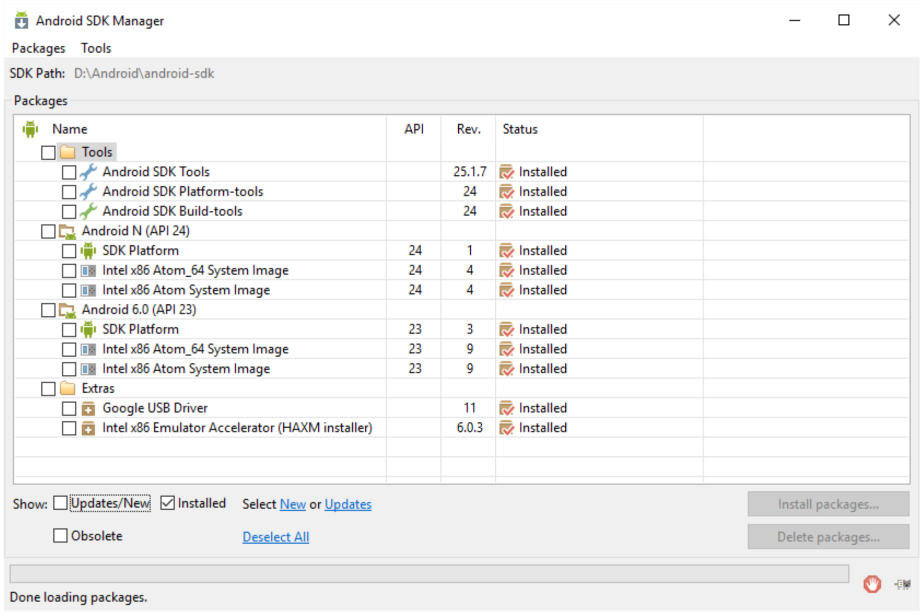
The basic summary was that most of the tests passed on Chrome. Which was a pretty good result and mobile interaction through Appium and WebDriver has advanced a lot.

We get slightly different results through the 'Browser' as you'll see in the Android Virtual Device @Test execution later in the section.

Create an AVD

To create an AVD, use `android sdk` and make sure you have installed:

- API you need e.g. v24 or v23, with the SDK Platform for that API and at least one System Image for that version
- If you encounter any HAX errors when starting an AVD then you should also install the "Intel x86 Emulator Accelerator"



SDK Path: D:\Android\android-sdk

Packages

Name	API	Rev.	Status
☐ Tools			
☐ Android SDK Tools		25.1.7	Installed
☐ Android SDK Platform-tools		24	Installed
☐ Android SDK Build-tools		24	Installed
☐ Android N (API 24)			
☐ SDK Platform	24	1	Installed
☐ Intel x86 Atom_64 System Image	24	4	Installed
☐ Intel x86 Atom System Image	24	4	Installed
☐ Android 6.0 (API 23)			
☐ SDK Platform	23	3	Installed
☐ Intel x86 Atom_64 System Image	23	9	Installed
☐ Intel x86 Atom System Image	23	9	Installed
☐ Extras			
☐ Google USB Driver		11	Installed
☐ Intel x86 Emulator Accelerator (HAXM installer)		6.0.3	Installed

