Scripture
App Builder

Installing and Building Apps on a Mac

Scripture App Builder:
Installing and Building Apps on a Mac

Last updated: 29 June 2022

The latest version is available at
http://software.sil.org/scriptureappbuilder/resources/

and on the Help menu of Scripture App Builder.

Contents

1. Introduction

This document provides information on how to install Scripture App Builder and build apps on an Apple macOS system. It is possible to build an Android app using SAB on Windows, Linux or Mac, but if you want to build an iOS app for the iPhone or iPad, you will need to build it using a Mac computer.

App Builder Platform	Build Android Apps	Build iOS Apps
Windows	✓	
Linux	✓	
macOS	✓	✓

Creating an Android app on a Mac is essentially the same process as it is for Windows or Linux. To create a corresponding iOS app, you will need to enter a few more configuration items.

The apps generated for iOS will run on an iPhone or iPad with iOS 12.2 or higher.

2. Installing Scripture App Builder

To install the Scripture App Builder program files:

1. Download the current Mac installer file (dmg) from the SAB website:

   http://software.sil.org/scriptureappbuilder/download/

2. Double click on the file within **Finder** to open the disk image that contains the Scripture App Builder application.

3. Copy the Scripture App Builder application to your **Application** folder. This can be done by dragging the Scripture App Builder icon from the disk image window to the shortcut of the Applications folder in the same window.

3. Installing Prerequisites for Android

If you want to build Android apps, you need to install the following components on your computer:

1. Java Development Kit (JDK)
2. Android Software Development Kit (SDK)

3.1. Java Development Kit (JDK)

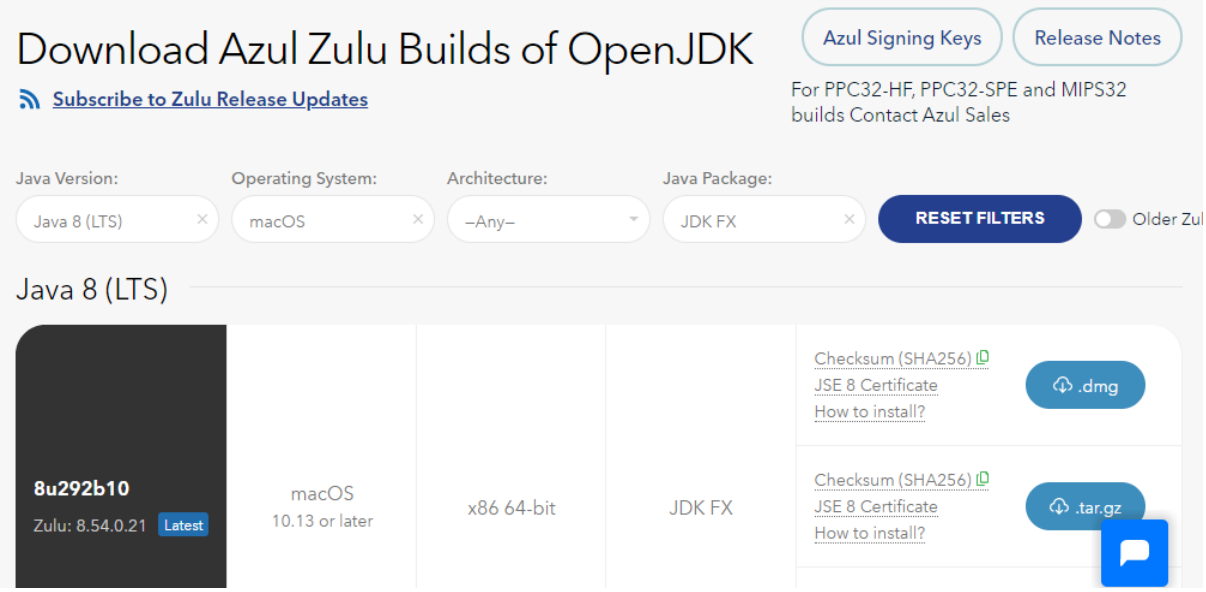
You will need version 8 of the Java Development Kit (JDK) to build apps. We recommend you use Zulu, which is a free distribution of OpenJDK from Azul.

1. Go to the **Download Zulu Builds of OpenJDK** website:

   https://www.azul.com/downloads/?version=java-8-lts&os=macos&package=jdk-fx

   There are many downloads on this page, but the above link will filter the ones you see (Java Version: Java 8 LTS; Operating System: MacOS; Java Package: Java FX).

2. Scroll down the page until you see the downloads under the heading **Download Azul Zulu Builds of OpenJDK**:



3. You have a choice between two different architectures: **x86 64-bit** and **ARM 64-bit**. You can find the processor type of your machine by clicking on the Apple symbol at the top left of your screen and selecting **About This Mac**. If you have one of the newer Macs with an M1 chip, choose ARM, otherwise you will need x86 (Intel).

   Once you have identified your device architecture, you have a choice between a **dmg** file, a **tar.gz** file and a **zip** file. **Download the .dmg file** since it comes with its own installer program.

   The file you download will have a filename something like this:

   zulu8.54.0.21-ca-fx-jdk8.0.292-macosx_x64.msi

4. **Double-click the file in Finder** and follow the instructions in the installation wizard to install it. By default, the installer will install the JDK to the following folder:

   /Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home

   Important: If you change the JDK install folder to something other than the default folder, you will need to remember the location of the folder so you can tell Scripture App Builder where to find the JDK.

3.2. Installing Android Software Development Kit (SDK)

The Android Software Development Kit (SDK) is needed for building Android apps. There are two ways of installing the Android SDK:

1. **Online: Download the Android SDK packages from the internet:**
   Use the Android SDK Installation wizard to download and install the command line tools and three additional packages. This method will require an internet connection.

   See 3.2.1 for more details.

2. **Offline: Copy the Android SDK files from someone else:**
   If you know someone who has already downloaded and installed the Android SDK, you can copy all the files from them.

   This method is especially useful in a training workshop where several people need to install the SDK but have limited internet bandwidth.

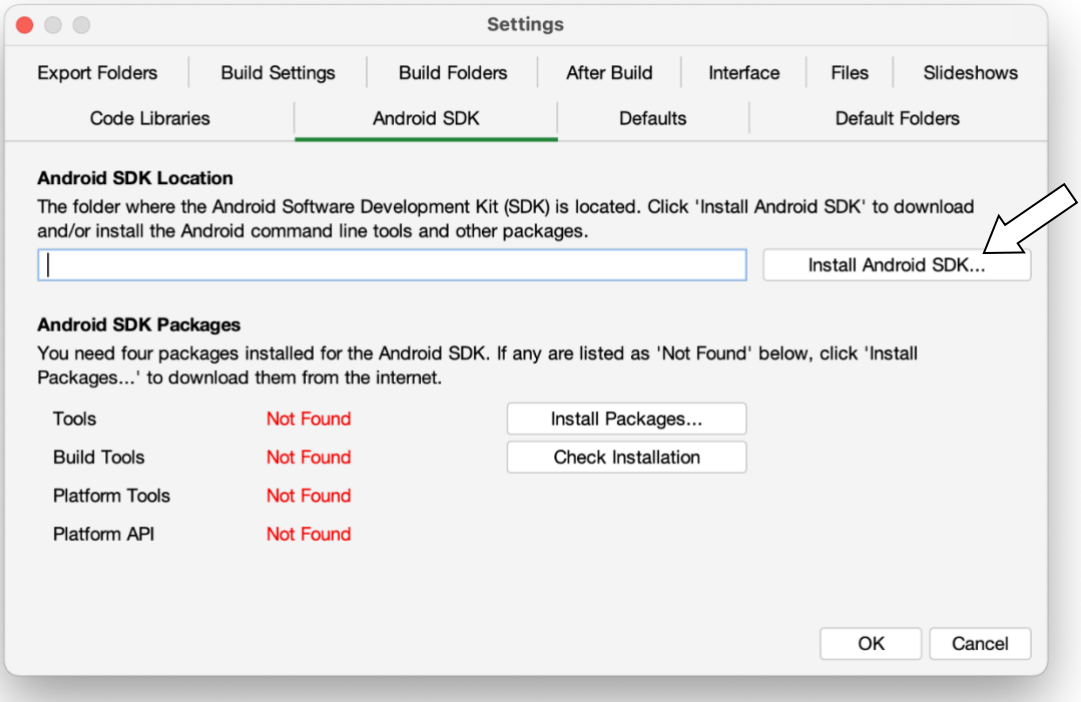
   See 3.2.2 for more details.

3.2.1. Downloading the Android SDK packages from the internet

To install the Android SDK from the internet:

1. Launch **Scripture App Builder**.
2. Select **Scripture App Builder** ➢ **Preferences** from the main menu.
3. Go to the **Android SDK** tab, which is the second tab.

4. Click the **Install Android SDK** button.



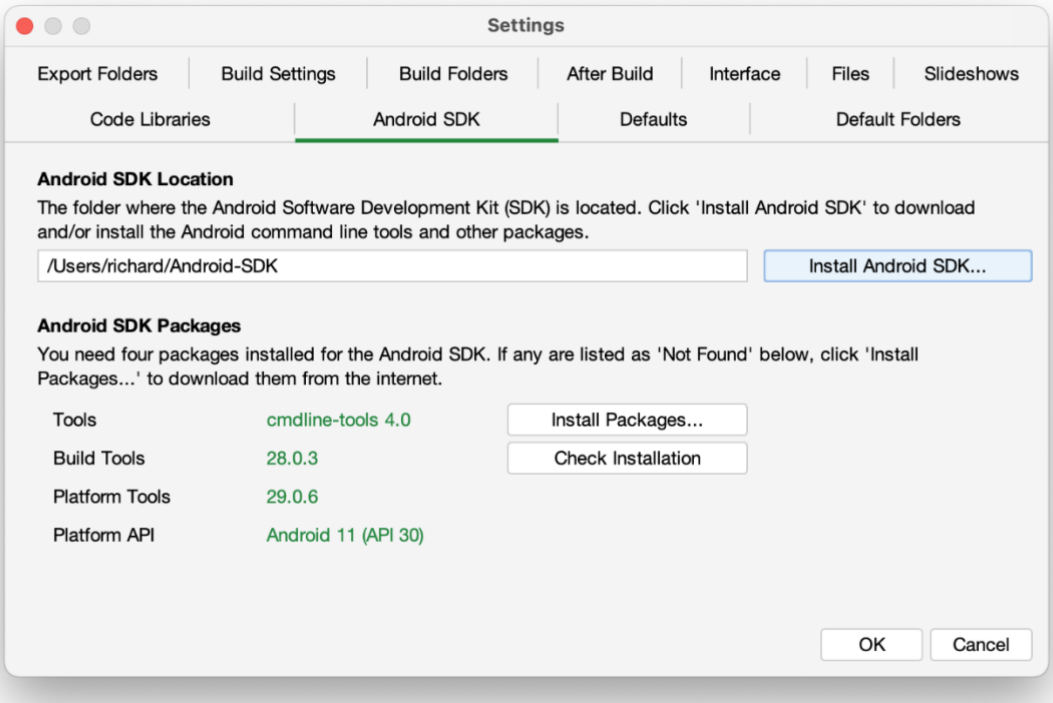
5. Follow the instructions on each page of the **Install Android SDK** wizard to download each of the Android SDK packages and install them.

