Appium Bootcamp 2: Configuring Appium

Appium Bootcamp is a series of articles prepared by Selenium guru Dave Haeffner, and leading Appium contributor Matthew Edwards, for Sauce Labs. Dave also authors the Elemental Selenium website, which includes tips for using Selenium, and where you can sign up for his weekly email on the topic of Selenium testing. He is also the author of the Selenium Guidebook.

In order to get Appium up and running there are a few additional things we'll need to take care of.

If you haven't already done so, install Ruby and setup the necessary Appium client libraries (a.k.a. "gems"). You can read a write-up on how to do that here.

Installing Necessary Libraries

Assuming you've already installed Ruby and need some extra help installing the gems, here's what you to do.

1. Install the gems from the command-line with `gem install appium_console`
2. Once it completes, run `gem list | grep appium`

You should see the following listed (your version numbers may vary):

```
sh appium_console (1.0.1) appium_lib (4.0.0)
```

Now you have all of the necessary gems installed on your system to follow along.

An Appium Gems Primer

`appium_lib` is the gem for the Appium Ruby client bindings. It is what we'll use to write and run our tests against Appium. It was installed as a dependency to `appium_console`.

`appium_console` is where we'll focus most of our attention in the remainder of this and the next post. It is an interactive prompt that enables us to send commands to Appium in real-time and receive a response. This is also known as a record-eval-print loop (REPL).

Now that we have our libraries setup, we'll want to grab a copy of our app to test against.

Sample Apps

Don't have a test app? Don't sweat it. There are pre-compiled test apps available to kick the tires with. You can grab the iOS app here and the Android app here. If you're using the iOS app, you'll want to make sure to unzip the file before using it with Appium.

If you want the latest and greatest version of the app, you can compile it from source. You can find instructions on how to do that for iOS here and Android here.

Just make sure to put your test app in a known location, because you'll need to reference the path to it next.

App Configuration

When it comes to configuring your app to run on Appium there are a lot of similarities to Selenium -- namely the use of Capabilities (e.g., "caps" for short).

You can specify the necessary configurations of your app through caps by storing them in a file called `appium.txt`.

Here's what `appium.txt` looks like for the iOS test app to run in an iPhone simulator:

```
[caps]
platformName = "ios"
app = "/path/to/UICatalog.app.zip"
deviceName = "iPhone Simulator"
```

And here's what `appium.txt` looks like for Android:

```
[caps]
platformName = "android"
app = "/path/to/api.apk"
deviceName = "Android"
avd = "training"
```

For Android, note the use of both `avd`. The `"training"` value is for the Android Virtual Device that we configured in the previous post. This is necessary for Appium to auto-launch the emulator and connect to it. This type of configuration is not necessary for iOS.

For a full list of available caps, read this.

Go ahead and create an appium.txt with the caps for your app.

Launching The Console

Now that we have a test app on our system and configured it to run in Appium, let's fire up the Appium Console.

First we'll need to start the Appium server. So let's head over to the Appium GUI and launch it. It doesn't matter which radio button is selected (e.g., Android or Apple). Just click the `Launch` button in the top right-hand corner of the window. After clicking it, you should see some debug information in the center console. Assuming there are no errors or exceptions, it should be up ready to receive a session.

After that, go back to your terminal window and run `arc` (from the same directory as `appium.txt`). This is the execution command for the Appium Ruby Console. It will take the caps from `appium.txt`and launch the app by connecting it to the Appium server. When it's done you will have an emulator window of your app that you can interact with as well as an interactive command-prompt for Appium.

Outro

Now that we have our test app up and running, it's time to interrogate our app and learn how to interact with it.