**Mac Machine Set up for Mobile Testing Using Appium**

1. **Prerequisites to Start Testing iOS Application:**

There are two ways to get the application from the Dev team for testing:

* 1. We should generate the build on the same machine where simulator is installed i.e., build should be as per machine configuration otherwise it may not work. To make this happen we need access to code repository so that we can clone it on automation machine and generate the build. This will also require provisioning certificates to install on our automation machine and developer's credentials on Apple account. If Firebase Cloud Messaging (FCM) push notifications are implemented then we may also need to register our machine's APM certificates on Firebase account. (Please refer the **section: 14**, How to generate the build).
  2. You can provide us the app bundle (file with .app/.ipa extension) which we try installing on simulator/real device directly.

**Note:**

1. **To test application on simulator we need application with .app extension and on real device we need .ipa file.**
2. **You have no need to create simulator like emulator is created using android studio for android automation, In iOS all the simulators (iPhone models) are directly available in Xcode.**

There are two ways to install application on devices:

1. Installation through TestFlight: The dev team sends an invitation mail to install the application on real device.
   * + Install TestFlight app on the device from the Appstore
     + Open the application and enter the redeem code (provided by dev team/client) which is provided through the invitation
     + It will display the application for installation.
     + Click the application to be installed.
2. Installation through Drag & Drop: The dev team shared the app bundle (.ipa/.app file). We can store the application on machine and simply drag the application from stored location to the device and drop it to be installed on the device.
3. **Install Xcode:**

**2.1: Download from App store**

* 1. Open the App Store on mac

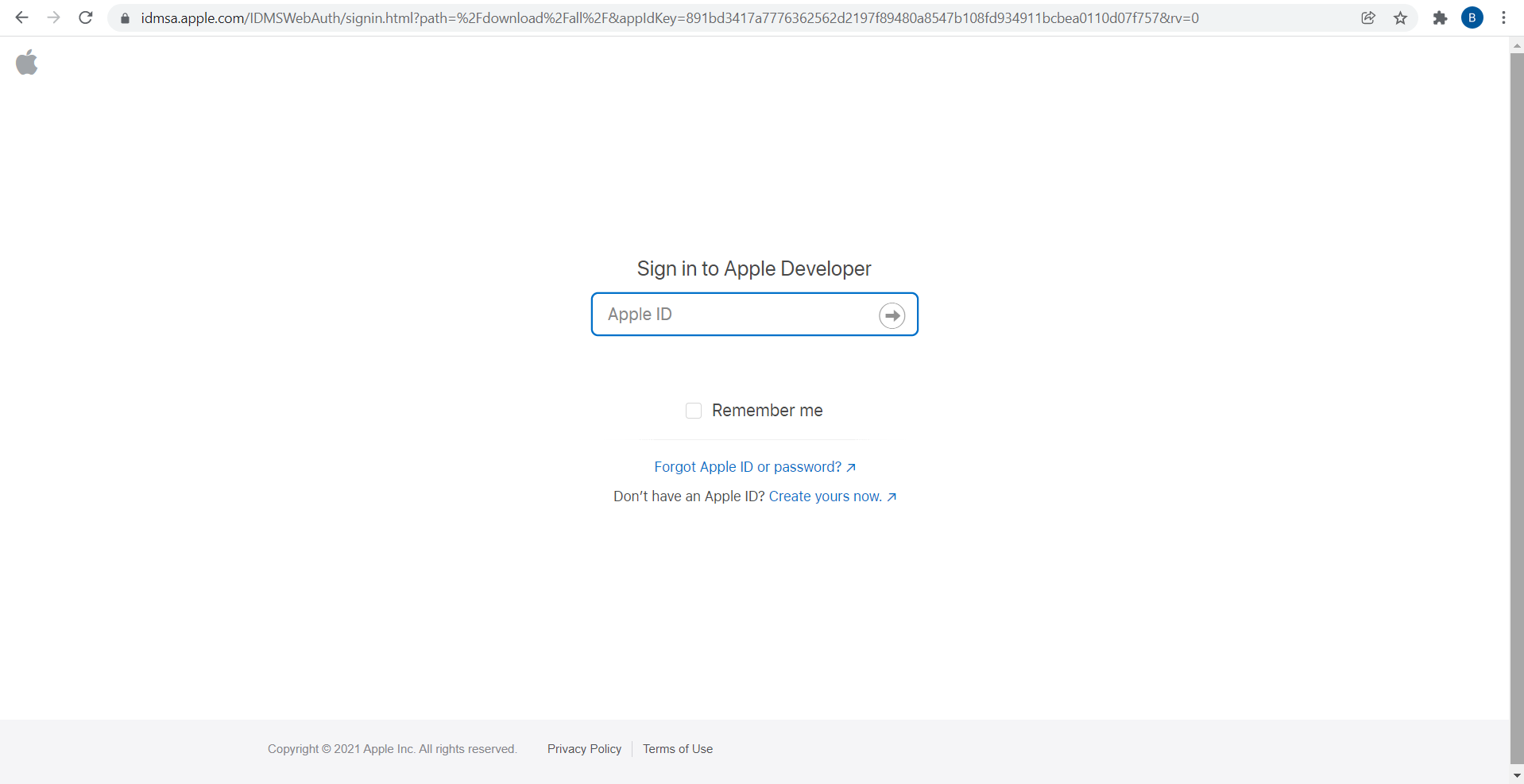
Sign in with your Apple ID. If you did not sign in.

* 1. Search for Xcode

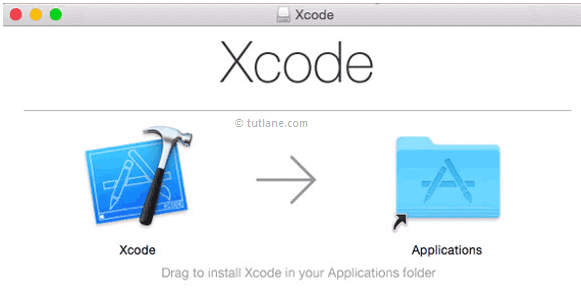


* 1. “GET” button will be displayed, Click on GET then Install Xcode.

**2.2: Download via the Developer site for a specific version**

****

1. You can find the DMGs or XIPs for Xcode and other development tools on <https://developer.apple.com/download/more/> (requires Apple ID to login). You must login to have a valid session before downloading anything below.
2. Type in the version that you'd like, and download the Xcode\_xx.x.x.xip file.
3. Once the file is downloaded, click on .xip to extract it. Your laptop will extract it to the same folder you downloaded it to. This extraction process is automatic.
4. Drag application to the Applications folder.



1. After successful copy, you will find Xcode in Applications.
2. **JDK Install, Set JAVA\_HOME and Path variables:**

Link to download**:** <https://www.oracle.com/java/technologies/downloads/>

**3.1: Installing JDK using a .dmg file:**

1. Double-click on the .dmg file to begin the installation. The system verifies the .dmg file, then opens a Finder window with the icon of an open box (package) and the name of the .pkg file.
2. Double-click the package icon to start the installation app. A window appears with a status bar with text similar to the following:

Verifying "JDK 8 Update 231.pkg"...

Then, a window appears with text similar to the following:

"JDK 8 Update 231.pkg" can't be opened because Apple cannot check it for malicious software.

This software needs to be updated. Contact the developer for more information.

Click **OK**.

1. Open **System Preferences**, click **Security & Privacy**, then click the **General** tab. In the section **Allow apps downloaded from:** the following text appears:

"JDK 8 Update 231.pkg" was blocked from use because it is not from an identified developer.

Click the **Open Anyway** button.

1. A window appears with text similar to the following:

"JDK 8 Update 231.pkg" can't be opened because Apple cannot check it for malicious software.

This software needs to be updated. Contact the developer for more information.

Click the **Open** button.

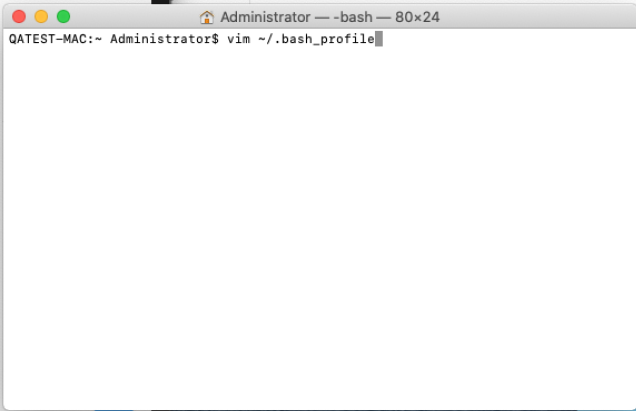
1. A window appears with the following text:

Installer is trying to install new software. Enter your password to allow this.

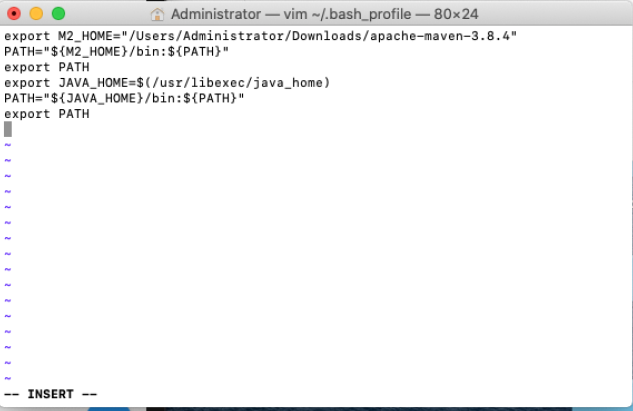
Enter your username and password, then click the **Install Software** button.