   When you are asked to specify a target folder, a good place is:
   /Users/*your-name*/Android-SDK.

   Four packages will be downloaded and installed:

   - Command line tools,
   - Build Tools,
   - Platform Tools, and
   - Platform API.

   If the installation was successful, you will see the version numbers displayed in green.



If any of the Build Tools, Platform Tools or Platform API is listed as “Not Found” (displayed in red), click the **Install Packages** button to install them.

Click the **Check Installation** button to confirm that all the packages have been installed correctly.

You can skip section 3.2.2 and go straight to section 4.

3.2.2. Copying the Android SDK files from someone else

If you know someone who has already downloaded and installed the Android SDK and is successfully building apps with it, you can copy all of their Android SDK files to a folder on your computer.

You need to look for the top-level Android SDK folder, such as **/Users/*user-name*/Android-SDK**, and copy the whole folder and its contents to your computer. A location such as **/Users/*your-name*/Android-SDK** is good. If it makes it easier, you can zip the folders and then unzip them onto your computer.

Note that there is no setup program to run. Copying the files from one computer to another is sufficient.

Tip: A typical Android SDK folder can be quite large (over 1 GB, depending on which additional packages have been installed). To build an app, you do not actually need all of the Android SDK files. If you want to cut down the number of files, here is a list of the essential and optional folders:

Android SDK Folder	Required for building apps?
cmdline-tools (or tools)	Yes
build-tools	Yes (you only need the sub-folder for the latest version)
platforms	Yes (you only need android-30 for now)
platform-tools	Yes
add-ons	No
docs	No
emulator	No, unless you want to use an emulator
extras	No
licenses	Yes
sources	No
system-images	No, unless you want to use an emulator
temp	No

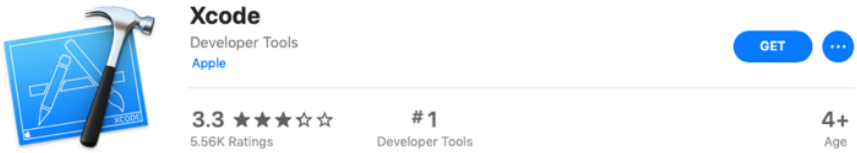
4. Installing Prerequisites for iOS

If you want to build iOS apps and upload them to the Apple App Store, you need to install the following components:

1. **Xcode**
2. **Transporter**

4.1. Install Xcode

Xcode is required to build iOS Apps. To install Xcode, simply search for Xcode in the Mac App Store and install it. Open Xcode at least once to agree to the licensing restrictions and install components.



4.2. Verify Xcode Installation

To verify that you have successfully installed Xcode and that it will be correctly used by Scripture App Builder, please open Terminal and run the following command to print the path to the active developer directory: `xcode-select -p`

```
$ xcode-select -p
/Applications/Xcode.app/Contents/Developer
```

If you have had Xcode Command-line Tools installed previously, it might still be pointing to that installation directory and Scripture App Builder will not work correctly.

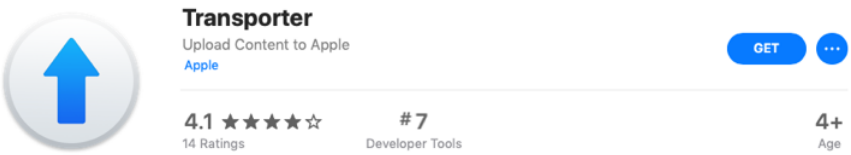
```
$ xcode-select -p
/Library/Developer/CommandLineTools
```

To correct this situation, run the following command to set the active developer directory.

```
sudo xcode-select -s /Applications/Xcode.app/Contents/Developer
```

4.3. Install Transporter

Transporter is used to upload iOS apps to App Store Connect. To install Transporter, simply search for Transporter in the Mac App Store and install it.



5. Installing Prerequisites for Progressive Web App

If you want to build a Progressive Web App (PWA) that can be downloaded from the browser on any platform instead of going through an app store, you need to install the following components:

1. **Node.js**
2. **Workbox-cli**

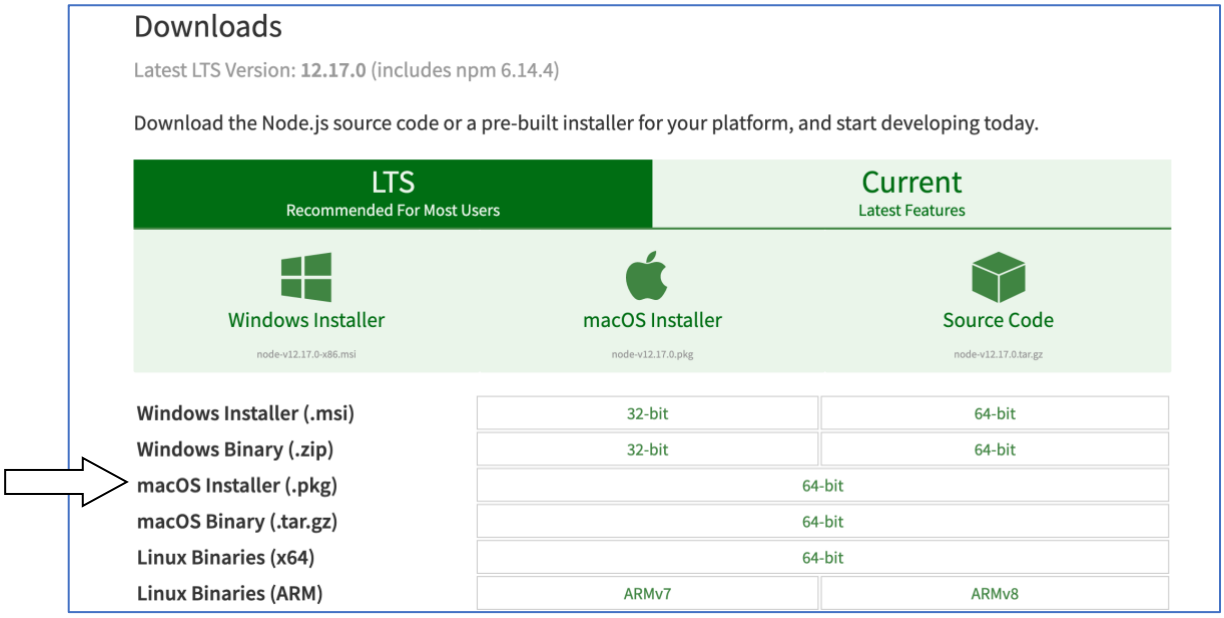
For more information on what PWAs are, why you might use them, and how to export the files needed, see Scripture App Builder Documentation 2: Building Apps.

5.1. Install Node.js

Go to the Node.js Downloads page:

https://nodejs.org/en/download/

There are many download files on this page. You are looking for the file that corresponds to your computer's operating system type. It is easiest to download the .pkg file (installer package file) rather than the tar file.



When the file has downloaded, run it to install Node.js on your computer.

5.2. Install Workbox-cli

After installing Node.js, open Terminal. Paste in this command and press enter:

```
sudo npm install workbox-cli -global
```

Once this command finishes running, you have the prerequisite tools for exporting PWA files. Close SAB if it is open and restart it.

6. Installing Aeneas

Aeneas is the audio-text synchronization tool that may be run from within Scripture App Builder to create timing files for phrase-by-phrase highlighting. If the apps you are building do not include audio or if the timing files are already available, then there is no need to install it.

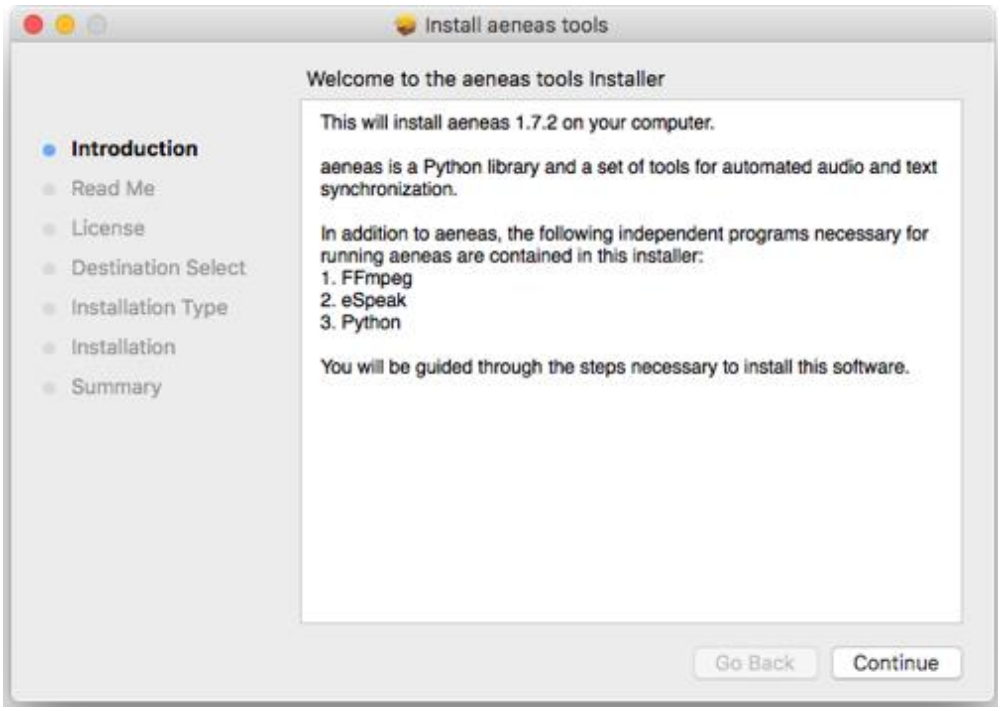
To install **aeneas**:

1. Download the **aeneas tools** for Mac file (dmg) from the SAB website from the section labeled **Audio Synchronization Tools**:

   http://software.sil.org/scriptureappbuilder/download/

2. Double click on the file within Finder to open the disk image that contains the aeneas-mac-setup install package. Control-click on the install package and select **Open**. You will get a warning that this package is from an unidentified developer. Press **Open**.

