The options of inspecting iOS 10 app with Appium 1.6

Background

Appium is an open source test automation framework for mobile apps. It has the similar concept as Selenium WebDriver. For iOS, underlayer, it uses Xcode's instrument tool UIAutomation to drive the UI operation on the app.

Apple introduced XCUITest in **Xcode 7** back in 2015. And in **Xcode 8**, Apple took one step further to remove the previous UI test tool UIAutomation. This leads Appium to overhaul its underlayer mechanism to use Facebook's WebDriverAgentwhich implements most of the WebDriver protocol using XCUITest. This has been released as Appium 1.6.0 recently.

In a word, to test an app in iOS 10 with Appium, we have to use Xcode 8 or newer with Appium 1.6 or newer.

However, the ***official*** inspector for Appium 1.6 is not there yet. Obviously, it is not ideal to use the client's get page sourcemethod to get the entire source tree of the current view then manually extract the element locator. People still need a GUI like inspector to easily identify the locator of the element to build up Appium tests.

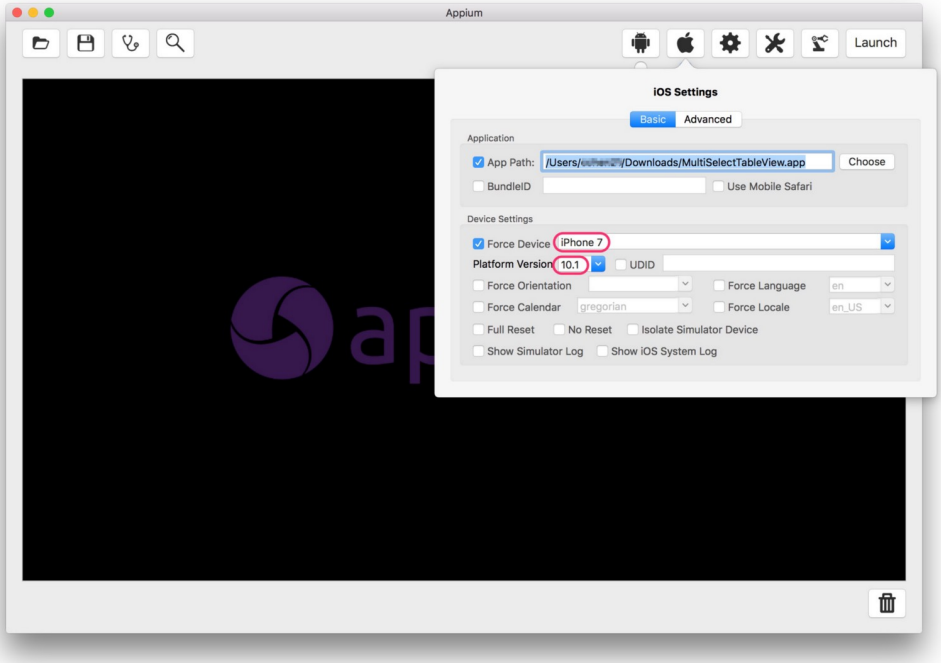
To do this, we actually have three options currently:

Note: the following are based on macOS system.

1. The good old Appium inspector, with a bit workaround

In the Appium GUI, iOS settings:

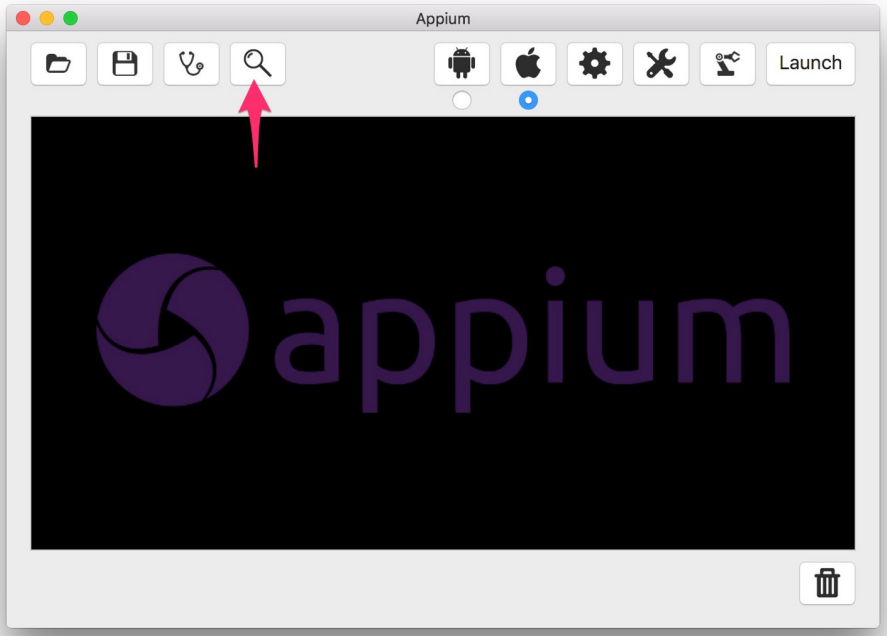
> **App Path:** select the path to your app that needs to be inspected
>
> **Force Device:** select *iPhone 7*
>
> **Platform Version:** manually *input 10.1*



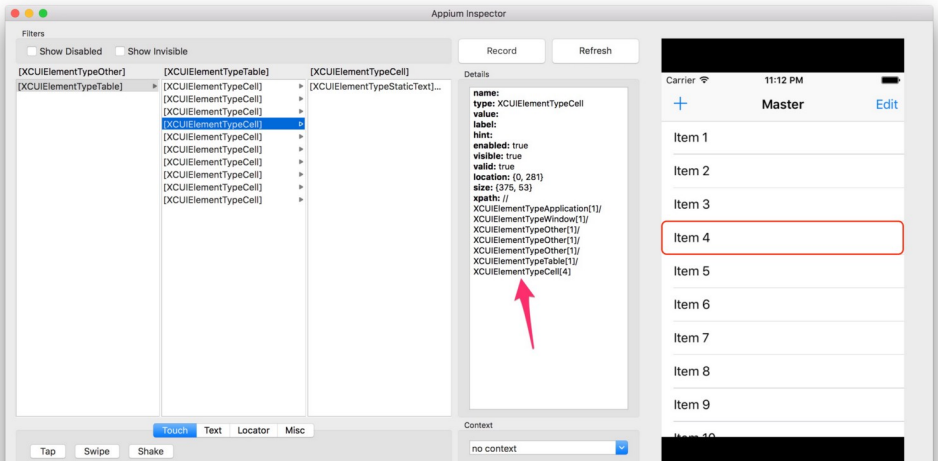
Then **DON'T click on Launch** to start Appium server through the GUI. Instead, do a `npm install appium` then start Appium 1.6 server in your terminal:

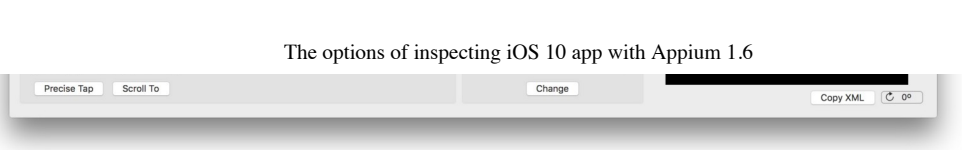
```
$ npm install appium
$ ./node_modules/.bin/appium
```

Once the Appium server starts, click on the Inspector icon on the GUI:



Then that's it. Wait for a while, you'll see Appium first installs WebDriverAgent into the simulator, then the app you want to inspect. And Appium Inspector will open with the full **XCUIElementType** locator information you are looking for:





2. A new web based inspector maintained by Macaca team

This is a slick web based inspector created by the Macaca team, that is from the mighty **Alibaba Inc.** *Macaca* basically provides a full suites of open source solutions based on Selenium WebDriver, for both Web and mobile. This inspector is just one of the tools.