**3.2: Path Set up:**

1. Open terminal window



1. Type command: $ vim ~/.bash\_profile and Hit Enter



1. Type command: $ export JAVA\_HOME=$(/usr/libexec/java\_home)
2. $ export PATH=${JAVA\_HOME}/bin:${PATH}
3. and press Escape key for Save changes.
4. Type command: :(colon)wq, It will Save and close .bash\_profile file.
5. Type command: source ~/.bash\_profile and hit Enter
6. Now we can check the value of the JAVA\_HOME variable:
7. Type Command: echo $JAVA\_HOME
8. **Maven Set up:**

Link to Download: <https://maven.apache.org/download.cgi>

Download the “Binary tar.gz archive” file.

Extract the downloaded file in the location you want.

**4.1: Maven Path set up**:

In terminal run command



* open ~/.bash\_profile

The bash profile will be opened, Insert



* export M2\_HOME= “path of the extracted apache maven”
* PATH=${M2\_HOME}/bin:${PATH}
* export PATH

In Terminal run command

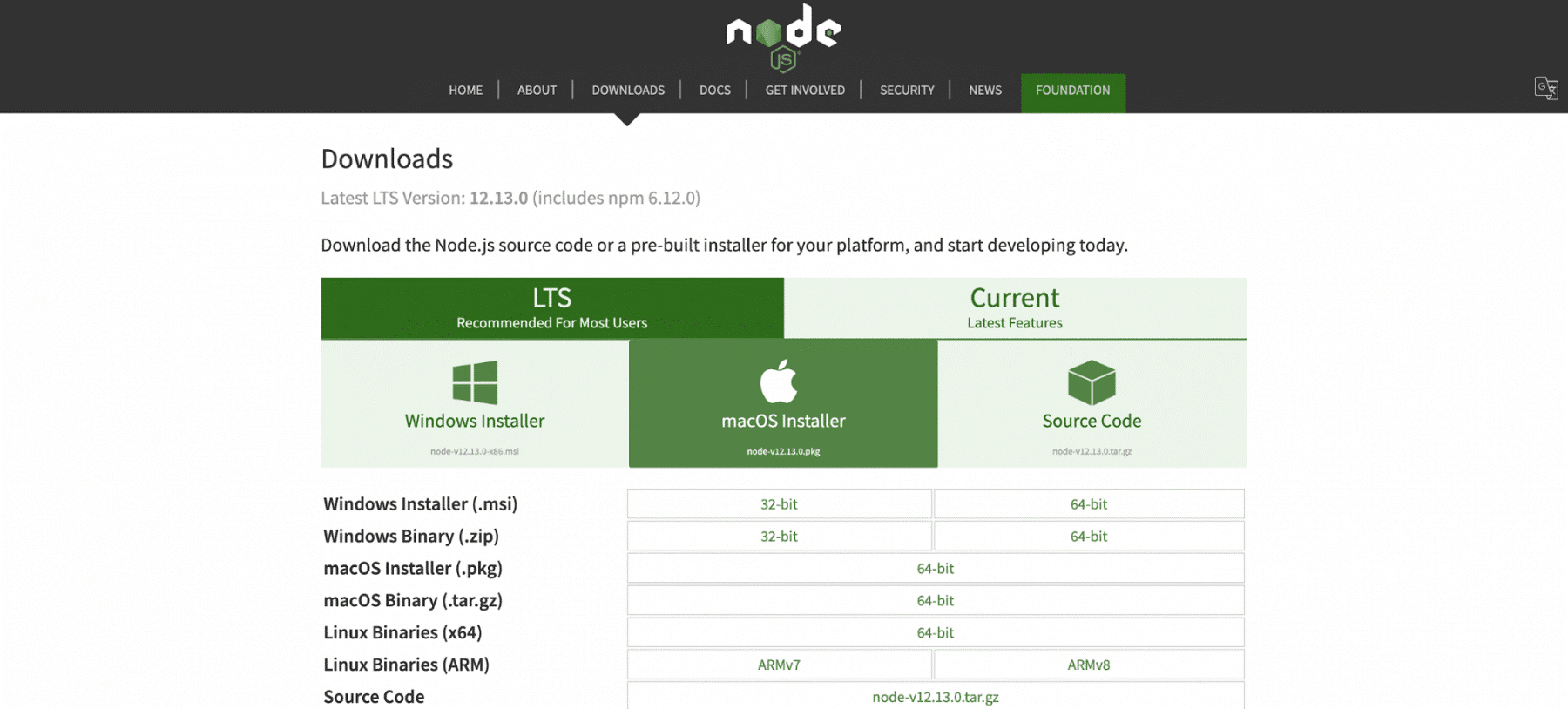
* source ~/.bash\_profile

It will save the details in bash profile.

1. **NodeJS Installation:**

Download and Install NodeJS in the machine.

Link to Download: <https://nodejs.org/en/download/>



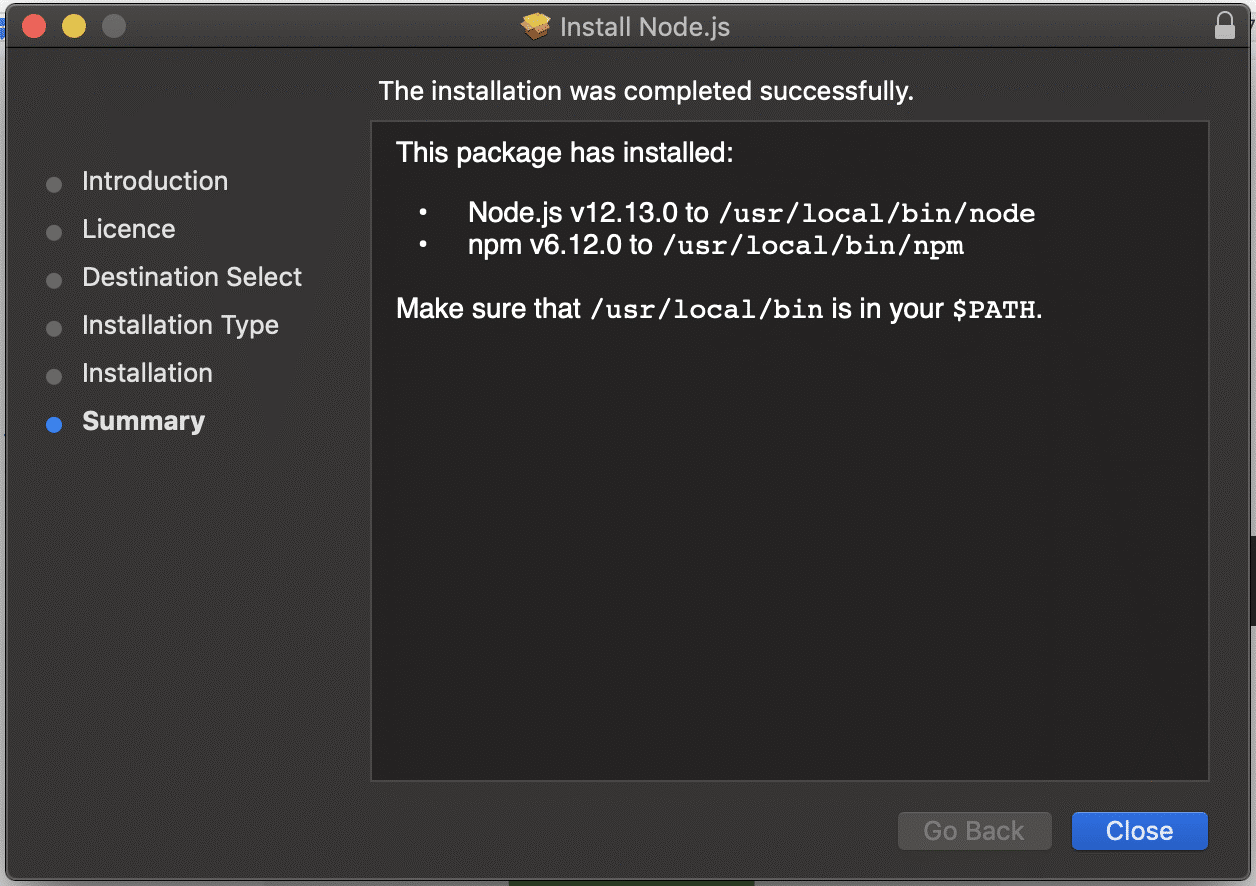
1. Download the .pkg Installer

* Click on the “macOS Installer” option to download the .pkg installer. Make sure you download it to your desired location.

1. Run Node.js Installer

* Now, your installer is ready to run. However, it will not take your much time. So, let’s explain in detail here.

Introduction > Continue  
License > Select Continue > Agree  
Installation Type > Install > Authenticate with your Mac to allow the Installation > Install Software  
Summary > Close



1. Verify Node.js Installation

* To verify whether you have properly installed Node.js in your macOS, run the following command in your terminal: *node -v*

The command we ran tests the version of Node.js that's currently installed.

1. **Brew Installation:**

Install brew on the machine. It will help us to install all other required tools.

Run the below command on terminal, It will install brew in the machine.

/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"

1. **Libimobiledevice Installation:**

we will install libimobiledevice, an open-source package which is able to communicate with iOS devices.

Run below command on terminal:

brew install libimobiledevice

1. **Ios-deploy Installation:**

Appium also uses a package called ios-deploy for transferring iOS apps onto your device, so let's install that too.

Run below command on terminal:

brew install ios-deploy

1. **Carthage Installation:**

WDA itself requires an iOS dependency manager called Carthage. Since Appium will be automatically building the WDA app, we need to install Carthage so it is available to the WDA bootstrap process.