3. The introduction screen will display. Press **Continue**.

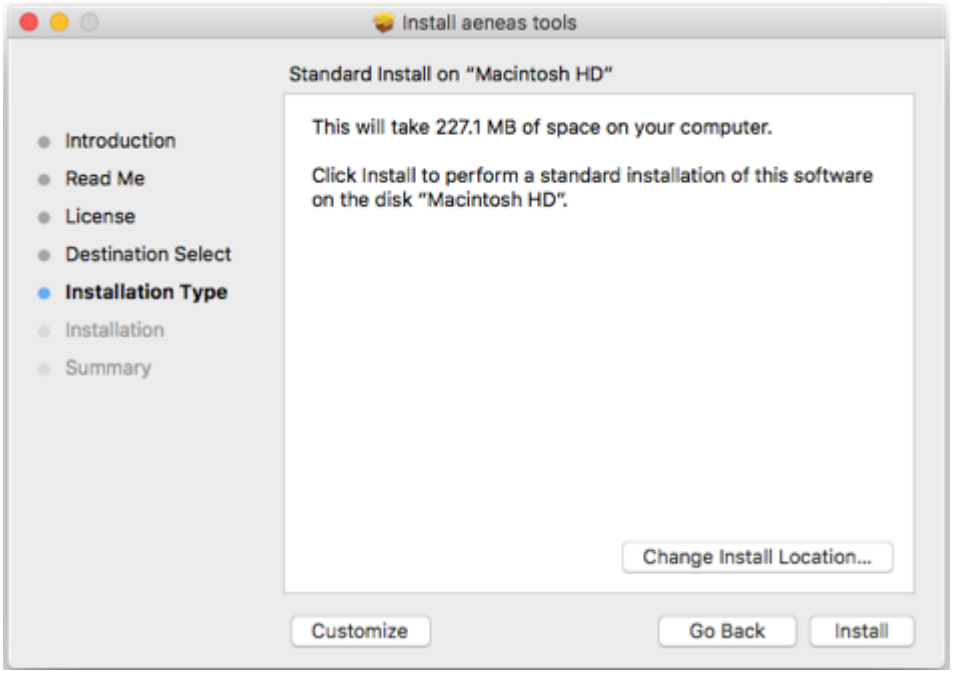


4. A “Read Me” screen will display next followed by a “License” screen. Press **Continue** on both screens.

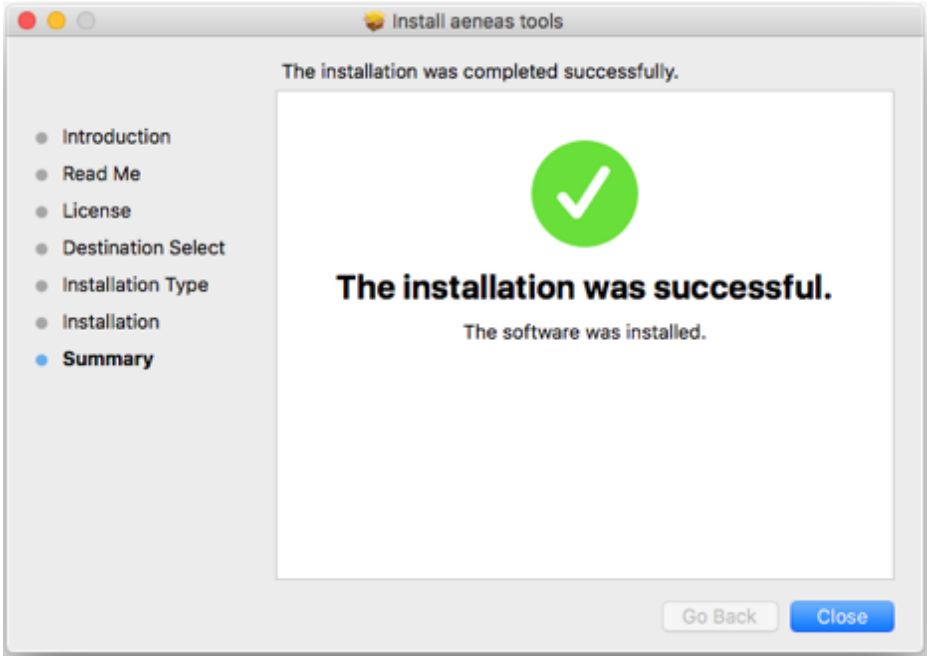
5. Pressing the **Continue** button on the License screen will bring up a screen asking if you agree to the terms of the license. Press **Agree**.

6. The “Destination Select” screen selects the default drive. Press **Continue**.

7. The “Installation Type” screen displays next for a standard install. Press **Install**.



8. The installer will prompt for credentials to install the software. Enter a username and password with permissions and press **Install Software**.
9. At this point the installation will start and will show progress screens until it completes. A terminal window will popup briefly to test the installation. When the application completes successfully, the original screen will show:



10. Press **Close**.

Aeneas is installed in /usr/local/lib/python2.7/site-packages.

7. Testing App in iOS Simulator

When you want to test your app, you can either use a device or a simulator. To test with a device, you will need a signing certificate and provisioning profile (see next section). To test with a simulator, you will need to download Xcode, the integrated development environment used to build and test iOS and Mac apps. Xcode is available for free in the Mac App Store and is quite large (5.46GB). SAB requires Xcode 9 or greater (which requires macOS Sierra). To install:

1. Go to https://itunes.apple.com/us/app/xcode/id497799835?mt=12 or search for "Xcode" in the App Store app on macOS.
2. Install from the App Store.
3. Start Xcode at least once to complete the installation.

7.1. Run the iOS Simulator

Once you have a project configured and ready to test, click on the **Run iOS App in Simulator** on the toolbar.

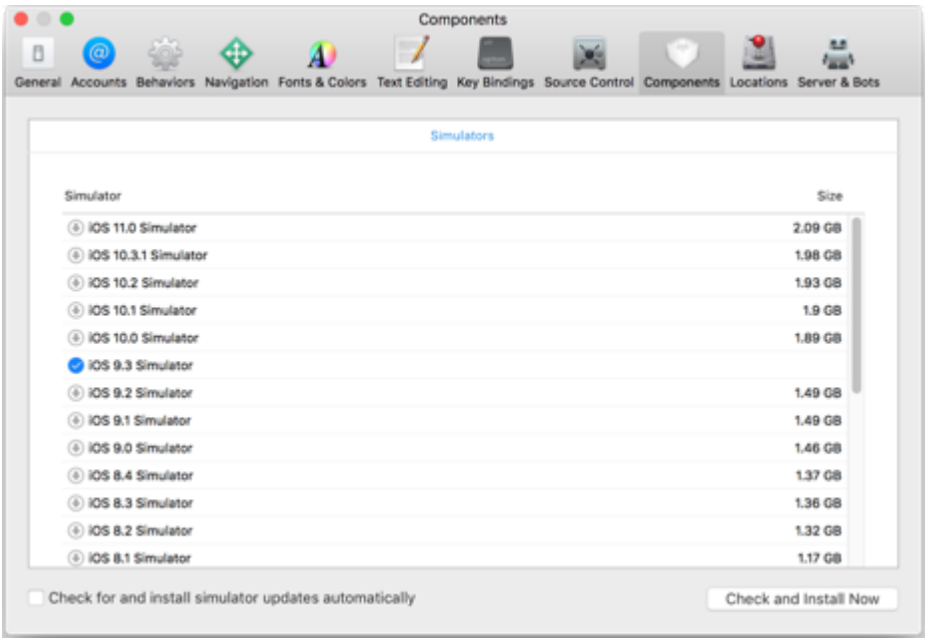
Select the simulator you would like to run on and click **Start**. It takes a little bit of time for the simulator to start. If you want to switch simulators, select a different Simulator from the **Simulator Type** combo box and click **Start** again. It will close the previous Simulator and start the new own.

Select the project you want to test (it defaults to the selected project in the Apps list) and click on **Build**. This will build the app for the Simulator in a separate terminal window. When the build is complete, you can click on **Launch** to run the app in the Simulator.

You may close the dialog and make changes to your project. When you restart the Run iOS Simulator dialog again, you will need to Build and Launch again for the changes to be included.

7.2. Installing Additional Simulators

You may install simulators for previous versions of iOS by launching Xcode and viewing the preferences dialog and selecting the Components tab.



7.3. Manually Installing Apps into the Simulator

If there is a problem with launching the simulator from the **Run iOS Simulator** dialog, you can manually install the app by dragging the built app from the Simulator output folder (found in ~/App Builder/Scripture Apps/Sim Output) onto the Simulator.

To start the Simulator, launch **Xcode** and from the **Xcode** menu select **Open Developer Tool ➢ Simulator**. You will still need to rebuild the app from the **Run iOS Simulator** dialog.

8. Creating iOS Certificates and Provisioning Profiles

8.1. Enroll in the Apple Developer Program

To build iOS apps and distribute them through the Apple App Store, you will need to be enrolled in the Apple Developer Program. You can do this as either (i) an individual, or (ii) an organisation. The cost is USD $99 per year.

To enroll:

1. Go to https://developer.apple.com/programs/enroll/
2. Press the **Start Your Enrollment** button to start.

8.2. Create Signing Certificate

When you create an iOS app, it needs to be signed with a certificate.

To create a certificate:

1. Go to the Apple Developer website and log in to your account if you are not already.
2. Select **Certificates, Identifiers & Profiles**.
3. Click the ⊕ button to the right of the **Certificates** header.
4. On the **Create a New Certificates** page, select **Apple Distribution**. Then press the **Continue** button.
5. Upload a Certificate Signing Request. Press the **Learn more** link for instructions on how to use Keychain Access on your Mac computer to complete this task. Then press the **Continue** button.
6. On the **Download Your Certificate** page, press the **Download** button to download the certificate (distribution.cer) to your Mac.
7. Open the Keychain Access application. Choose the **login** item from the Keychains section on the left. Choose the **File** ➢ **Import Items…** menu item and browse to the Downloads folder and select the certificate (distribution.cer).

Now that the certificate is installed in the Keychain, you will be able to access it from within Scripture App Builder.

Note: To work with certificates, you will need the Apple Worldwide Developer Relations (WWDR) Certification Authority. It should have been installed with Xcode. When viewing your certificate in the Keychain Access application, if there is an error stating the certificate is not trusted, then install the certification authority.

To get the certification authority:

1. Download the Apple WWDR Certification Authority from:
   https://developer.apple.com/certificationauthority/AppleWWDRCA.cer

   For signing certificates created after January 28, 2021:
   https://www.apple.com/certificateauthority/AppleWWDRCAG3.cer
2. Open the Keychain Access application. Choose the **System** item from the Keychains section on the left. Choose the **File** ➢ **Import Items…** menu item and browse to the Downloads folder and select the certification authority (AppleWWDRCA.cer). You will be prompted for a username and password that has admin privileges in order to modify the **System** Keychain.

3. Choose the **File** ➢ **Lock Keychain "System"** menu item.

8.3. Create Provisioning Profile

Distribution provisioning profiles are used to for two primary purposes:

- AdHoc - to install your app on a limited number of registered devices for testing.
- App Store - to submit your app to the App Store.

Creating a provisioning profile

To create a new AdHoc provisioning profile, you will need an App ID and at least one registered device. To create a new App Store provisioning profile, you will need just the App ID.

