**APPIUM: IOS TESTING WITH NODE.JS STEP-BY-STEP GUIDE**

Good day to everyone, in this tutorial, we will go through the process of writing node.js tests for iOS, from software installation to running actual tests on simulated devices with use of Appium.

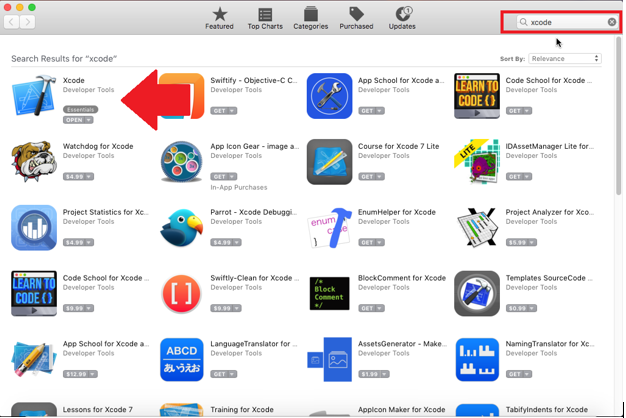
Appium is a freely distributed open source framework for mobile application UI testing. Appium supports all languages that have Selenium client libraries like Java, Objective-C, JavaScript with node.js, PHP, Ruby, Python, C# etc.

**Prerequisites to use Appium for iOS testing:**

1. **OS X**. I recommend using OS X Sierra (10.12 or higher). This is an essential requirement, since Appium cannot start sessions for iOS without Xcode, which is available only for Mac.
2. **JDK** ([Java Development Kit](http://www.oracle.com/technetwork/java/javase/downloads/index.html)).
3. **Xcode**. Get one either from AppStore or [Apple Developer](https://developer.apple.com/xcode/). By default, comes with iOS SDK, which we will need for running simulators.
4. **Homebrew** – package manager for Mac. Will make our life easier in the long run.
5. **Appium Client** **for Mac** ([client](http://appium.io/)).
6. **IDE and testing framework of choice**. In our case, we will use **js**and**mocha** testing frameworks, so we need to install node.js at first. We will do so in the “Environment Setup” section below. NOTE: even though Appium client supplied with “Node.exe” and NPM (node package manager), we will install those manually to have the latest version.
7. **carthage** – dependency for running XCUItests for iOS apps.

**Environment Setup**

**Step 1.**Download and install Xcode. Go to AppStore and type *“xcode”* in a search field. Grab the first one, provide your Apple ID if needed. After download is completed, go to the Applications folder and run Xcode to finish installation.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p1.png)

**Step 2.** Install Homebrew.

**2.1.**Open your terminal and run this command “*/usr/bin/ruby –e “$(curl -fsSL*[*https://raw.githubusercontent.com/Homebrew/install/master/install)*](https://raw.githubusercontent.com/Homebrew/install/master/install))*“”*(without outer quotes). Homebrew will start installation and ask for root password if needed. Just follow installation guidelines and you’ll be fine. For more details visit [Homebrew site](https://brew.sh/index.html).

**2.2.**After installation run *“brew doctor”* (without quotes) to verify, that everything is okay.

**2.3.** Run a command *“export PATH=”/usr/local/bin:$PATH””* (without outer quotes).

**Step 3**. Install node.js.

**3**.**1.**We will use Homebrew for this, so in terminal run a command *“brew install node”*. This will setup everything you need.

**3**.**2.** After installation, run commands “node –v” and “npm –v” to see installed version of node.js and npm respectively.

**Step 4**. Download and install JDK. After download is complete, navigate to your Downloads folder, double-click the downloaded .dmg file and follow installation instructions.

**Step 5**. Download and install Appium Client. Installation process doesn’t differ from the one described in **Step 4.**

**Step 6.**Install carthage. Simply run *“brew install carthage”* in the terminal.

Now, all of the required software is installed, let’s proceed with further setup.