Run below command on terminal:

brew install carthage

Sometimes there will be issue with Linking Carthage:

For that run command:

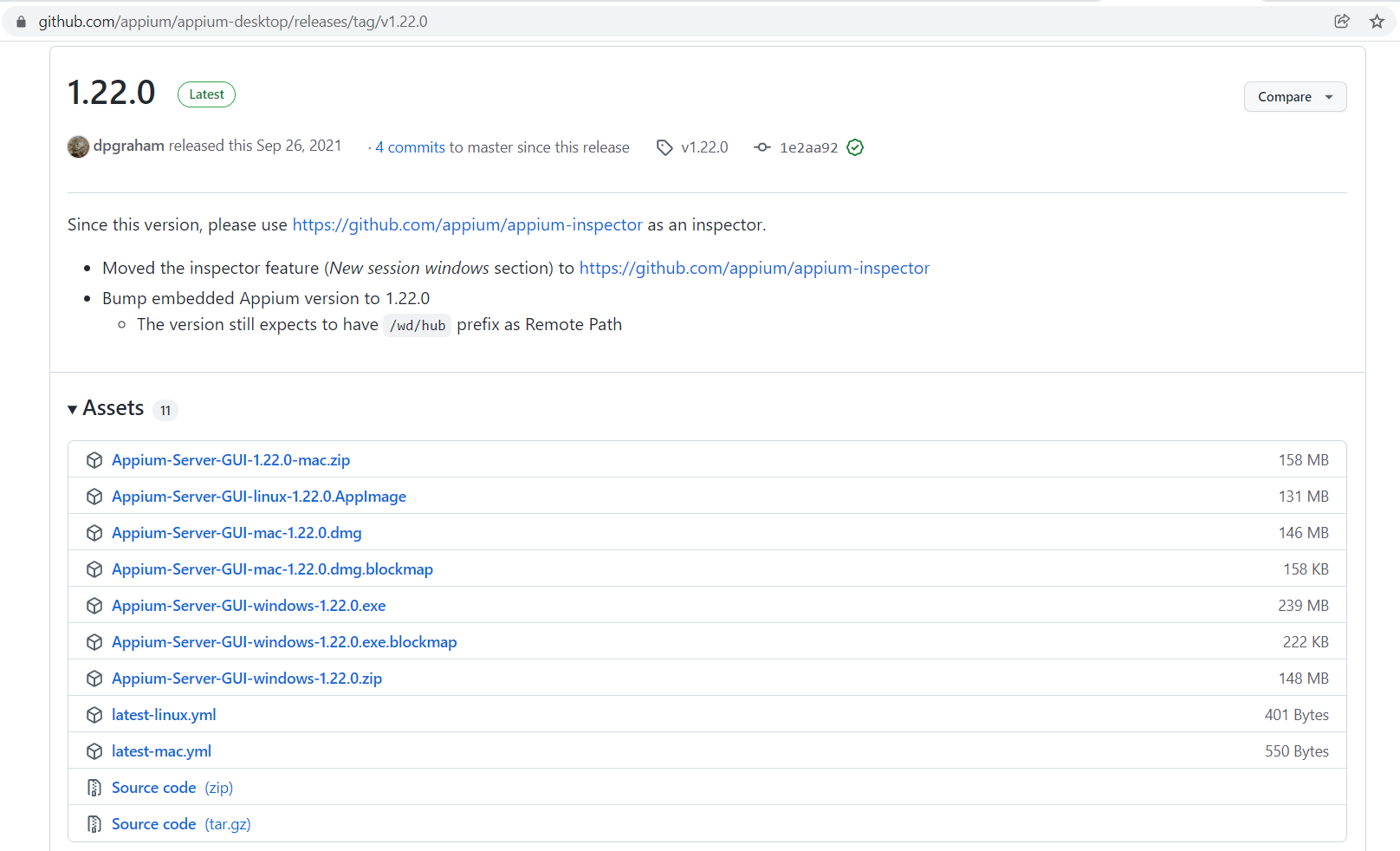
brew link carthage

Still, you are getting some issue. Go to the file location which asked for access. Change access to Read & Write for that Path and Rerun the command.

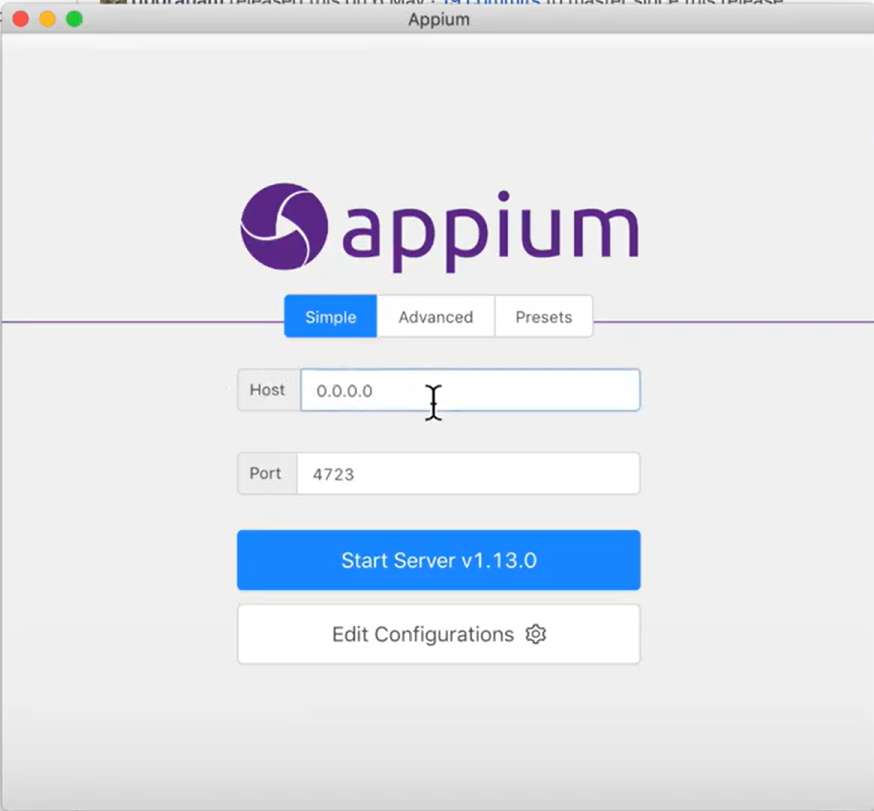
**Note**: If some point you get issue regarding permission to certain directory. Run command “sudo chown -r [owner] directory path”. Then it will ask for your system password. Now, you can install the software.