Show: ☐ Updates/New ☑ Installed Select New or Updates

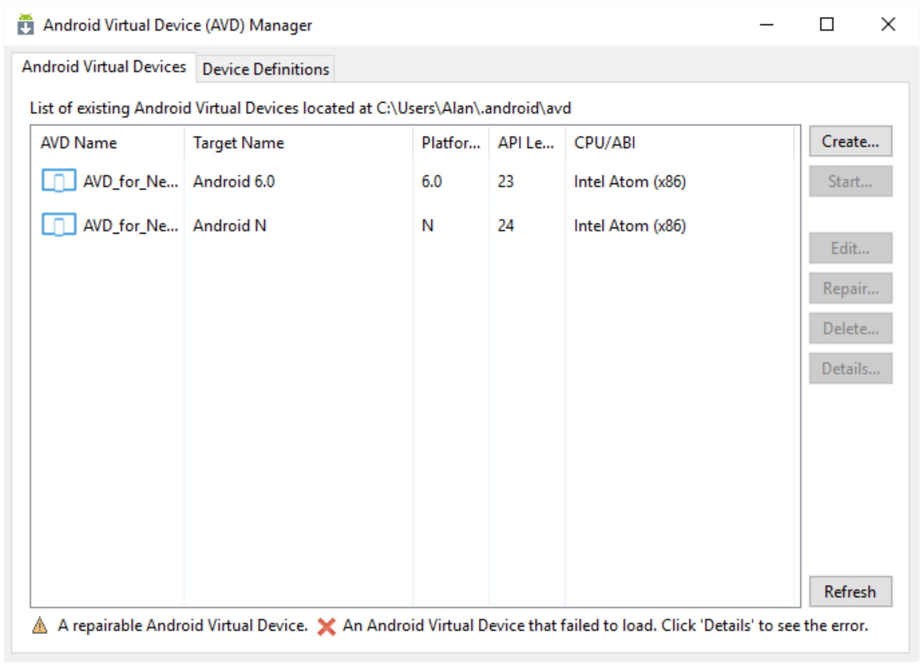
☐ Obsolete Deselect All

Done loading packages.

Figure 2: SDK Installs

Use `Tools \ Manage AVDs` to create an AVD and chose the 'Device Definitions' to make it easy to configure one.

Select a 'Device' from the list and `Create AVD`



Android Virtual Device (AVD) Manager

List of existing Android Virtual Devices located at C:\Users\Alan\.android\avd

AVD Name	Target Name	Platfor...	API Le...	CPU/ABI
AVD_for_Ne...	Android 6.0	6.0	23	Intel Atom (x86)
AVD_for_Ne...	Android N	N	24	Intel Atom (x86)

⚠ A repairable Android Virtual Device. ✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

Figure 3: SDK Installs

- Give config a unique name
- Select the API Level that will work with your version of Appium
- Make sure you have a system image downloaded for that version so you can choose it as the `CPU/ABI`
- I use the `Skin with dynamic hardware controls`
- Set enough RAM - you might have to adjust this
- I choose to `Use Host GPU`

Use an AVD

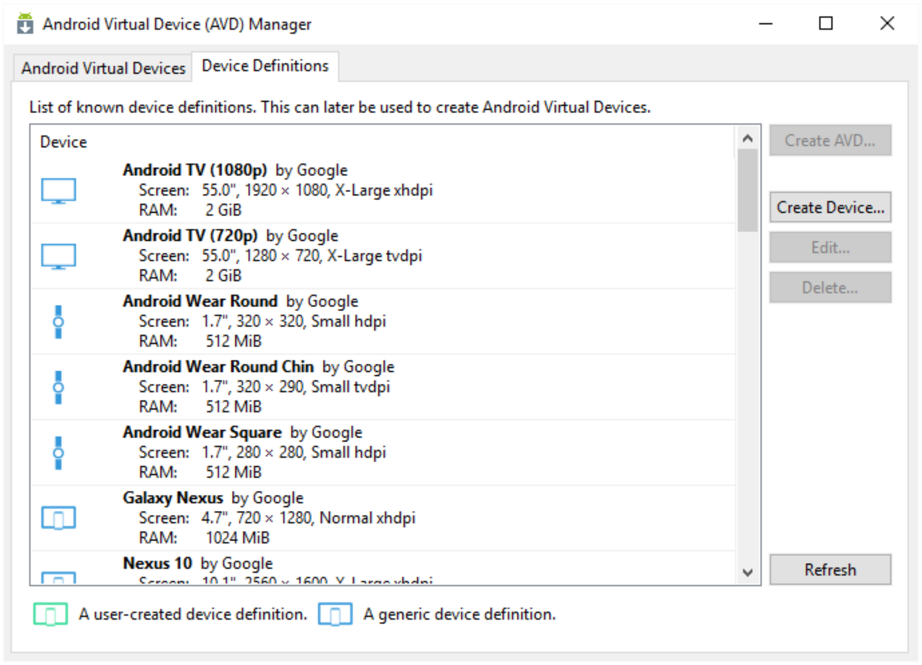
You can start and AVD from the `Manage AVDs` list.

You can see the device name in `adb devices`

If the device is running Chrome then you can start a remote debugging session:

- `chrome://inspect`

Connect to the running AVD using a RemoteWebDriver as shown in the previous code.



List of known device definitions. This can later be used to create Android Virtual Devices.