**Simulator setup.**

In order to run our tests, we will need a device to run them on. For sake of this tutorial we will use iOS simulator, which is bundled with Xcode.

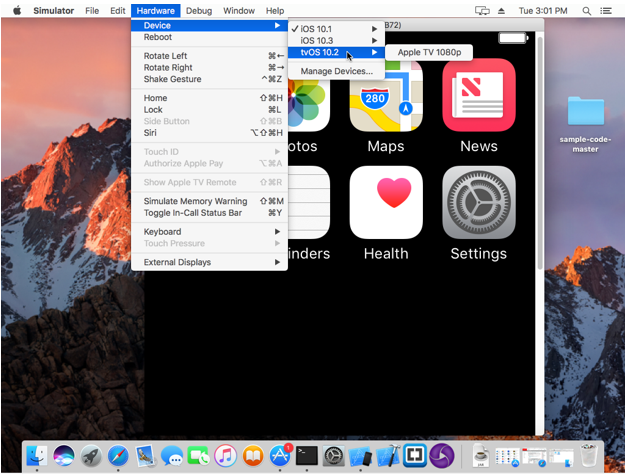
**Step 1.**Press *Command + Space* to launch Spotlight.

**Step 2.** Type *Simulator* and hit Enter.

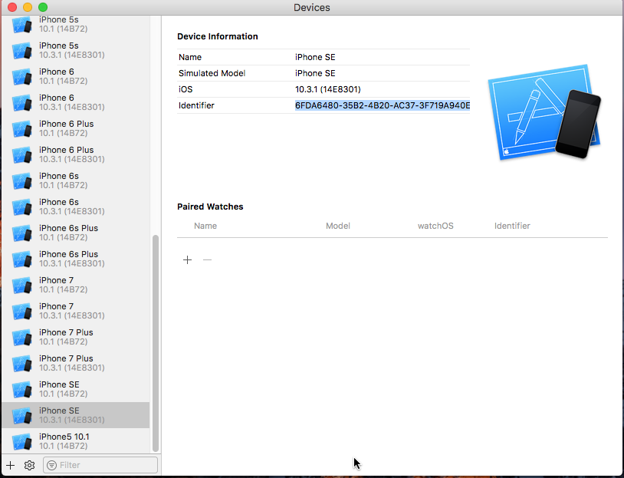
**Step 3.**Launch a Simulator from search results

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p2.png)

**Step 4.** When in Simulator, click the *“Hardware”* menu, then the *“Device”*→ *“Manage Devices…”*.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p3.png)

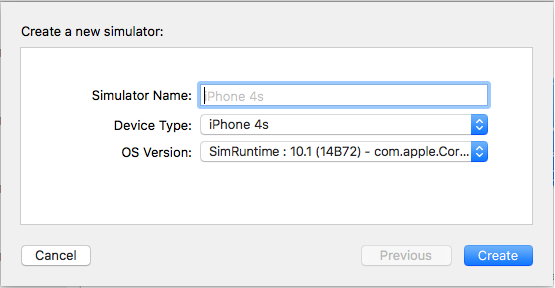
**Step 5.**In a Devices window you can view all available devices to you. Click the one you want to use for tests, let’s say *“iPhone SE”* with iOS version of 10.3.1.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p4.png)

**Step 6.** We will need info about the **name, iOS,**and**Identifier**to start both Appium Inspector session and our test execution. NOTE: if you need to run tests on device with iOS version not presented in list, you can create it by yourself.

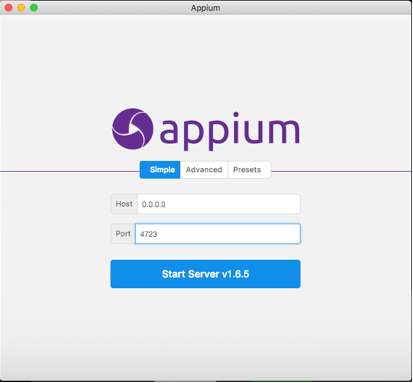
**6.1.** Click the “+” button at bottom left corner of the window and choose the “Add Simulator”.