1. **Appium Server GUI:**

Link to Download: https://github.com/appium/appium-desktop/releases/tag/v1.22.0

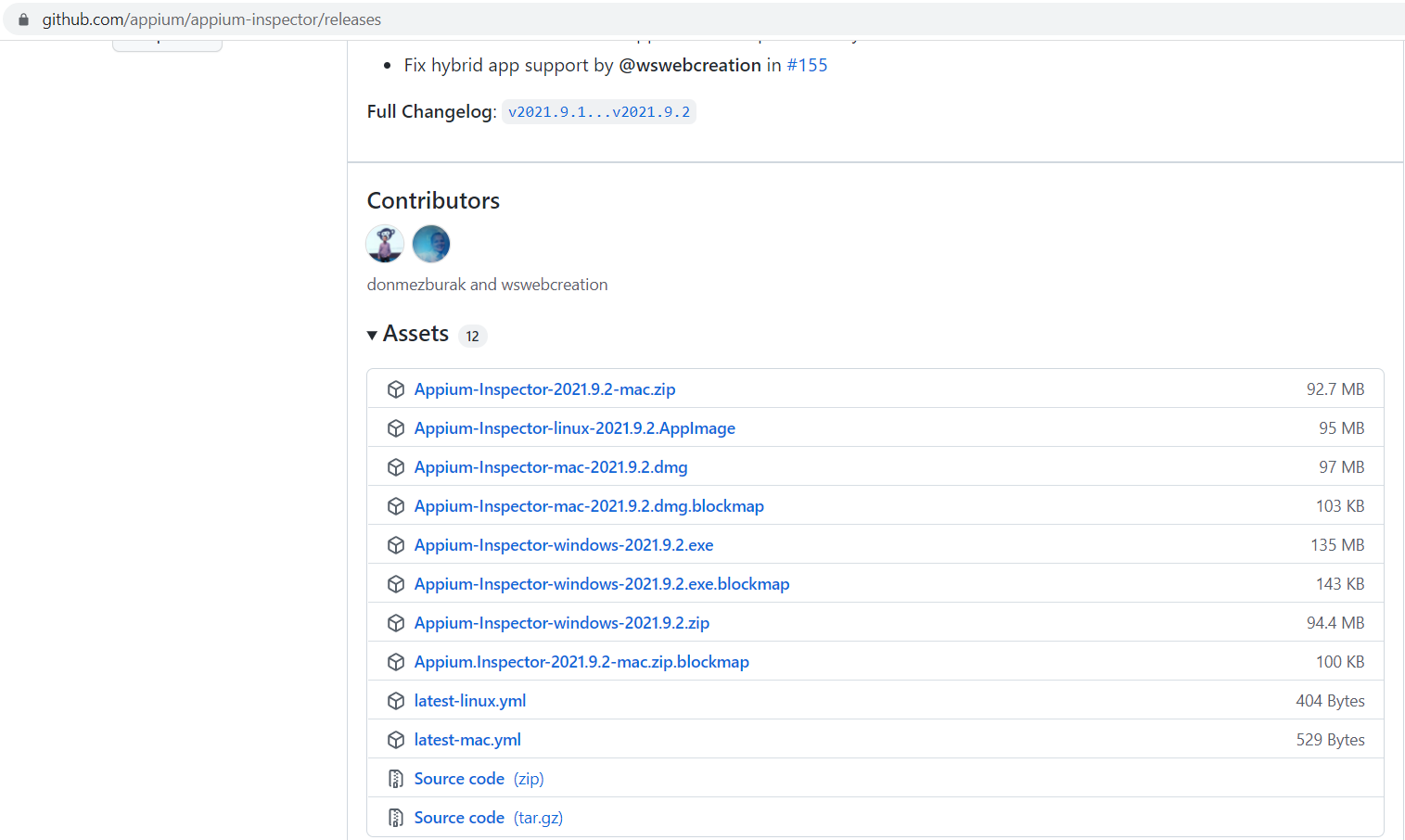


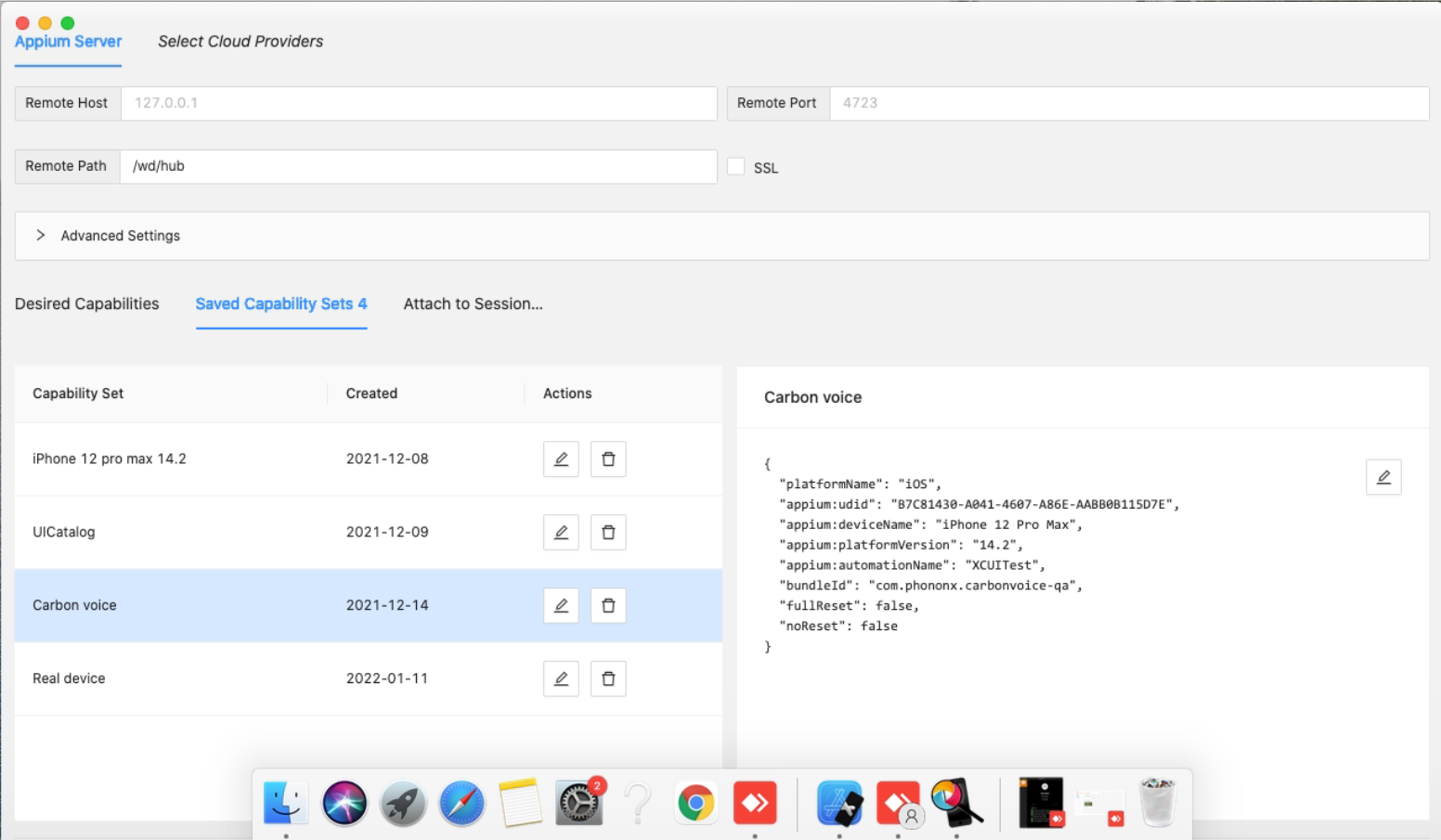
1. Download .dmg or .zip file for mac.
2. Click on the download application, Window will be opened.
3. Drag Appium server GUI to Applications, Once completed
4. You can search for Appium Server GUI on Applications



1. **Appium Inspector:**

Link to Download: https://github.com/appium/appium-inspector/releases



1. Download .dmg or .zip file for mac
2. Open the downloaded file, it will be opened a window
3. Drag Appium Inspector to Applications, once completed
4. Now search for Appium Inspector in Applications, and open the application
5. Open the application and in Remote Path, specify “/wd/hub”.
6. Add capabilities in Desired Capabilities as shown in below picture
7. The Json Payload generated after adding desired capabilities will look like below:

{

"platformName": "IOS",

"appium:udid": "2825A0CF-9687-4ABF-8BAD-619BB9D230A2",

"appium:automationName": "XCUITest",

"appium:platformVersion": "15.0",

"appium:deviceName": "iPhone 12 Pro",

"appium:bundleId": "specify bundle id"

}

When your Appium tests run, they are using an Appium client to send WebDriver commands to the Appium server. The Appium server installs both your app and WDA on the iOS device, waits for WDA to start, then forwards your test commands to WDA. WDA in turn executes XCUITest API functions on the device, corresponding to the commands it receives from the Appium server. In this way, we are able to arbitrarily interact with the user interface of an iOS device.

**11.1: The Capabilities**

We have the app running on our device, now let's write a simple automated test which will launch our app and look for a particular set of words on the screen. Because all of our Apple setup has been done (hopefully) correctly, all that we need to do from the Appium side of things is use the right set of capabilities:

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("platformName", "iOS");

capabilities.setCapability("platformVersion", "12.0.1");

capabilities.setCapability("deviceName", "iPhone 8");

capabilities.setCapability("udid", "auto");

capabilities.setCapability("bundleId", "<your bundle id>"); Or capabilities.setCapability("app", "Path of the app with name");

capabilities.setCapability("xcodeOrgId", "<your org id>");

capabilities.setCapability("xcodeSigningId", "iPhone Developer");

capabilities.setCapability("updatedWDABundleId", "<bundle id in scope of provisioning profile>");

The trick here is knowing how to fill all of these out!

**platformName** is iOS (as you would no doubt expect).

**platformVersion** is the version of iOS our app is running, 14.2 in my case.

**deviceName** does not actually matter for us, since we have plugged in a real device and will select that device using the udid desired capability. Appium still requires us to supply a value for deviceName, though, so I put iPhone 8.

**udid** is the unique ID of the device we want to run our test on. We could find our device udid by running the command instruments -s devices in the terminal, but since we only have a single device plugged in, we can put auto and Appium will automatically find the udid of the device for us and use it.

Command to get list of Simulators and Real devices: xcrun simctl list

**bundleId** is the special iOS-internal name of our app, which is set in the same app-settings form where we selected our Team in Xcode.

Command to get the bundleId of the application stored in machine: osascript -e ‘id of app“location of the app”’

**xcodeOrgId** is the "Organizational Unit" value we made a note of earlier. It is the ID of the Developer Team which signed the certificate used to create the app.

**xcodeSigningId** is the first part of the "Common Name" associated with the developer certificate. Since Xcode set this up for us, it is almost always iPhone Developer but could be something different for you if you are automating a different iOS device.

**updatedWDABundleId** is used by Appium to trick your device into allowing Appium to install WDA on it. You might have wondered, given how many hoops you had to jump through to get your app running on a device, how Appium is able to get WDA on the device. The short answer is that, typically, it can't. WDA's bundle ID (com.facebook.webdriveragent) will not show up as an App ID in any of your provisioning profiles, so the app would not be allowed to run on your device. But given that Appium has the WDA code, it can actually change the bundle ID on the fly, so that when WDA is built and signed it will be allowed past Apple's security restrictions. What this means is that you must supply a bundle ID value that is allowed by an App ID in your provisioning profile. We typically recommend using wildcard App IDs (like com.test.\*), so that we can give a new bundle ID to WDA of com.test.webdriveragent. You could also give it the same bundle ID as your app, but that could cause some confusion in the system later on. If you prefer, you can omit this capability and simply open up the WDA project inside of Appium, and make all these modifications yourself using Xcode.

**11.2: The Test**

Your actual test code, of course, is up to you to define! You'll simply instantiate a Driver and run your test as in any other case. All the heavy lifting is done by Appium in response to the capabilities above and in the context of correctly-signed apps and correctly-provisioned devices. Here's a step-by-step guide of what to do:

1. Start the Appium server (by opening the Appium Desktop app or using the CLI).
2. Run the Java test you wrote including the capabilities above, and watch as the iOS device opens your app!
3. Make sure the device is unlocked, and if it asks you to "Trust the Computer", tap the button to trust our Mac. You'll need to do this the first time.
4. If anything goes wrong, check the logs which Appium prints. There is another helpful [AppiumPro post](https://appiumpro.com/editions/10-anatomy-of-logging-in-appium) about how to read these logs.