Device

Android TV (1080p) by Google
Screen: 55.0", 1920 × 1080, X-Large xhdpi
RAM: 2 GiB

Android TV (720p) by Google
Screen: 55.0", 1280 × 720, X-Large tvdpi
RAM: 2 GiB

Android Wear Round by Google
Screen: 1.7", 320 × 320, Small hdpi
RAM: 512 MiB

Android Wear Round Chin by Google
Screen: 1.7", 320 × 290, Small tvdpi
RAM: 512 MiB

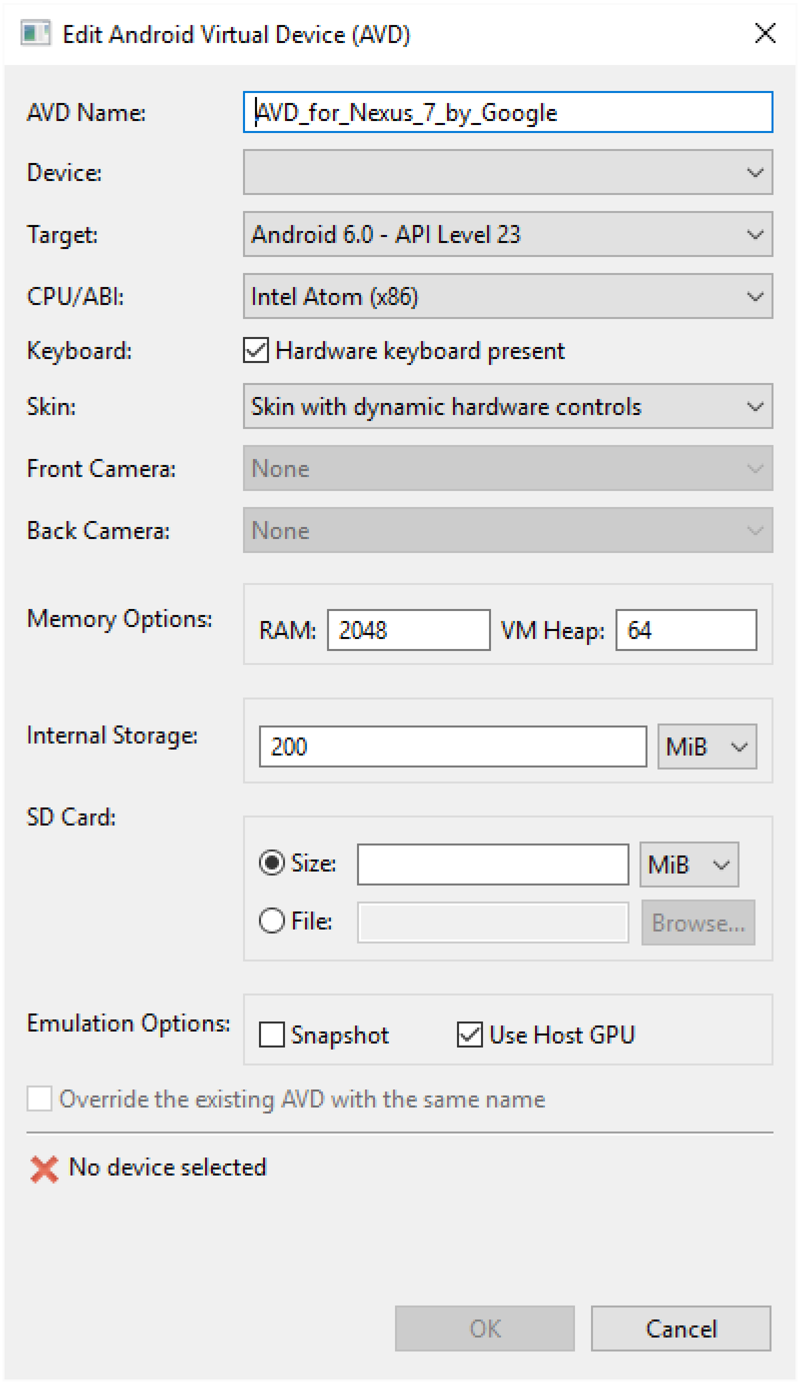
Android Wear Square by Google
Screen: 1.7", 280 × 280, Small hdpi
RAM: 512 MiB

Galaxy Nexus by Google
Screen: 4.7", 720 × 1280, Normal xhdpi
RAM: 1024 MiB

Nexus 10 by Google
Screen: 10.1", 2560 × 1600, X-Large xhdpi

A user-created device definition. A generic device definition.

Figure 4: Device Definitions


Edit Android Virtual Device (AVD)

AVD Name:	AVD_for_Nexus_7_by_Google
Device:	
Target:	Android 6.0 - API Level 23
CPU/ABI:	Intel Atom (x86)
Keyboard:	☑ Hardware keyboard present
Skin:	Skin with dynamic hardware controls
Front Camera:	None
Back Camera:	None
Memory Options:	RAM: 2048 VM Heap: 64
Internal Storage:	200 MiB
SD Card:	[✓ Size:] MiB; File:
Emulation Options:	☐ Snapshot ☑ Use Host GPU

☐ Override the existing AVD with the same name

No device selected

Figure 5: Device Config