**6.2.** Set device name (it is mandatory to set unique device name), choose device type and iOS version. Download one you need if it is not presented in list.

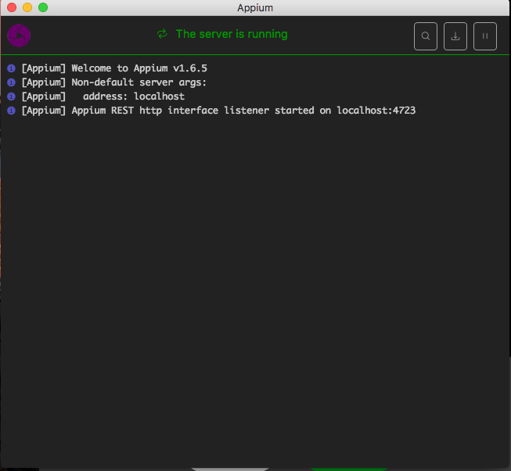
[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p5.png)

**6.3.**Now you should see the newly created simulator in a list of simulators. Do not close this window yet as we will need it.

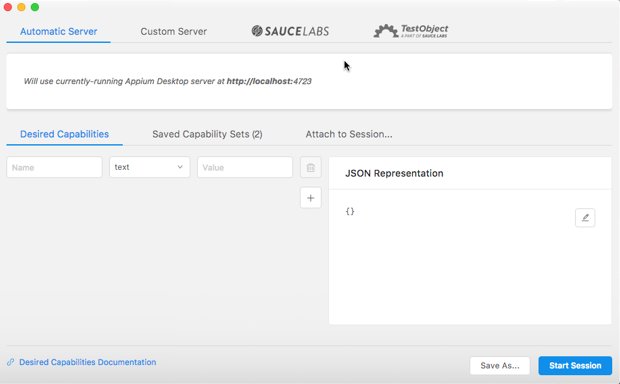
**Step 7.**Launch the Appium Client from your Applications folder.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p6.png)

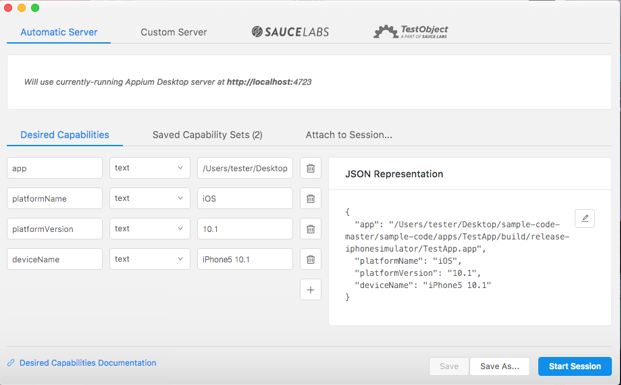
**Step 8.** Type *“localhost”* (without quotes) in a *Host* field and click the *“Start Server v.1.6.5”*button. Now we have Appium Server up and running.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p7.png)

**Step 9.** Click the *“Start Inspector Session”* (button with magnifier on it). You should see this:

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p8.png)

**Step 10.** Now, let’s specify details of simulator we want to use and app we want to install. Create the following set of Desired Capabilities:

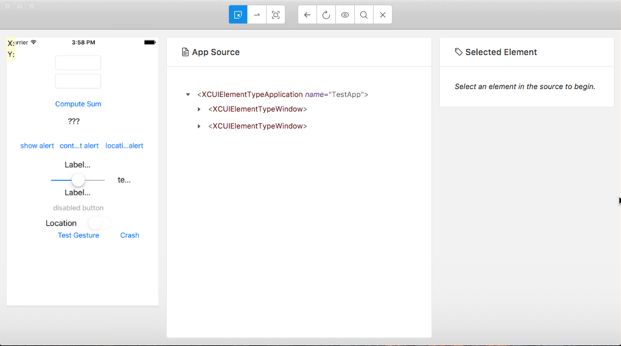
[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p9.png)