To create an App ID:

1. Go to the Apple Developer website and log in to your account if you are not already.
2. Select **Certificates, Identifiers & Profiles**.
3. Select **Identifiers** on the left of the page.
4. Click the ⊕ button to the right of the **Identifiers** header.
5. Select **App IDs** and click the **Continue** button.
6. Select **App** and clock the **Continue** button
7. Enter a **Description** of your choice.
   It can be the **App Name** from the **App** ➢ **Package** page of Scripture App Builder.
8. For Bundle ID, choose **Explicit** and enter your app package name from the **App** ➢ **Package** page of Scripture App Builder.
9. Of the App Capabilities, there are only two that may need to be checked. Check the Push Notifications capability if the app uses Firebase Cloud Messaging, Verse of the Day, or Daily Reminders. Check the Associated Domain capability if using Deep Linking.
10. Click the **Continue** button.

To register a device (i.e. a specific iPhone or iPad for testing):

1. Select **Devices** on the left of the page.
2. Click the ⊕ button to the right of the **Devices** header.
3. In the section **Register a Device**, enter a name of the device (such as "John Smith's iPhone") and its UDID (unique device identifier, a sequence of 40 letters and numbers). Press **Continue**.

4. On the next page, check that the device information is displayed correctly and click **Register** to confirm.

   Note that you can register up to 100 devices of each type (e.g. up to 100 iPhones, 100 iPads) per year of your Apple Developer Program membership. You can remove devices that you no longer need at the beginning of the next membership year.

To create a provisioning profile:

1. Select **Profiles** on the left of the page.
2. Click the  button to the right of the **Profiles** header.
3. On the **Register a New Provisioning Profile** page, under **Distribution**, select **Ad Hoc** (for testing) or **App Store** (for submission to the App Store). Click **Continue** to move to the next page.
4. On the 'Select an App ID' page, select the App ID from the list of App IDs you have already defined. Click the **Continue** button.
5. On the 'Select Certificates' page, select the certificate (Distribution) to include in the profile. Click the **Continue** button.
6. If creating an AdHoc provisioning profile:
   On the 'Select Devices' page, check the device(s) that you want to be able to install and run the app. Click the **Continue** button.
7. On the 'Review, Name, and Generate' page, give the profile a name of your choice. It is helpful to include "App Store" or "AdHoc" in the name. Click the **Continue** button.
8. On the 'Download and Install' page, click **Download** to save your new .mobileprovision file to your computer.

   This is the Provisioning Profile file that you will select in Scripture App Builder.

Download existing mobile provision files

If other members of your team have already created provisioning profiles, they can be downloaded from the App Developer website by selecting the profile to be downloaded and pressing the **Download** button.

9. Building an iOS App

The first step in building an iOS app is to create a new app project following the instructions in the SAB **Building Apps** document. Then follow the instructions below.

9.1. Application builds available for iOS

For iOS distribution, three different app types are available.

9.1.1. Dedicated App

This is the standard traditional app and is the default. The content that is defined for this project will be used to generate an iPA file that may be delivered to the App Store for publication. A dedicated app could be viewed as a container app that only runs one preselected asset package that is included in the app at build time.

9.1.2. Container App

App Builder users have encountered issues with the Apple App Store reviewers if too many apps generated by the App Builder are published to the same account. Since the actual application differs only in the content provided, Apple rejects the app as being spam. To avoid these issues, a container app can be generated instead of the standard dedicated app.

The container app does not contain content of its own. It does not have book collections, audio or any of the other normal content. Instead, on its initial startup, it accesses a web site that is maintained by the app developer. The web site provides specially formatted links to multiple asset packages. The app user selects one of these asset packages. The container app downloads it and runs using the downloaded content. From this point, it resembles the dedicated app, running this set of content without further downloads being required.

The user of a container app does have the option of resetting the content at any time. If the user elects to reset the app content, the app asset package currently in use is deleted from his device. The web site accessed at the initial startup is displayed, and the user can once again select the asset package to be run from the available list. The selected package is then downloaded and run in place of the discarded one.

Building a container app will generate an IPA file similar to the dedicated app with the exception that no content is included with the app beyond some configuration information and a reference to the website used by the app. The IPA file is submitted to the Apple App Store in the same manner as the dedicated app.

While most of the settings associated with the app need to be set in the asset package projects, there are a few settings that must be set at the container app level. The first of these are the Firebase related features. Firebase tracks at the app level, and the settings for the *Firebase* tab and the *Security* tab will apply for all app asset packages associated with this container app. The GoogleService-Info.plist file that is used by the iOS Firebase configuration should be associated with the Package Name for the container app set on the package tab. Firebase will track information associated with the container app, not with the individual asset package projects. A project field is included in events generated by asset packages and sent to Firebase and can be used to identify the events that originated with a particular package that is being distributed.

Several other features follow the settings in the asset package project definition but must be accommodated by the container app that loads them. If any of the asset packages that can be loaded by this container app application use *Verse of the Day* or *Daily Reminder* features on the *Notifications* tab, these options must be enabled in the container app as well as in the asset package that wants to use the feature. The same applies for the *Verse on Image* feature. While the backgrounds and watermarks that are available are set in the asset package project, if any asset package that can be loaded by this container app application uses the *Verse on Image* feature, then the *Include the "Verse on Image" editor in the app* checkbox on the *Verse on Image* tab must be checked here, as well as in the asset package that wants to use the feature. The *IPA* tab also includes a checkbox that must be selected if any of the asset packages include audio files.

On the *Images* tab, the *iOS Icon* and the *iOS Splash Screen* (optional) should be configured for the container app as these will be the images used. The images set in asset packages for these two items will not be used.

9.1.3. Asset Package

Asset packages are used in conjunction with Container Apps. The configuration of an asset package project should be the same as it is for a dedicated app, setting the book collections, audio and feature required for that app. The only exceptions are those mentioned in the section above on Container Apps. However, where the build of Dedicated Apps or Container Apps results in an IPA file to submit to the Apple App Store, the build of an Asset Package project results in a zip file, saved to the IPA output folder, that is intended to be saved to a location where it can be referenced by the web site supporting the Container App. A user running who has downloaded the Container App from the App Store will select this package from the list presented by the Container App Web Site, at which point it will be downloaded to the device, decompressed and run as the project for the app. All of the book collection, audio, font, images, features and configuration information that is usually embedded within the dedicated app is saved in this zip file so that the project can be run as it would if it were configured as a dedicated app.

Dedicated Apps and Container Apps can only be generated on an Apple Mac device. Asset Packages do not require a Mac. Therefore, App Builder can build a project configured as an iOS Asset package on a Mac, a Linux device, or a Windows device. On a Mac Device, selecting *Build iOS App* for a project configured as an Asset Package will generate the zip file in the IPA Output Folder. For testing purposes, so that the app developer can test the device outside of the container app, selecting *Run iOS App in Simulator* will cause an IPA file to be generated in the Simulator Output Folder that will run using the project configuration. The app developer can load the IPA file on an iOS device simulator and test the project before generating the zip file and adding it to their Container App web site.

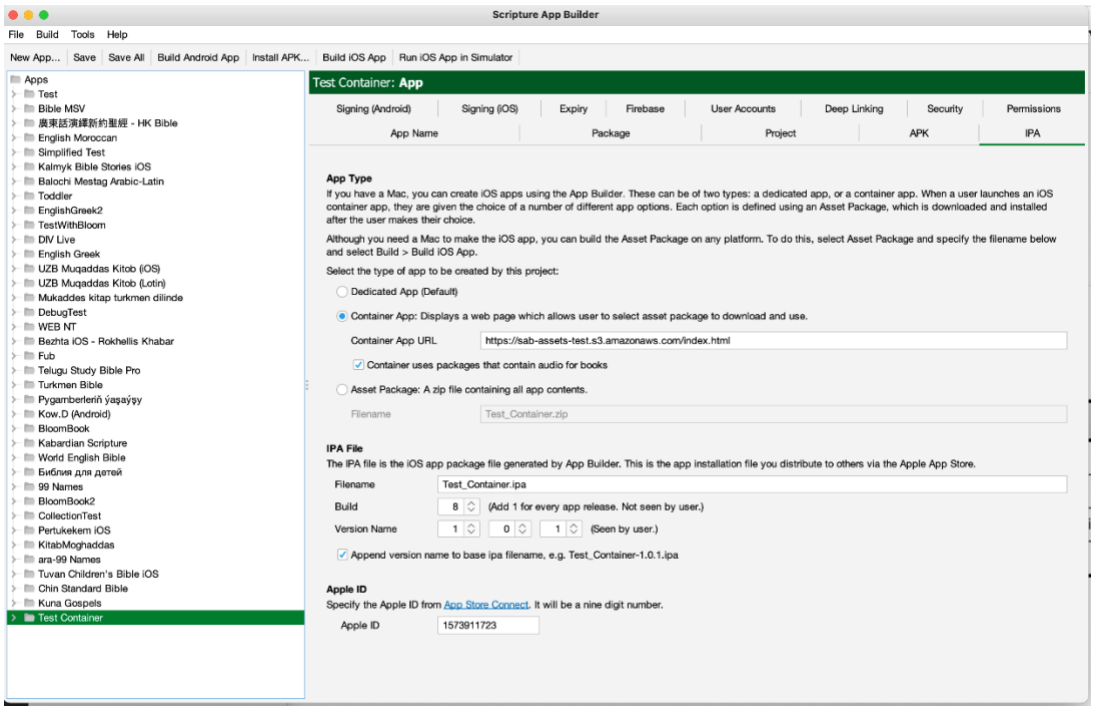
9.1.4. Container App Website

While not generated by App Builder, a Container App project must have a web site where the asset packages are located and a web page is defined by the user to allow selection of the asset package. The Container App displays the web page referenced by the URL that is part of its configuration. Anything other than links that reference *asset://yourwebsite/yourassetpackage.zip* or *https://[something].zip* or *http://[something].zip* are passed through the browser and are displayed to the user. When the user selects a link that starts with *asset://* or a normal URL that ends in *.zip*, that is interpreted as the selection of an asset package. If the URL starts with *asset://*, the *asset://* in the link is replaced with *https://* to generate the destination URL. The file referenced is downloaded, unzipped, and used to initialize the Container App. A very simple example of a Container App Website web page would be:

```
<html>
<head>
<style>
.container {
  display: flex;
  justify-content: center;
  background-color: powderblue;
  font-size: 40px;
}
.center {
  width: 800px;
}
</style>
</head>
<body>
<p id="top" class="container">Container App Test</p>

<div class="container">
<ul> Select a translation
<li><a href="asset://sab-assets-test.s3.amazonaws.com/EnglishGreek2.zip">English
Greek</a></li>
<li><a href="https://Test.zip">Test</a></li>
</ul>
</div>
</body>
</html>
```

9.2. IPA Tab



Click on the **IPA** tab.

This allows you to set the name of the ipa file to be generated as well as the build and version information.

The ***App Type*** information identifies the type of application being built, as defined in the previous section. Select either *Dedicated App, Container App,* or *Asset Package.*

If the *Container App* option is selected, the ***Container App URL*** needs to be entered. The URL entered here is the web page that will be displayed to the user by the container app to allow them to select the asset package to be used. The checkbox underneath it must be selected if any asset package used by the Container App will contain audio for books.