I met some of the maintainers back in July 2016. They are a bunch of very talented engineers who spend their extra hours on this project. If you are interested, I do recommend playing around with their tools.

Back to the inspector. Here are the steps to set it up:

> Find the UDID for the simulator where you want to install and inspect the app:

```
$ xcrun simctl list
```

> Install Macaca inspector:

```
$ npm install app-inspector
```

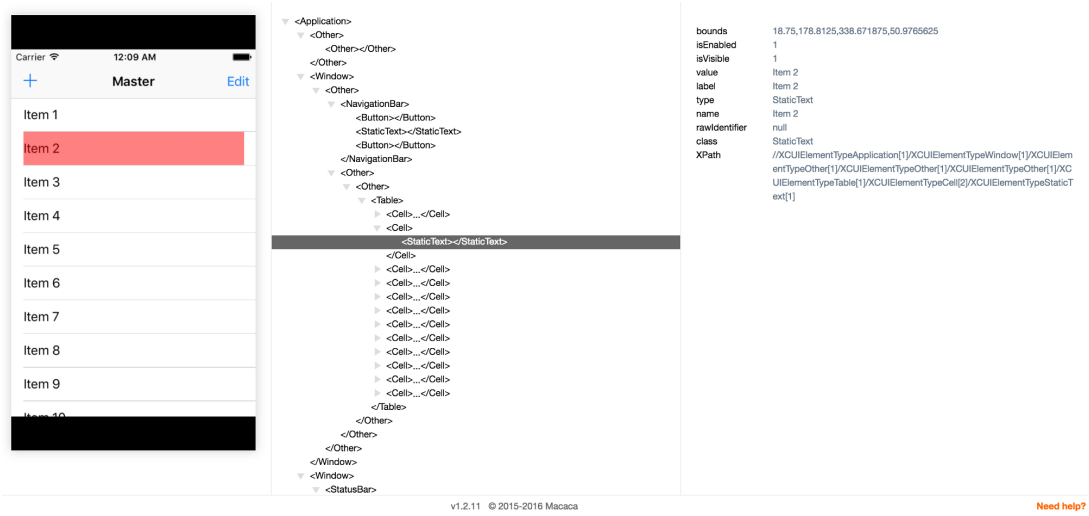
> Start Macaca inspector with the simulator's UDID, e.g.:

```
$ ./node_modules/.bin/app-inspector -u 6EE5E44A-F4F2-
4A8A-AB5C-7A7FC9CC3512
```

Install the app to the booted simulator, e.g.:

```
$ xcrun simctl install 6EE5E44A-F4F2-4A8A-AB5C-
7A7FC9CC3512 ~/Downloads/MultiSelectTableView.app/
```

Then go to the link showed from the terminal and start to inspect the app:



3. A new web based inspector inspired by Selendroid Inspector

This is another new Appium iOS Inspector created by Mykola Mokhnach who is an active contributor to both Appium and WebDriverAgent projects, including the very anticipated https://github.com/facebook/WebDriverAgent/pull/308

I haven't tried it out by myself, but based on feedback from the Appium forums, it definitely works. It requires to use a bootstrap Appium test to load the app into simulator and set break point to where you want to inspect the UI so you can go to the web browser to inspect it.

Conclusion

While I like the idea of web based inspector, both are missing a convenient feature to the majority of testers: **automatic app installation**.

> Macaca needs to manually extract the UDID and install the app using *xcrun*. I opened a feature request issue several month back, but was turned down.
>
> Mykola Mokhnach's inspector needs to execute an Appium test then set a break point so you can continue to inspect.

So, for a one-in-all Appium inspector solution, I would stay with option 1 for now.