Points to remember here:

* **app** – full path to your built app on Mac. I’m using TestApp provided by Appium Team and available [here](https://github.com/appium/sample-code). **In case you want launch tests for Safari**, omit this option and use *“browserName”: “safari”* NOTE: your app must be built specifically for simulator and testing purposes.
* **platformName** – name of our platform. Since we want to run iOS tests, type iOS.
* **platformVersion**– version of iOS on tested device. Look at *“iOS”* parameter of your simulator of choice on the *“Devices”* window and set it here.
* **deviceName**– name of device you want to use. Grab the *“Identifier”* parameter of desired simulator on the *“Devices”* window and set it here.

**Step 11.** After that, switch back to your Simulator, go *“Hardware”*→ *“Device”* and choose a device type you want if your simulator isn’t already of that type. Wait for Simulator to restart and keep it running – this will decrease Appium Inspector startup time drastically.

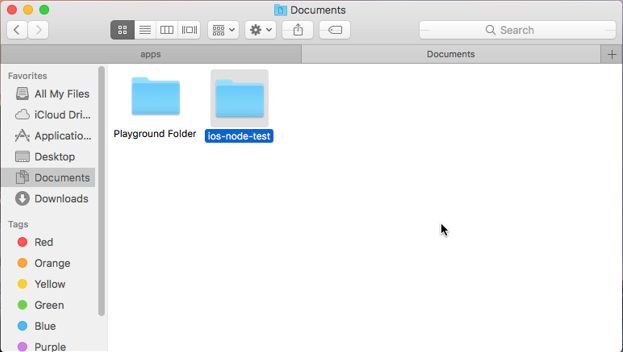
**Step 12.**In Appium, click the *“Start Session”* button. You should see app being installed on your simulator of choice as good as Appium Inspector loading. NOTE: if you’ve encountered Appium error stating *“could not get xcode version”* you may want to set path to Xcode manually. In terminal, run commands “***sudo xcode-select –reset***” and *“****sudo xcode-select –switch /Applications/Xcode.app****”* (both without quotes). This should fix the issue. You should specify path to your Xcode.app according to your installation folder in case it differs from Applications. You should see something like this:

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p10.png)

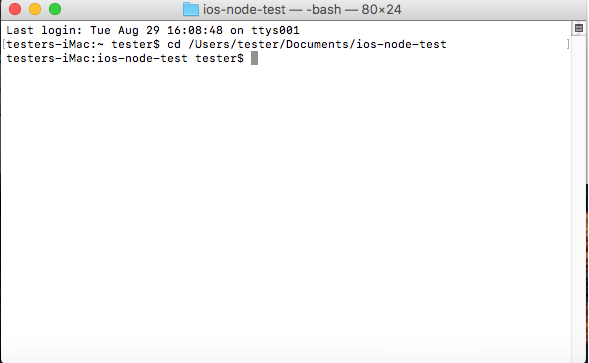
**Writing tests**

Now, we have everything set up. At this point, we can proceed with writing tests for our app using node.js.

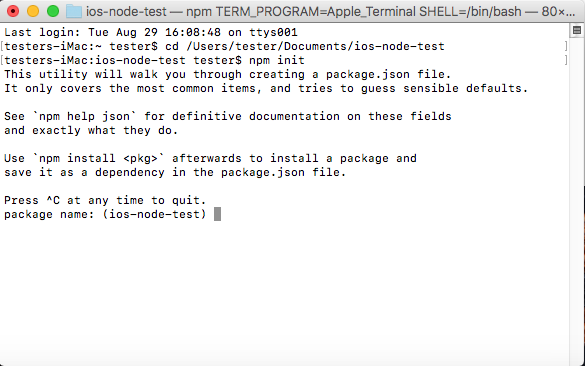
**Step 1.** In Documents, create a new folder and name it *“ios-node-test”*. This will be our project folder.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p11.png)

**Step 2.** Open Terminal and navigate to your project folder using command *“cd /full/path/to/your/folder”* or start it in project folder directly.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p12.png)

**Step 3.** Execute the “npm init” command in the folder. Follow the setup process in a command line to configure package.json file of your project.