If the *Asset Package* option is selected, the ***Filename*** field under the *Asset Package* option is enabled. This field defines the base name, without path, that will be assigned to the Asset Package zip file. The file will be created in the IPA Output folder when the user selects *Build iOS App* (or on a Linux or Windows system, selects *Build iOS App Asset Package*.

The remaining fields, under the ***IPA File*** and ***Apple Id*** headers are only enabled for a *Dedicated App* or *Container App.*

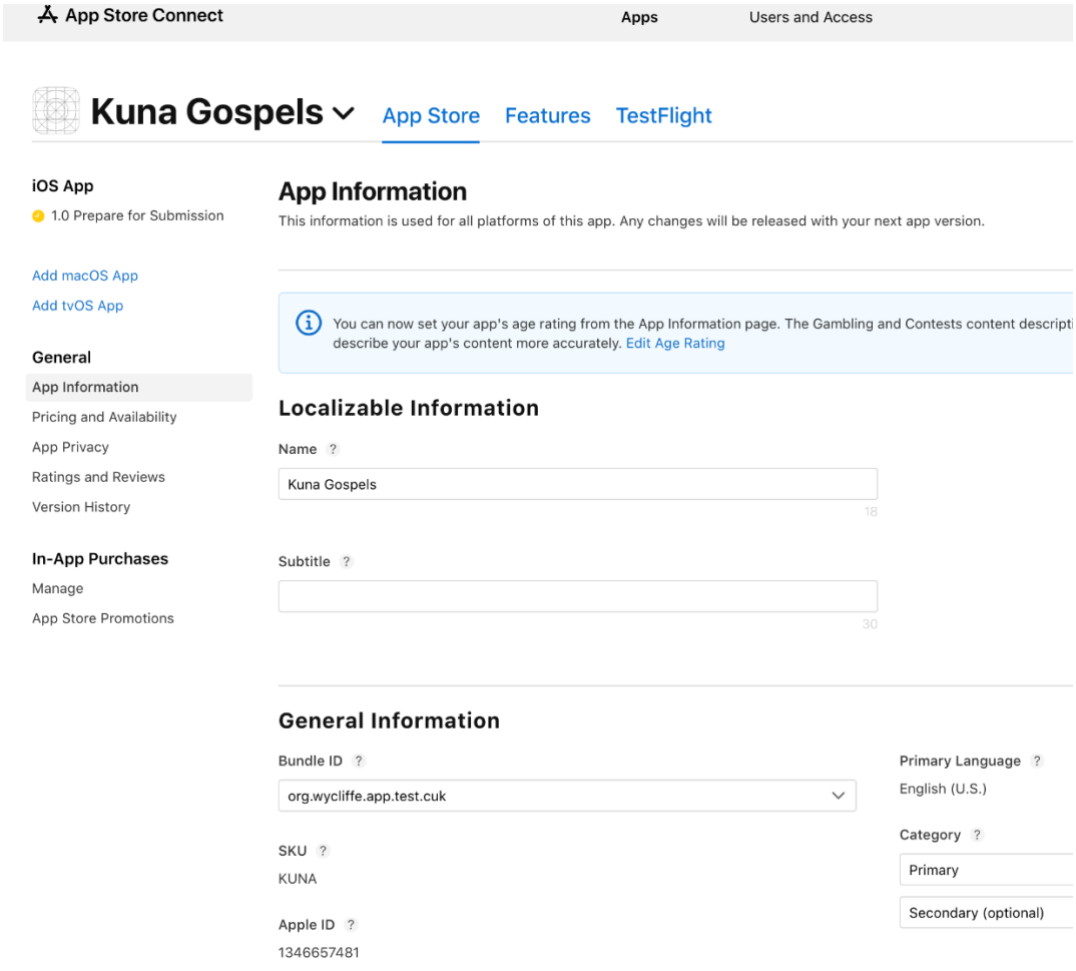
The ***Filename*** field on the screen for the ***IPA File*** section specifies the base name of the ipa file to be generated. If the checkbox at the bottom of the screen for *Append*

version name to ipa filename is checked, then the version indicated by the *Version Name* fields is added to the base filename.

The ***Build*** field referenced as *Build* is also called *Bundle Version String, Bundle Version* or *CFBundleVersion* within Xcode and ITunes Connect. It represents the build number. The *Build* field expects an integer value and should be incremented with each file that is submitted to the ITunes Connect for release or testing.

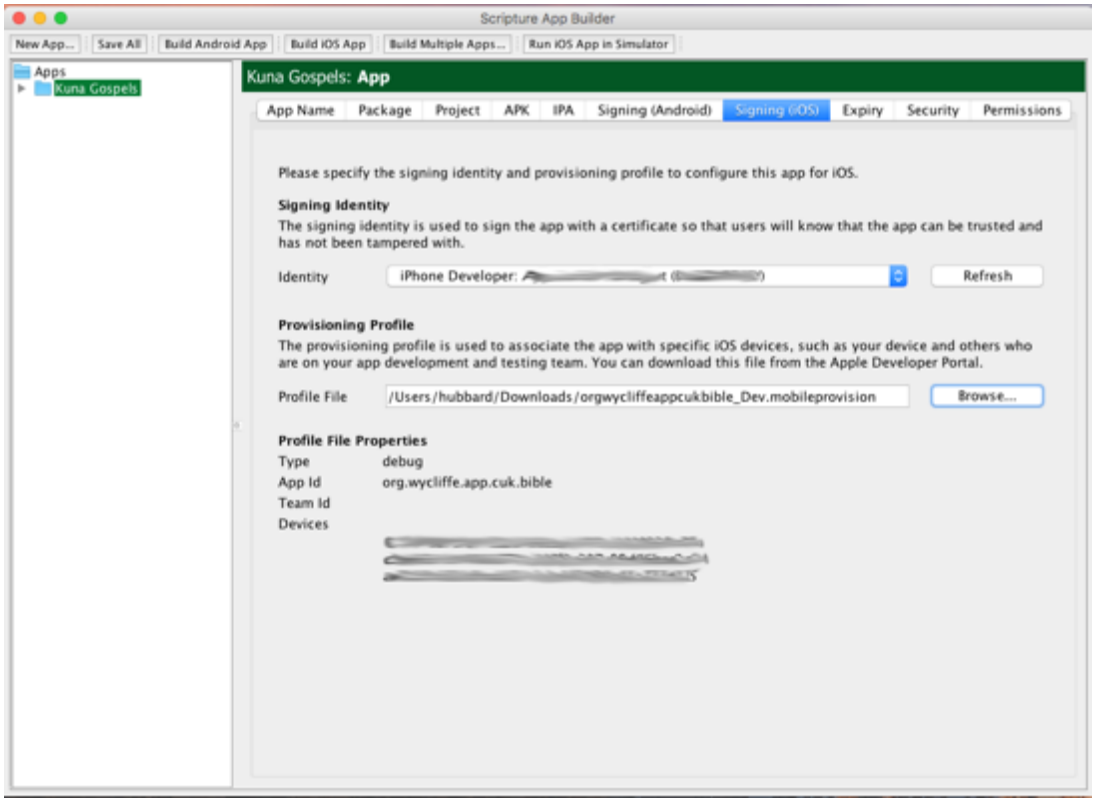
The ***Version Name*** field is referenced as *Version, Bundle Short Version String, Bundle versions string, short* and *CFBundleShortVersionString* within Xcode and ITunes Connect. The field is created as a concatenation of the values of the three fields separated by a period. If the final field has a value of 0, then the version string is created from just the first two values. So for values of 1, 2 and 0, the resulting string is "1.2". For values of 1, 2 and 3, the resulting version string is "1.2.3".

The **Apple ID** field is the id assigned by App Store Connect to the application in the app store. It can be obtained from **Apple ID** filed on the App Information tab in the App Store Connect entry for the app as shown below



9.3. Signing (iOS) Tab

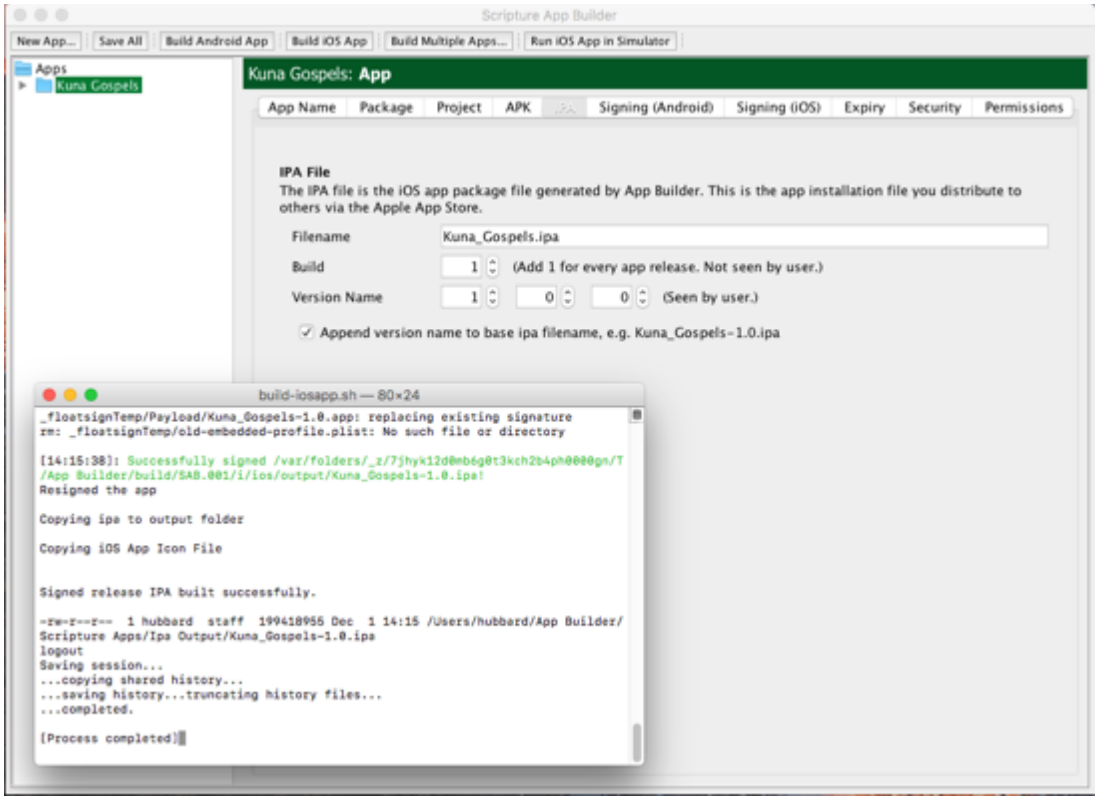
1. Select the **Signing (iOS)** tab to open the iOS signing options for the app.