Note that the **bundleId** capability can only be used for apps which are already installed on our iOS device. We installed our app manually using Xcode, so the app is already there. If we make changes to the app code, we will have to click the ▶ button in Xcode in order to install the latest version of our code on the device. Then we can run our Appium tests again.

Alternatively, we could use the **app** capability and set it to the path of an .app/.ipa file on disk. This must be an app bundle generated in Xcode and signed correctly.

Fail Scenarios:

1. XCode version is not found in /Library/Developer/info.plist

Run command “sudo xcode-select -s /Applications/Xcode.app”

1. Settings -> Developer -> **Enable UI Automation** on Real Device
2. Settings -> Safari -> Advanced -> **Web Inspector** and **Remote Automation**
3. **WebDriverAgent(WDA) set up:**

**12.1: For Simulator:**

1. Go to the WebDriverAgent directory e.g., /Applications/Appium\ Server\ GUI.app/Contents/Resources/app/node\_modules/Appium/node\_modules/Appium\_webdriveragent/WebDriverAgent.xcodeproj
2. Open WebDriverAgent.xcodeproj in Xcode.
3. Select WebDriverAgentRunner project
4. Select your Simulator you'd like to run automated tests on as build target
5. Select Product->Test from the main menu

Xcode should successfully build the project and install it on the Simulator, so you'll see the icon of WebDriverAgentRunner application on the springboard.

**For Real Device some extra configuration needs to be done. Please refer the documentation:**

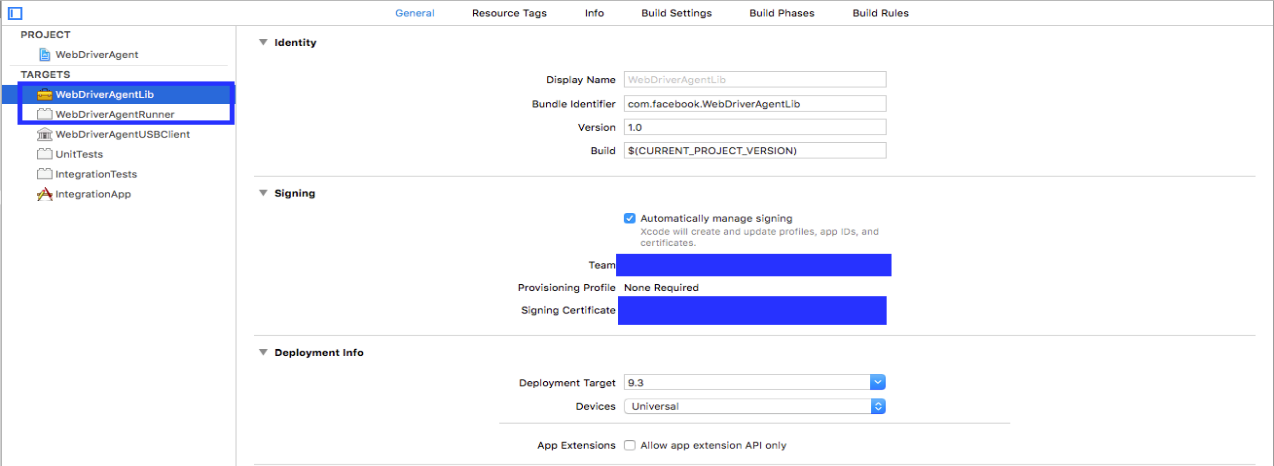
<https://github.com/appium/appium-xcuitest-driver/blob/master/docs/real-device-config.md>

**12.2: For Real Device**: (If you have a free Apple developer account)

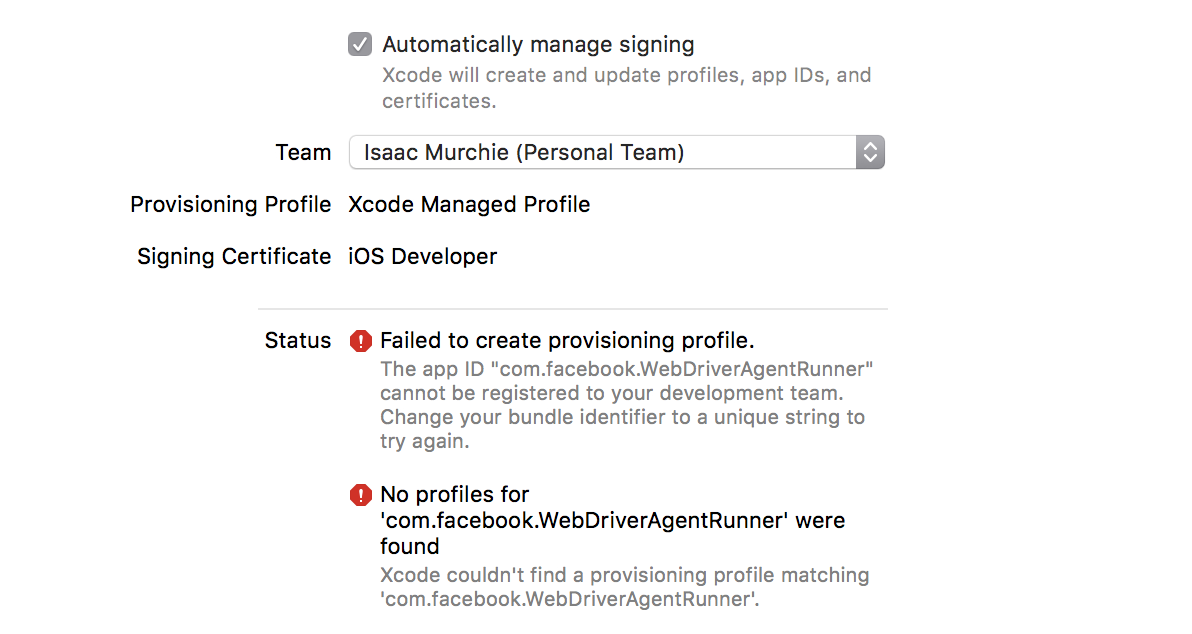
1. Given this installation location, project will be found in /Applications/Appium\ Server\ GUI.app/Contents/Resources/app/node\_modules/appium/node\_modules/appium-webdriveragent. Open a **terminal** and go to the location, then run the next script.

mkdir -p Resources/WebDriverAgent.bundle

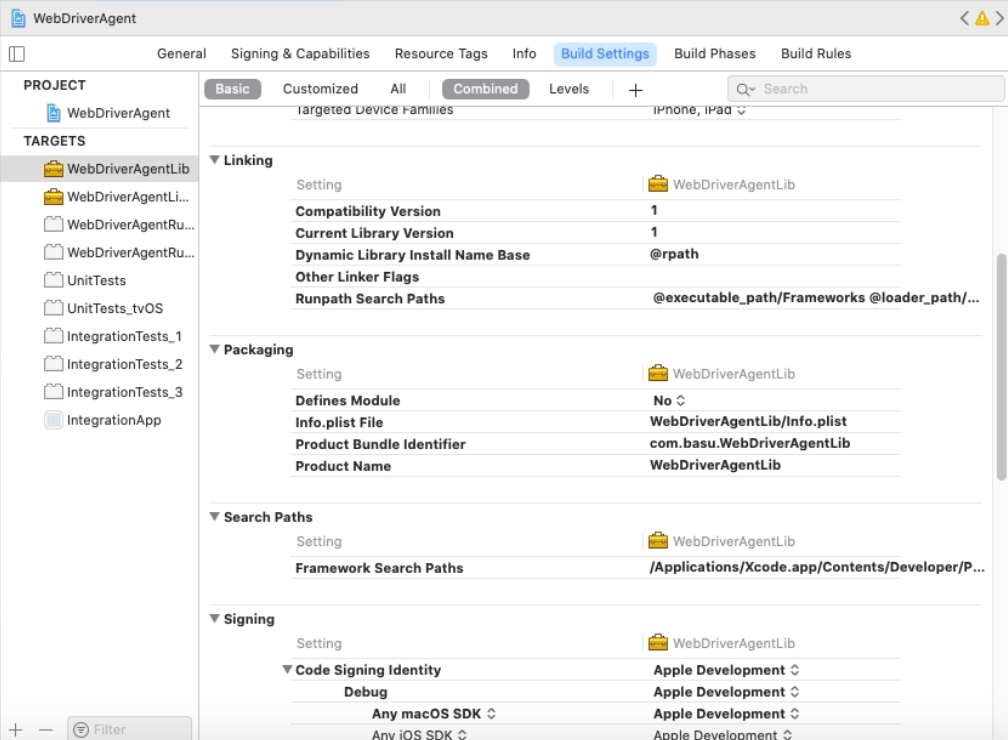
1. Open WebDriverAgent.xcodeproj in Xcode. For WebDriverAgentLib, IntegrationApp and WebDriverAgentRunner targets, select "Automatically manage signing" in the "General" tab, and then select your Development Team. This should also auto select Signing Ceritificate. The outcome should look as shown below:

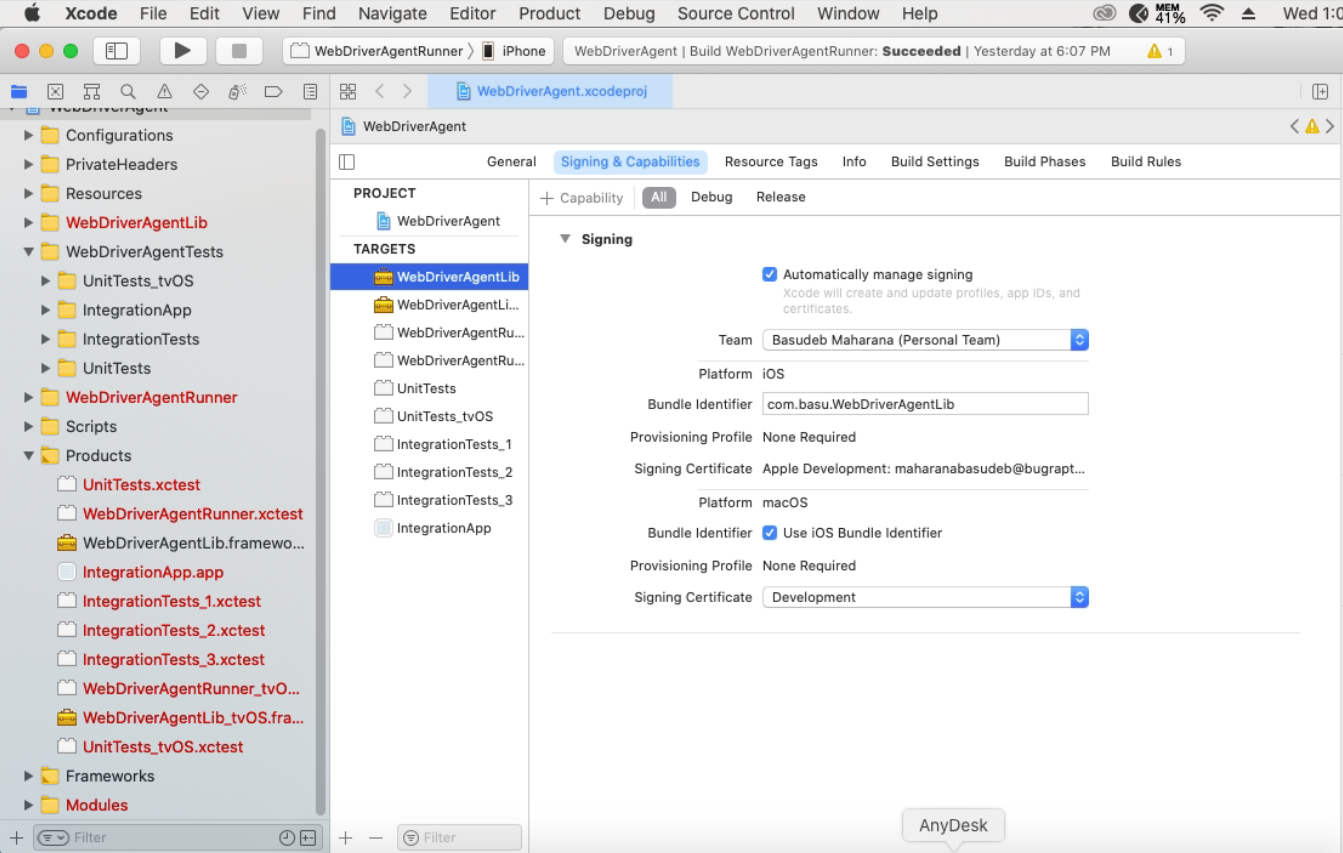


1. Xcode may fail to create a provisioning profile for the WebDriverAgentRunner target:



1. This necessitates manually changing the bundle id for the target by going into the "Build Settings" tab, and changing the "Product Bundle Identifier" from com.facebook.WebDriverAgentRunner to something that Xcode will accept:





1. Now Select the Real device from the device selection box



1. Click Product> Build to generate the build for real device.
2. Finally, you can verify that everything works. Test the project:

Go to WebDriverAgent directory in terminal and run following command:

xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination 'id=<udid>' test

1. If the command fails, try passing the -allowProvisioningUpdates:

xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination 'id=<udid>' -allowProvisioningUpdates test

**E.g.,** xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination 'id=7345b84bc9354e524643a49586f5f22e422a49bc' test -allowProvisioningUpdates

1. Then Trust the WDA Application on real device:

Go to Settings>General>Profile and Device Management

Select the Profile and Trust it.

1. Then run following command One more time:

xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination 'id=<udid>' test

1. If this is successful, the output should end with something like:

Test Suite 'All tests' started at 2017-01-23 15:49:12.585

Test Suite 'WebDriverAgentRunner.xctest' started at 2017-01-23 15:49:12.586

Test Suite 'UITestingUITests' started at 2017-01-23 15:49:12.587

Test Case '-[UITestingUITests testRunner]' started.

t = 0.00s Start Test at 2017-01-23 15:49:12.588

t = 0.00s Set Up

1. Now, you can run your script for real device.

Note: Don’t define DesiredCapabilities for WDA. (If you are doing set up in this way using free Developer Account.)

1. **Steps need to be followed to run Appium server through code are as following:**
2. Install node into the system and with node install Appium.
3. Create an object of AppiumServiceBuilder class

AppiumServiceBuilder appiumServiceBuilder = new AppiumServiceBuilder();

1. With the help of appiumServiceBuilder object call method withAppiumJS(location) and give the location of main.js file located at

/Applications/Appium\ Server\ GUI.app/Contents/Resources/app/node\_modules/appium/build/lib/main.js

This is the path for mac.

Path of node executable file:

/usr/local/bin/node

**Note:**

The path to executable appium.js (version of Appium 1.4.x and lower) or main.js (version of Appium 1.5.x and higher).

1. With the help of appiumServiceBuilder object call another method using DriverExecutable(location) and give the location of node.exe
2. With the help of withIPAddress(IP address) method you can specific the like in which IP address server will run if not called default value will be used.
3. With the help of usingPort(port number) method you can tell Appium in which port your Appium server will run if not called than default port will be used or use can use method usingAnyFreePort() it will use any free port available.
4. After all these 6 steps than you have to create variable of class AppiumDriverLocalService. AppiumDriverLocalService objectName; after that we will call the static method(buildService(appiumServiceBuilder)) of AppiumDriverLocalService class and pass the object of appiumServiceBuilder

AppiumDriverLocalService server = AppiumDriverLocalService.buildService(appiumServiceBuilder);

1. With the help server object we will call methods of AppiumDriverLocalService class like to start the server we will call server.start(); after that if you want stop the server than call server.stop();

For reference you can refer to this website

<https://www.seleniumeasy.com/appium-tutorials/how-to-start-appium-server-programmatically>

**Sample code:**

AppiumServiceBuilder builder = new AppiumServiceBuilder();

builder.withAppiumJS(new File("pathofmain.js+main.js"));

builder.usingDriverExecutable(new File("path to node executable file "));

builder.withIPAddress("127.0.0.1");

builder.usingPort(Integer.parseInt(port));

AppiumDriverLocalService service = AppiumDriverLocalService.buildService(builder);

service.start();

Define the below code in different tag by which it will be executed after execution of testcases are done.

service.stop();

1. **Ways to generate the build:**

**14.1: For Applications developed in Flutter**

Navigate to the Project directory after getting the source code. Then follow the below steps:

1. In Xcode, open Runner.xcworkspace in your app’s ios folder. This file will be present in the source code that we had cloned from git.
2. To view your app’s settings, select the **Runner** project in the Xcode project navigator. Then, in the main view sidebar, select the **Runner** target.
3. Select the **General** tab.

Verify the most important settings.

In the **Identity** section:

Display Name

The display name of your app.

Bundle Identifier

The App ID you registered on App Store Connect.

In the **Signing & Capabilities** section:

Automatically manage signing

Whether Xcode should automatically manage app signing and provisioning. This is set true by default, which should be sufficient for most apps. For more complex scenarios, see the [Code Signing Guide](https://developer.apple.com/library/content/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html).

Team

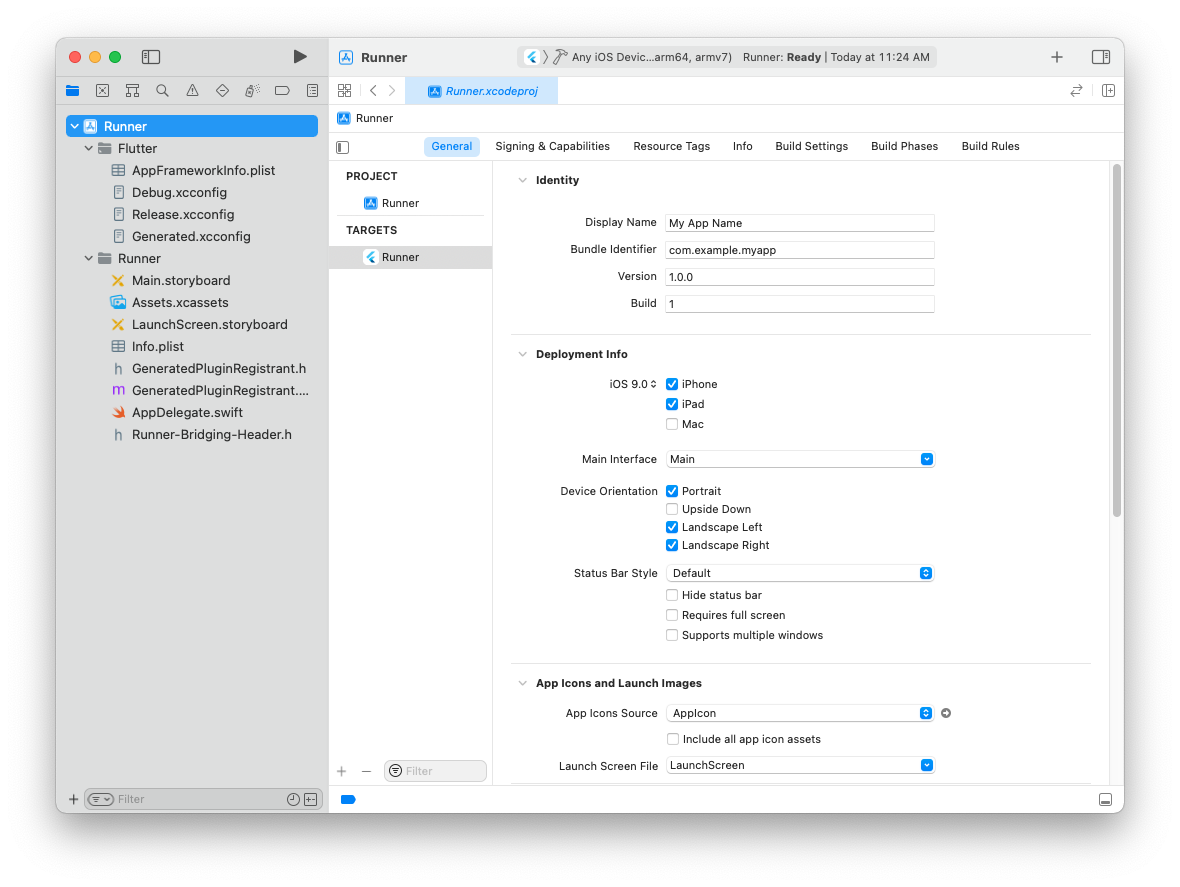
Select the team associated with your registered Apple Developer account. If required, select **Add Account…**, then update this setting.