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p13.png)

**Step 4.** After the basic setup, we need to add a bunch of dependencies in order to make our tests work. We will use **mocha** as our testing framework with **chai**for assertions. Also, we will require **webdriverio** testing utility, which improves interaction with mobile web, native and hybrid apps. So, run the following commands:

* npm install mocha
* npm install chai
* npm install chai-as-promised
* npm install wd
* npm install wdio-mocha-framework –D (–D parameter here stands for installation this dependency only for developers)
* npm install webdriverio@4.7.1 –D

NOTE: at the time of writing this guide the latest version of webdriverio is 4.8.0. However, we are installing v.4.7.1 due to *setValue()* method working incorrectly when using webdriverio v.4.8.0 in conjunction with Appium v1.6.5. Probably, fix for this issue will be included in future releases either from Appium or Webdriverio team.

**Step 5.** Create a subfolder and call it *“helpers”*. In *“helpers”* create file with .js extension and name in *“desiredCapabilities”*. In this file we will specify all the necessary details for our script, such as application we want to run and device we want to use for testing. Here is the completed content of the file, which follows our previous steps:

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p14.png)

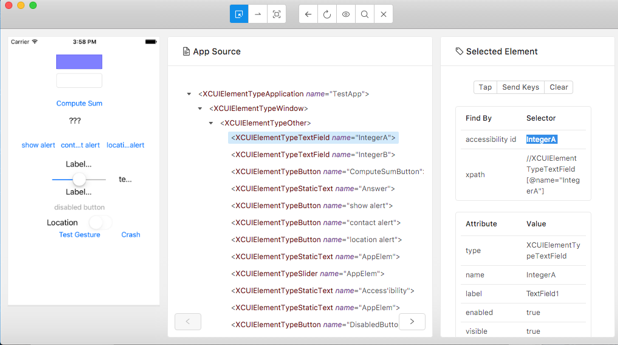
As you can see, in *desiredCapabilities* we haven’t change anything compare to Appium Inspector, although you may consider setting apps *bundle id* if you have one.

**Step 6.**Return to project folder and create a file *“native.js”*. This will be our script to run tests for native app. In order to locate elements, we will use *accessibility id* strategy, provided by **webdriverio**, for example:

*.click(‘~IntegerA’)*

This resource is specific to TestApp application, provided by Appium team as code sample. To get accessibility id of the element or any other identifier available we will use Appium Inspector. Just click the element while in the “Select” mod to properties panel to appear and grab the most suitable attribute or identifier. If by any means you cannot retrieve any selector except *xpath* you can use it as well, for example:

*.click(//XCUIElement TypeButton[@name=”ComputeSumButton”])*

**[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p15.png)**

**Step 7.** With that in place, we are ready to start scripting. As I said earlier, we will use [webdriverio](http://webdriver.io/) for writing our test, so consider looking at their docs for more info.

**7.1.**Let’s import all required components and create **client** – main object for interactions and actions:

var webdriverio = require(‘webdriverio’);

var expect = require(‘chai’).expect;

var config = require(‘./helpers/desiredCapabilities’).options;

var client = webdriverio.remote(config);

**7.2.**Now, create the main function by using **describe** keyword and pass “Simple cases” as the first parameter. This function will wrap our tests and declare **before**hook. This will be executed before our test-suite.

describe(‘Simple cases’, function () {

before(function () {

this.timeout(50000);

return client.init();

});

**7.3.** Create a sub-function by using **describe**keyword and call it “test-app-addition”. Set timeout for executing tests of this suite and write the tests with use of **it** keyword.