2. Select the **Signing Identity** from the drop down list of signing certificates which have been downloaded and installed to this system in the earlier steps.

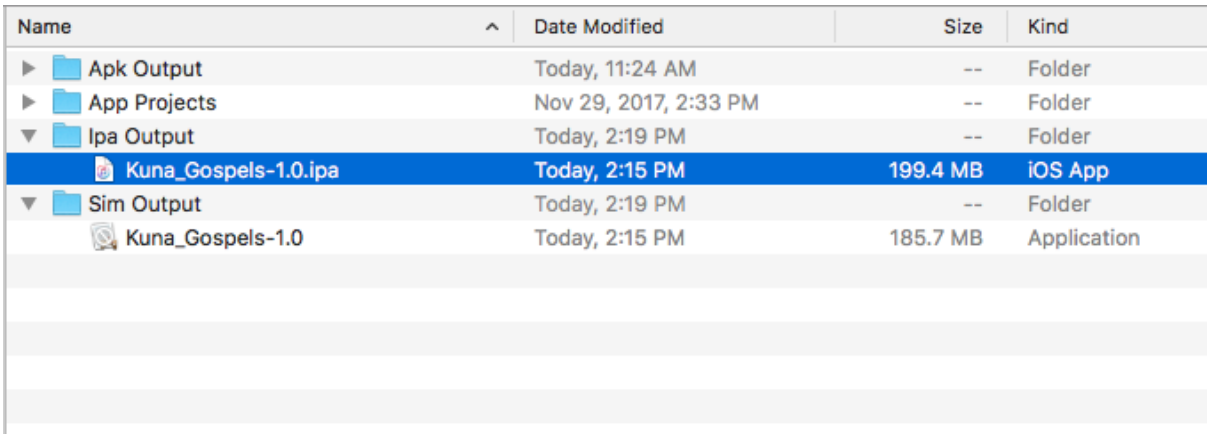
3. For the **Provisioning Profile** entry, enter or browse to the mobile provisioning file associated with the app that was downloaded in the earlier steps.

4. Click on the **Build iOS App** button at the top of the screen. A terminal window should open. The build script for the iOS App should run within that terminal window.



5. Examine the terminal window once the shell script has been completed. The message “Signed release IPA built successfully” should appear in the window if the app has been built successfully. (Note that occasionally the terminal window will appear behind the Scripture App Builder and that you have to select the terminal to review the results).

6. The results of the build are an IPA file and an app that can be run in the simulator. They can be found in ~/App Builder/Scripture Apps/Ipa Output/ and ~/App Builder/Scripture Apps/Sim Output/.



10. Testing an iOS App

After building the iOS IPA file, you will want to install it and test it on one or more devices before you submit it for publication to the Apple App Store. This manual describes two ways of doing this:

1. Use **Xcode** to install the IPA file to an iPhone or iPad that is connected to your computer.

   This method is recommended if you have your test devices with you. It does not involve uploading and downloading the IPA to and from the internet.

2. Use **DeployGate** to upload the IPA file to the internet and share it with limited number of devices to download, install and test.

   This method is recommended if you have good internet access and/or you have a team of testers who are elsewhere.

11. Using Xcode to Test an iOS App

To test your iOS IPA file using **Xcode**, do the following:

1. Launch **Xcode** and select **Window ➢ Devices and Simulators**.
2. Connect an iPhone or iPad to the Mac using a cable and unlock the device.
3. On the Mac, iTunes may launch and show a dialog asking "Do you want to allow this computer to access information…" Click on the **Cancel** button.
   - Note: This feature can be turned off in iTunes. Select **iTunes ➢ Preferences…**, select the **Devices** tab, and click on the **Prevent iPods, iPhone, and iPad from syncing automatically** checkbox.
4. On the device, it may prompt to **Trust This Computer**. Tap on the **Trust** button. This may require to enter the Passcode to trust this computer. The device will show up in the **Xcode Devices** window.
5. The first time the device is connected to the Mac, Xcode will take some time to Prepare debugger support. This may take some time. Please wait.
6. In the **Xcode Devices** window, there will be a section named **INSTALLED APPS**. Click on the **+** button.
7. Find the IPA file to add. Click **Open** after you have selected it.
8. Click the **Install** button next to the name of the app.
9. Wait until the Mac installs the app to your device.

After the install is complete, you will see the app icon on your device and you can test it.

12. Using DeployGate to Test an iOS App

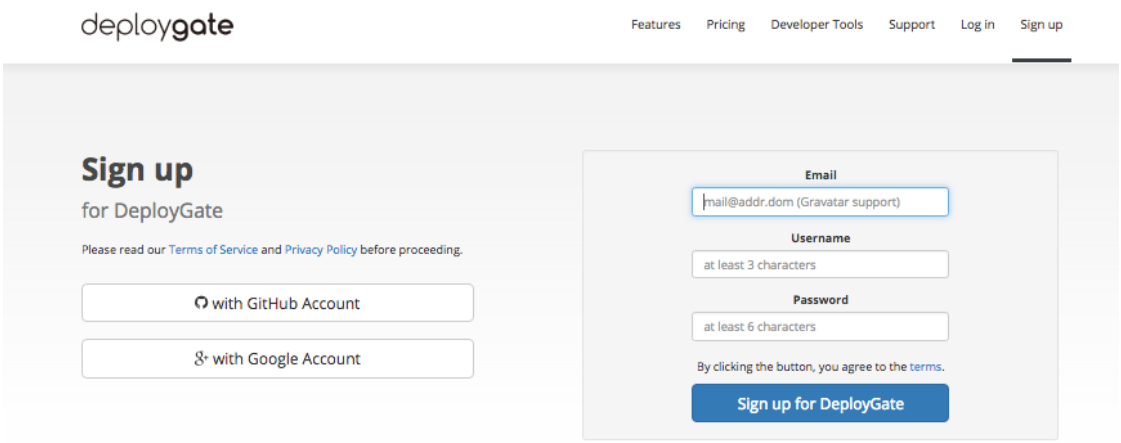
DeployGate enables you to test your app and share it with a limited number of users to test. You upload the IPA file for iOS apps or the APK for Android apps and then download them to your phone or tablet device. You can also invite testers to install your app and help with the testing.

A DeployGate app is installed on the testing device. It will show all of the apps that you have uploaded to DeployGate and allows them to be installed on the device.

12.1. Creating a DeployGate Account

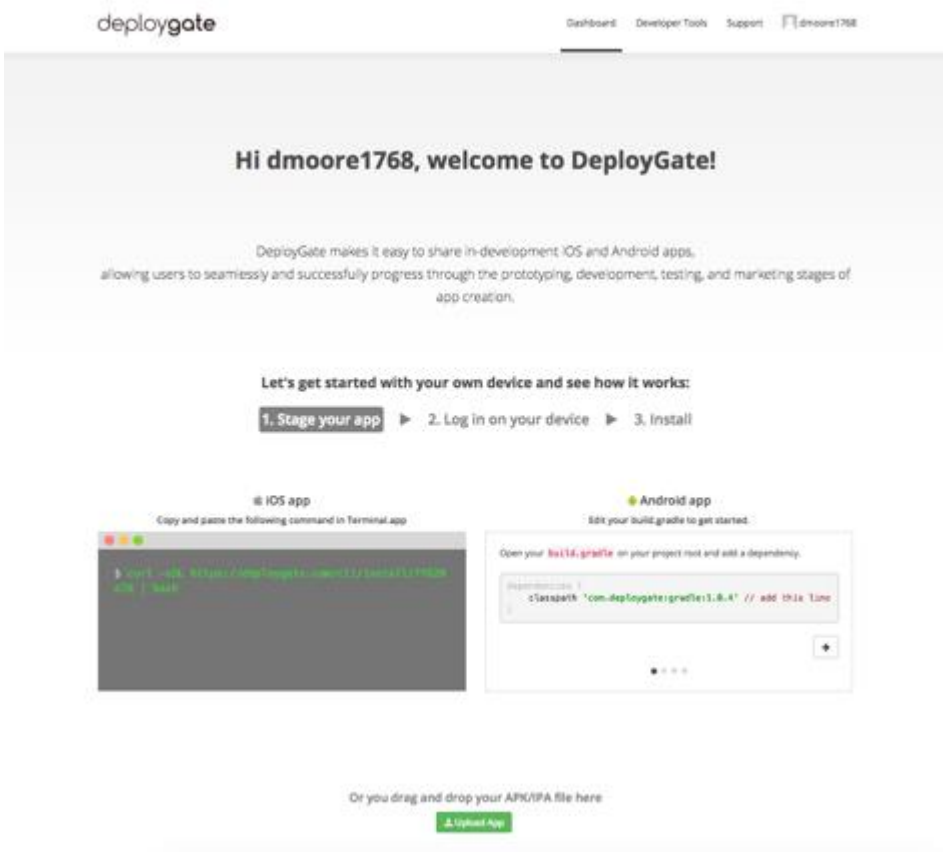
The first step is to create a DeployGate account. To do this:

1. Go to https://deploygate.com
2. Press the **Get Started** button.
3. Enter an email address, a user name and a password.
4. Press the **Sign up for DeployGate** button.

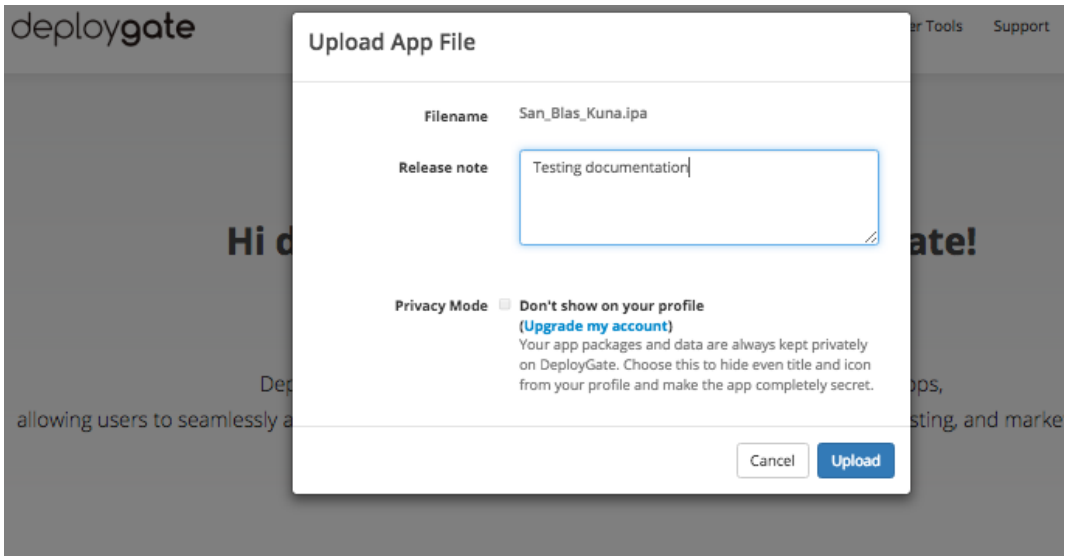


12.2. Uploading Your First App

Once the sign in screen has been successfully completed, you are presented with a screen that prompts you to upload your app. While there are several methods described for embedding it as part of your build process, the way we have used this to date is to simply upload the IPA file that has been created by locating the ipa file in Finder and then dragging it to the bottom area of the screen where it has a green **Upload App** area:

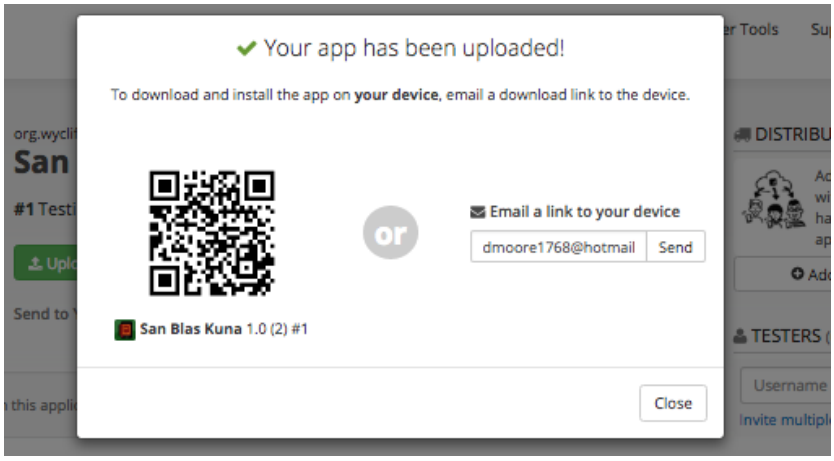


Dragging the file to the upload area causes a new dialog to be displayed with the name of the file and a text box where you can enter a short note that will be displayed on your profile window and also on the DeployGate app when the user is selecting the app. It is a good place to write a short note on the reason for the update so that it is easy for the testers to see that the app has been updated and to see what the primary reason they need to update is. Complete the screen and press the **Upload** button.

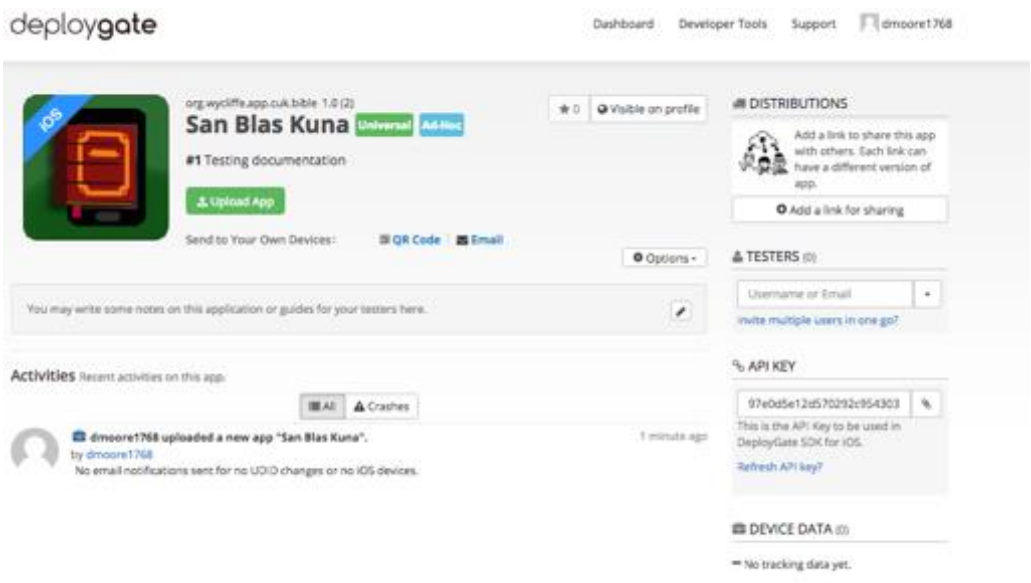


When the upload is complete, a new dialog is displayed with a QCode bar code and the option to send an email to your device. The QCode can be read in with your iPhone or iPad which will trigger an installation of the DeployGate app using your app profile.

Alternatively, you can enter an email address at this point, which you would also open on the iPhone to install the DeployGate app with the correct profile. Or you can simply go on and add users and devices later.



After this, the screen that is displayed is what you will normally see when you login. The screen has an entry for each app that you have uploaded. It has an **Upload App** button that can be used to upload new versions of the same app or to upload a new app.



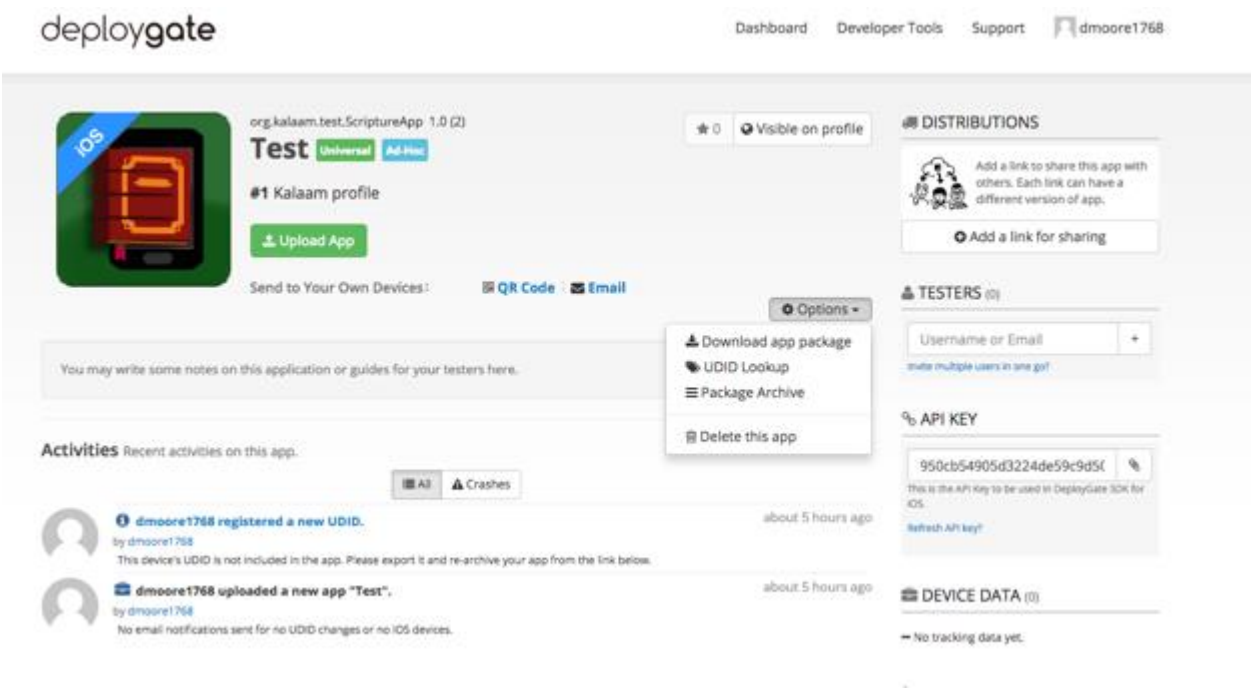
12.3. Registering a Device

If the iOS device was not originally in the mobile provisioning profile and if the device has not been previously registered in your Apple Developer account, you need to add it to both.

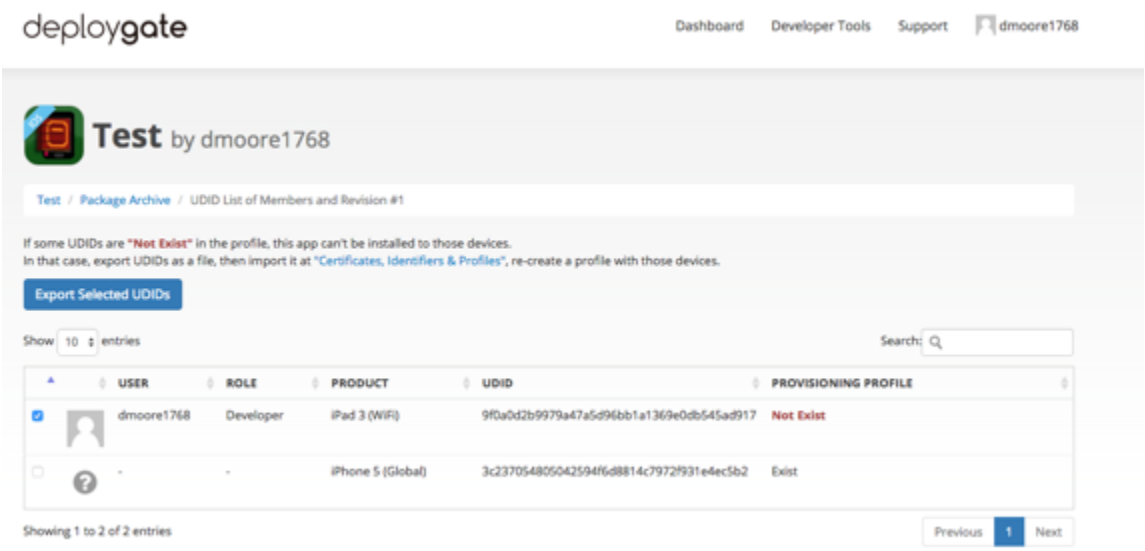
There is a method for manually doing this, but DeployGate provides a way of simplifying the process so that you don't need to go and look up UDID for the device.

First, try to install the app using the method described above. You will not be able to install it because the UDID is not registered for the app. However this will result in the device being registered in DeployGate which allows the following steps.