In the **Build Settings** section:

iOS Deployment Target

The minimum iOS version that your app supports. Flutter supports iOS 9.0 and later. If your app or plugins include Objective-C or Swift code that makes use of APIs newer than iOS 9, update this setting to the highest required version.

The **General** tab of your project settings should resemble the following:

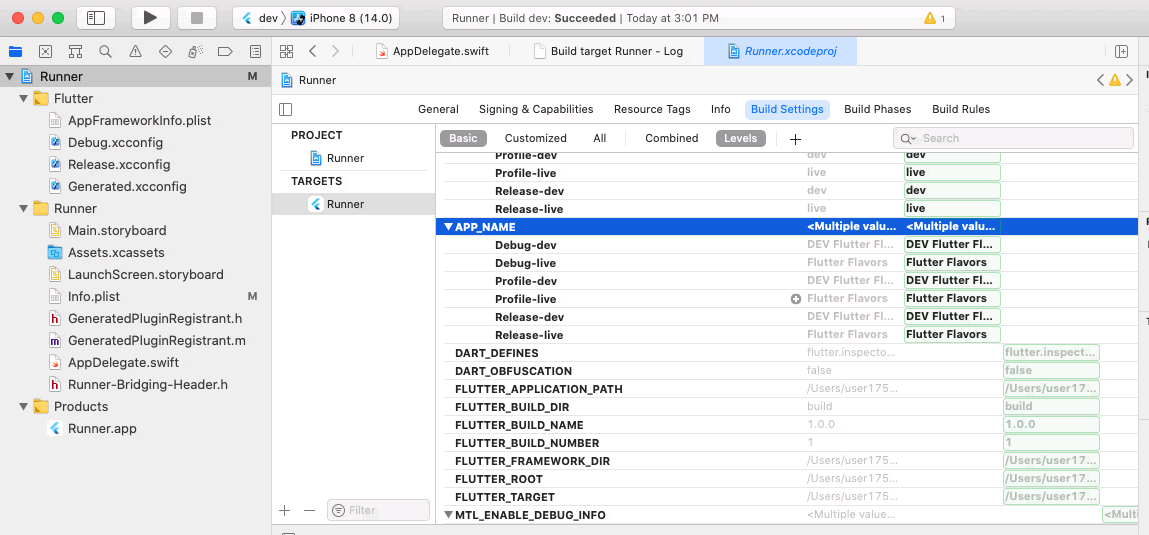


Finally, create a build archive:

Run flutter build ipa to produce a build archive to run on real device.

Open build/ios/archive/MyApp.xcarchive in Xcode. Click the **Validate App** button. If any issues are reported, address them and produce another build.

Run flutter build ios --simulator --flavor dev to produce a build archive which will run on simulator.



If you have any doubts please go through below URLs:

1. <https://www.chwe.at/2020/10/flutter-flavors/>
2. <https://docs.flutter.dev/deployment/ios>

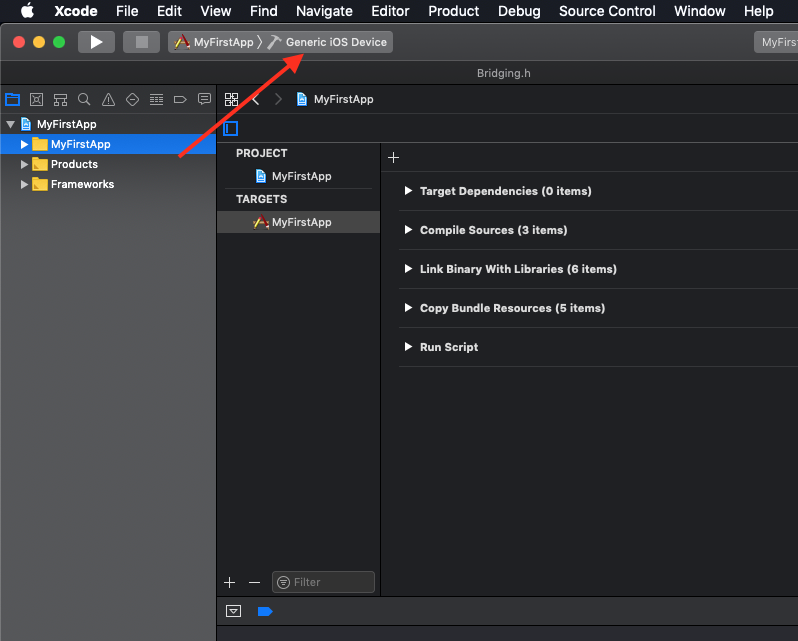
**14.2: For Native apps:**

Apple allows app distribution for testing on registered devices using an Ad-Hoc or Enterprise provisioning profile. The output file you create is an iOS App file (a file with an .ipa filename extension) that is then used to install your app on registered devices.

Navigate to the project path and open the project\_name.xcworkspace file in Xcode.

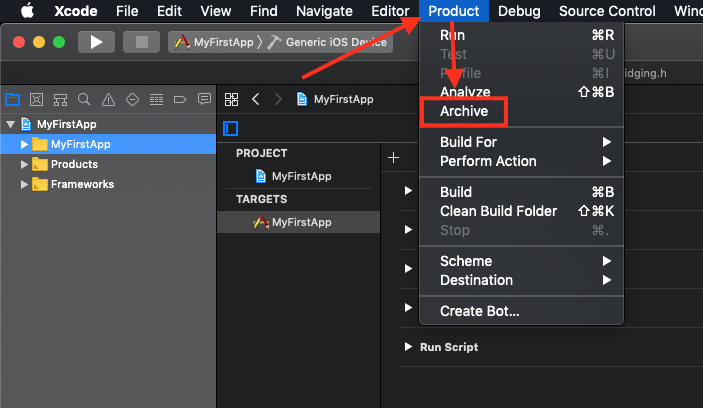
Create an archive of your app. Xcode stores this archive in the **Archives organizer**.

In the Xcode project editor, choose Generic iOS Device — or your device name from the Scheme toolbar menu. (You can’t create an archive of a simulator build). If a device is connected to your Mac, the device name appears in the Scheme toolbar menu. When you disconnect the device, the menu item changes to the Generic iOS Device.

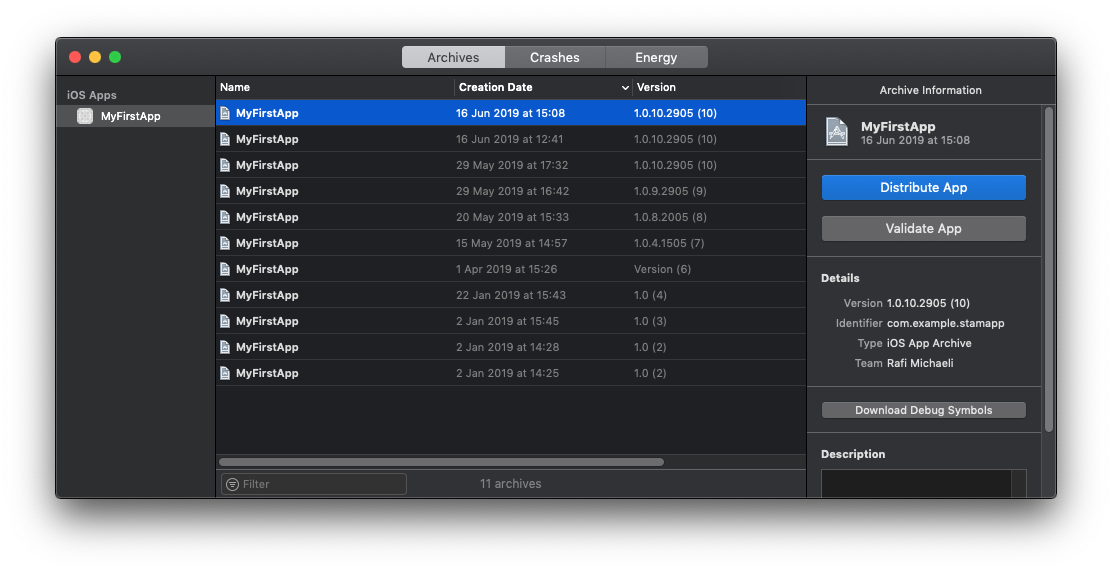


Choose **Product --> Archive**. The Archives organizer appears and displays the new archive.