describe(“test-app-addition”, function(){

it(“add-3-3”, function(){

return client.element(‘~IntegerA’)

.setValue(“3”)

.element(‘~IntegerB’)

.setValue(“3”)

.click(‘~ComputeSumButton’)

.element(‘~Answer’)

.getText()

.then(function (text) {

expect(text).to.equal(‘6’);

});

});

});

**7.4.** In scope of wrapper function, add **after**hook, which will close the connection as well as the AUT.

after(function() {

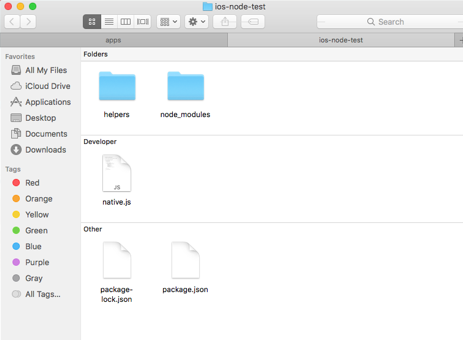
return client.end();

});

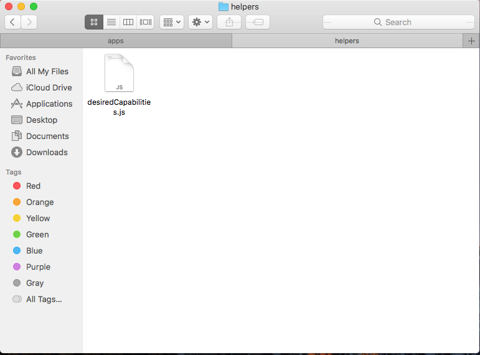
});

**Step 8.** Open “package.json” and under scripts section add exactly the same line: “native”: “mocha native.js”. This will allow us to run our test by “npm run native”.

**Step 9.** Finally, your “ios-node-test” folder should be like this:

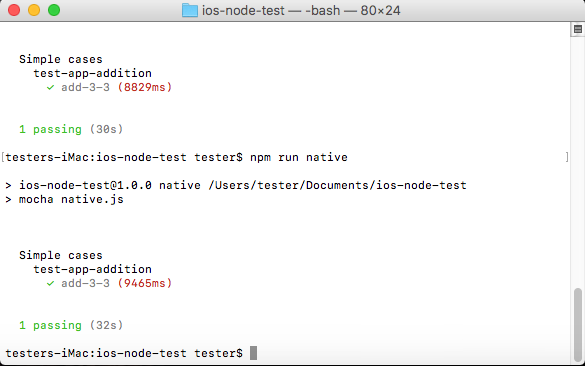
[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p16.png)

And “helpers” folder should be like this:

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p17.png)

**Step 10.** Close your Appium inspector by hitting the “Quit Session & Close Inspector” button at the top-center panel.

**Step 11.**Run a script by executing “npm run native” in your project folder. You should see execution process on your device as good as mocha log in the console:

[](http://latestsoftwaretestingnews.com/mobiletesting/wp-content/uploads/2017/09/p18.png)

That’s it! Now you know how to install, setup and configure all required software to run your tests for iOS UI automated tests, as well as prepare environment for your tests. And, of course, you have a basic knowledge about writing tests themselves using node.js with webdriverio and mocha.

[TestMatick](http://testmatick.com/) company offers automation testing services for web and mobile applications. With TestMatick you can also utilize all the advantages of continuous integration. It includes setup of the environment which automatically integrates all parts of a project and executes tests every-time when someone makes a commit to the version control.

We provide QA automation services using the following automation tools:

– Selenium WebDriver  
– Appium  
– Automated QA TestComplete  
– HP / Mercury QTP  
– Segue SilkTest  
– Ranorex  
– Robotium

Source http://latestsoftwaretestingnews.com/mobiletesting/appium-ios-testing-with-node-js-step-by-step-guide/