Xcode runs preliminary validation tests on the archive. To create an IPA file, press the Distribute App button.

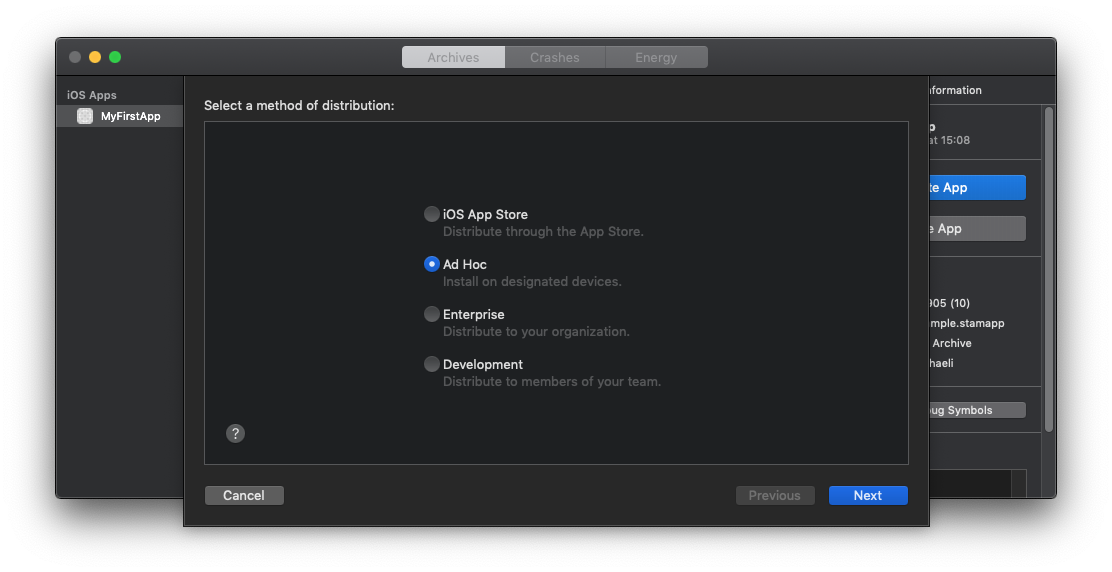


To create an iOS App file for testing, select an archive in the **Archives organizer**.

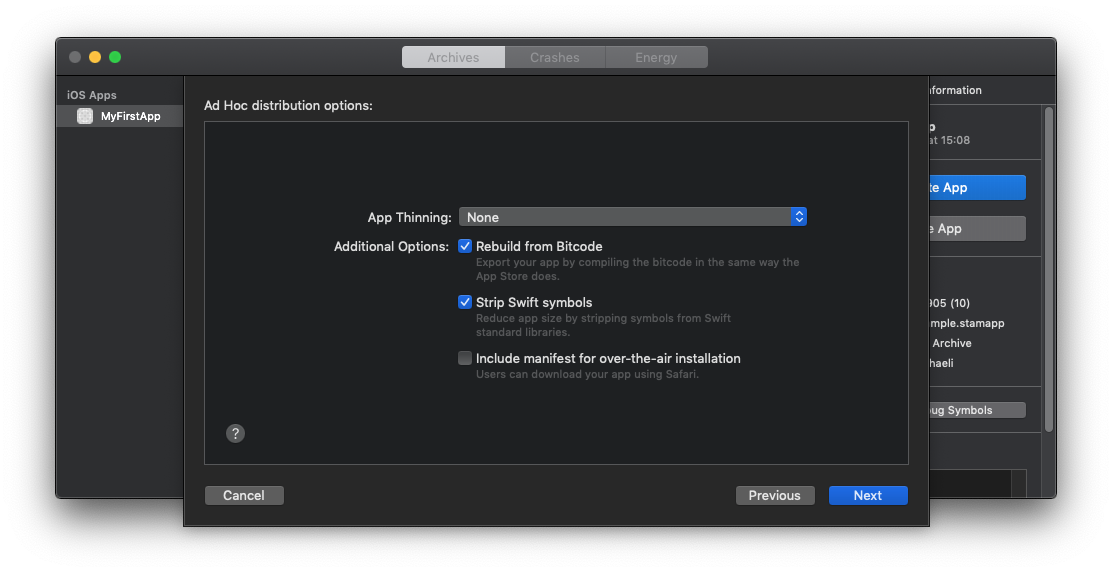


Click the Distribute App button. Select an export option, and click **Next**.

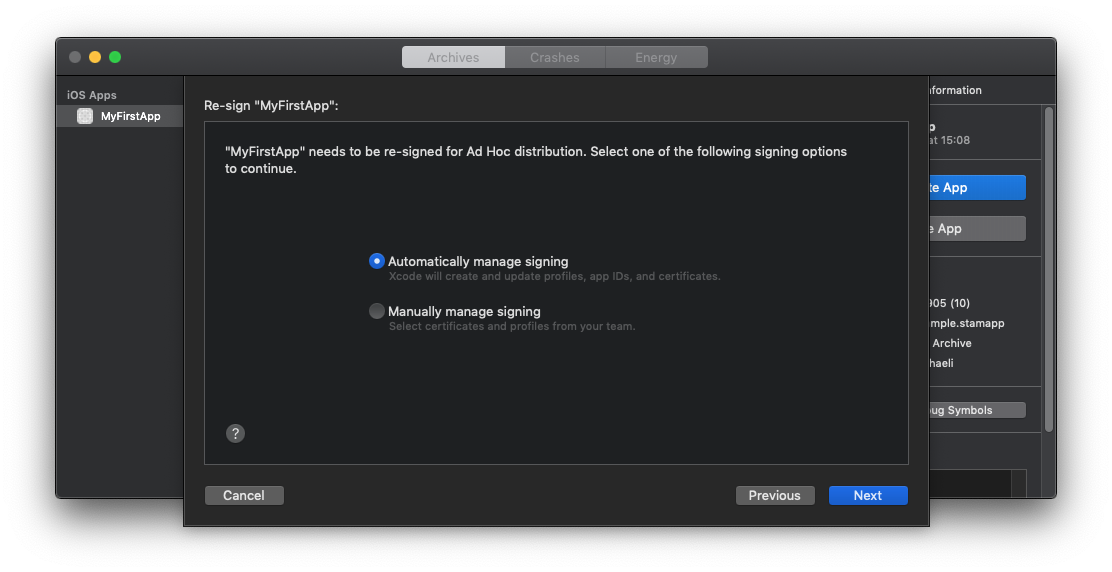
To distribute your app to users with designated devices, choose the Save for Ad Hoc Deployment. The app will be code signed with the distribution certificate.



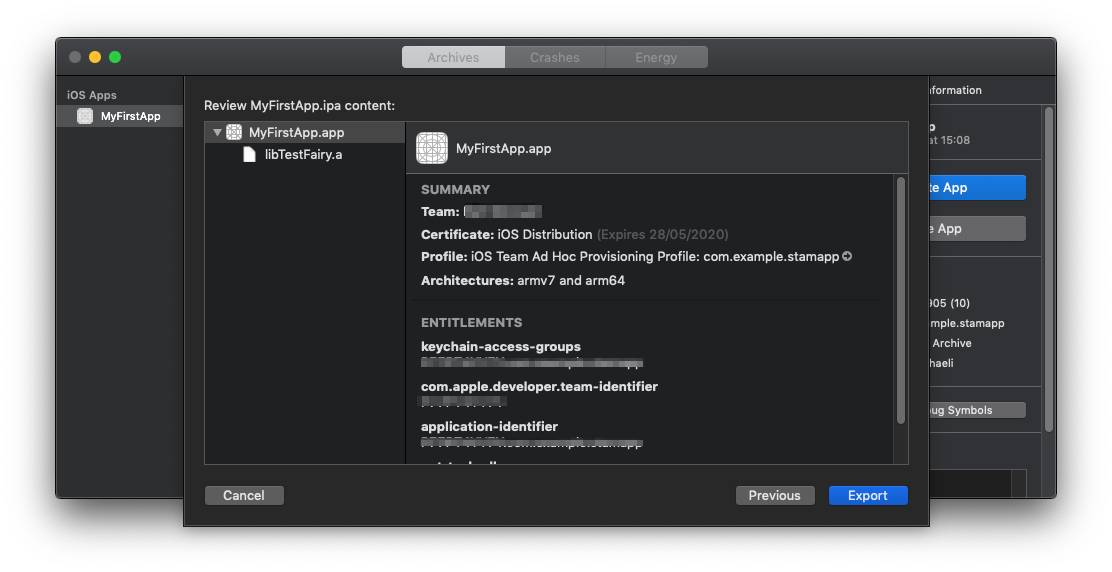
In the dialog that appears, choose a signing method and click **Next**.



In the distribution options screen, choose the options you prefer and click Next.



After you choose the signing options click **Next**.



In the dialog that appears, review the app, its entitlements, and the provisioning profile.  
Click **Export** and finally the Finder shows the exported files. Save it to your desired location.

If you have any doubts please go through below URLs:

1. <https://mobikul.com/how-to-generate-ipa-for-in-house-distribution/>
2. <https://docs.testfairy.com/iOS_SDK/Exporting_Ad_Hoc_IPA.html>
3. <https://developer.apple.com/documentation/xcode/distributing-your-app-to-registered-devices>