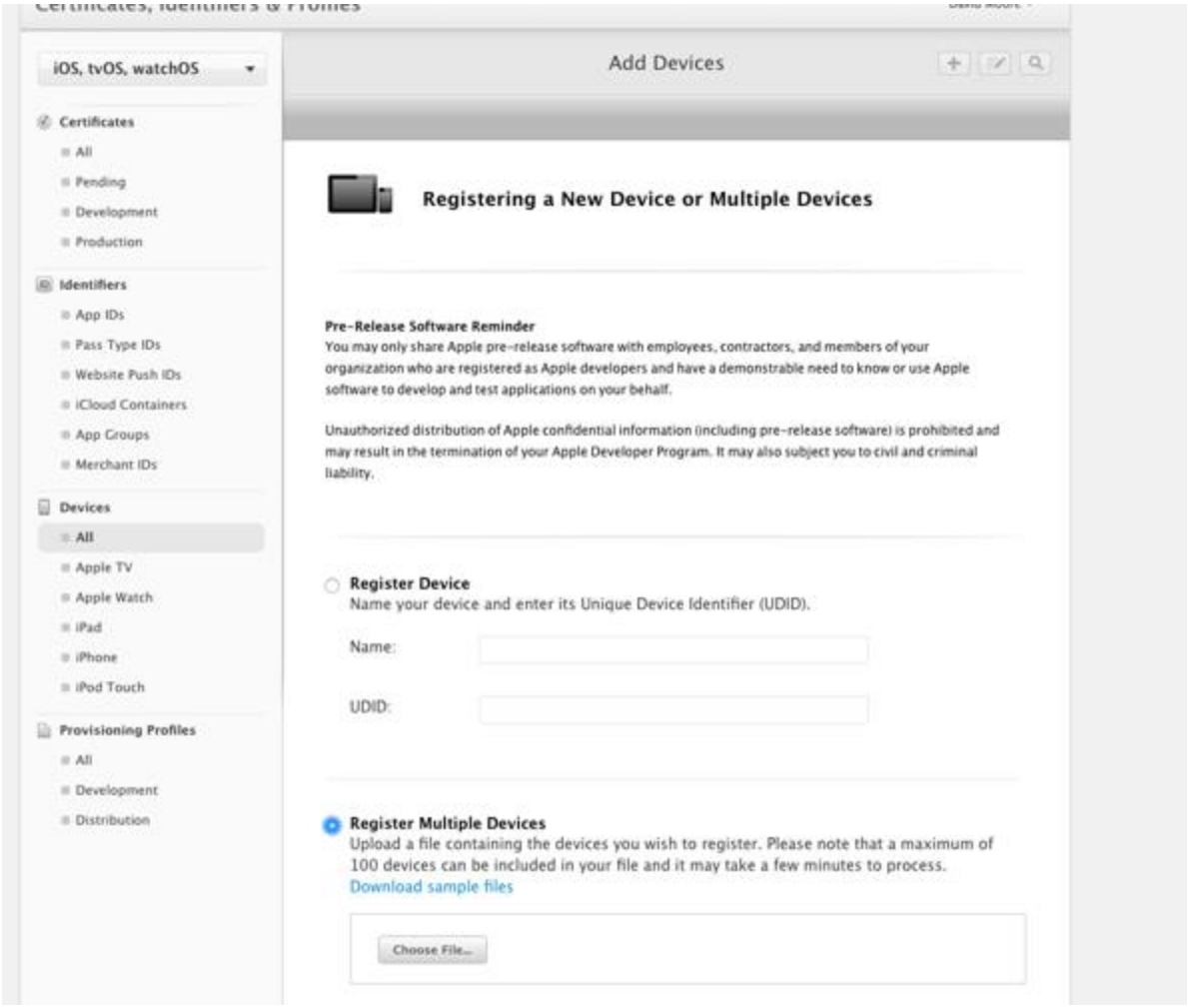
After attempting to install the app, re-enter DeployGate in your browser and open the entry for your app. As you can see in the screen below, it will show that a new UDID has been registered for the device. Press the **Options** button below and select the **Package Archive** option. Next click the little tag symbol inside the app box to open the UDID list.



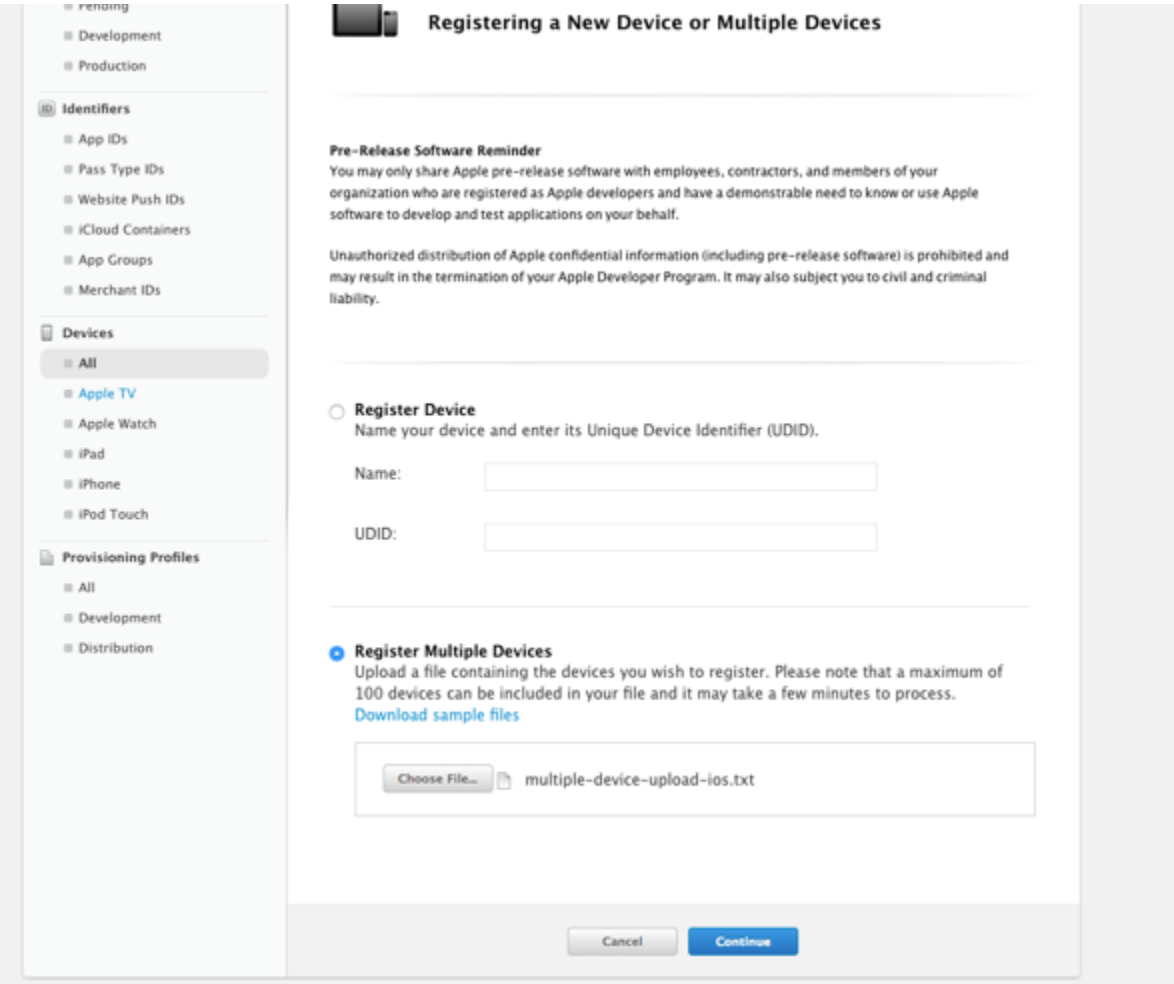
The next screen shows a list of the devices that have been observed by DeployGate or that were included in the provisioning profile. Your new device entry will show up on the screen with a **Not Exist** entry. Make sure that the entry for the new device is checked and then press the **Export Selected UDIDs** button. This will create a file "multiple-device-upload-ios.txt" that can be used on the Apple Developer website to add these devices to the mobile provisioning file.



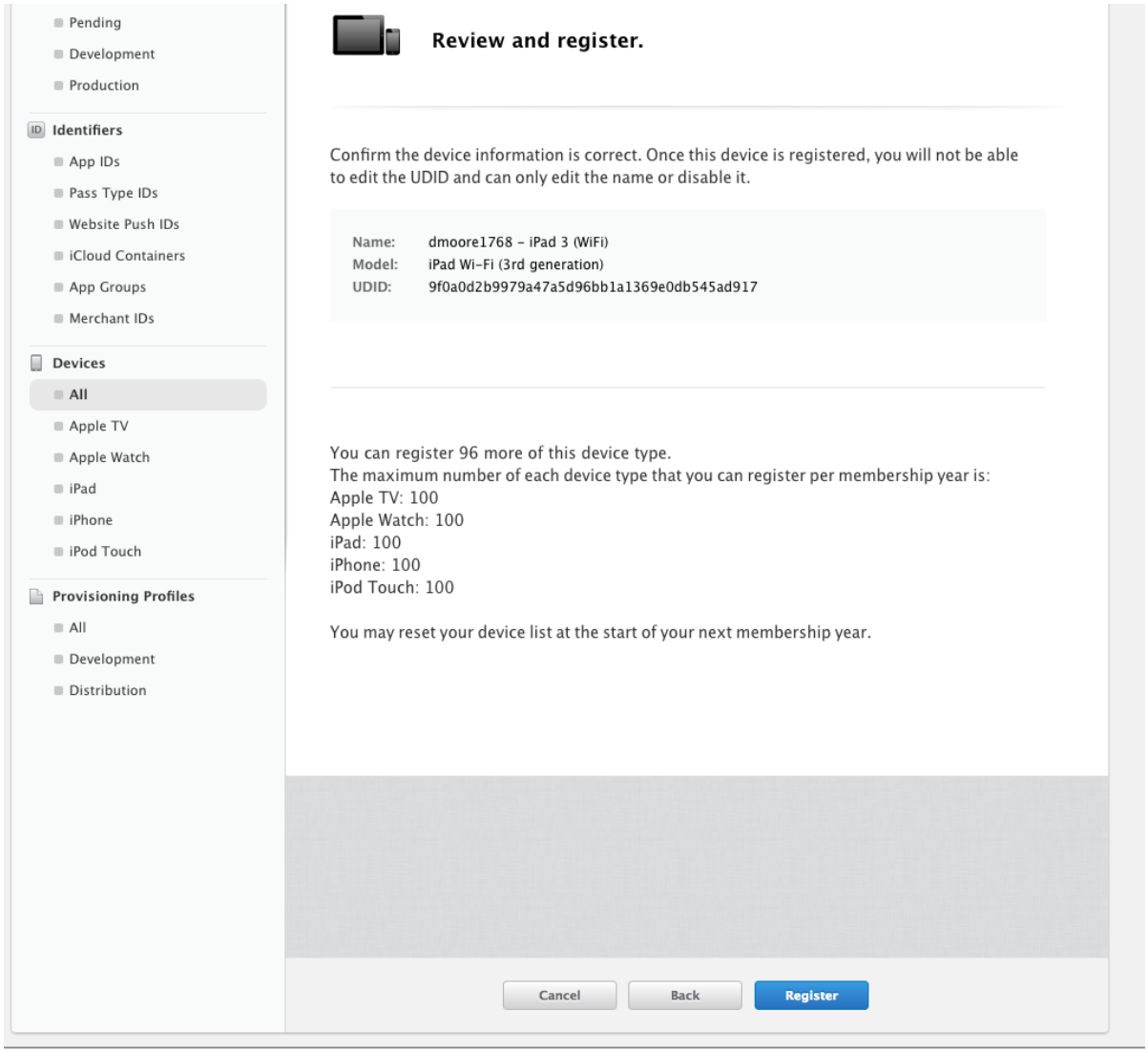
After logging in to Apple Developer, click on **Certificates, IDs and Profiles**. Press the **All** selection under **Devices** as shown in the illustration below and then select **Register Multiple Devices**. Then press the **Choose File** button.



Find the multiple-device-upload-ios.txt file that was created by DeployGate and then press **Continue**.

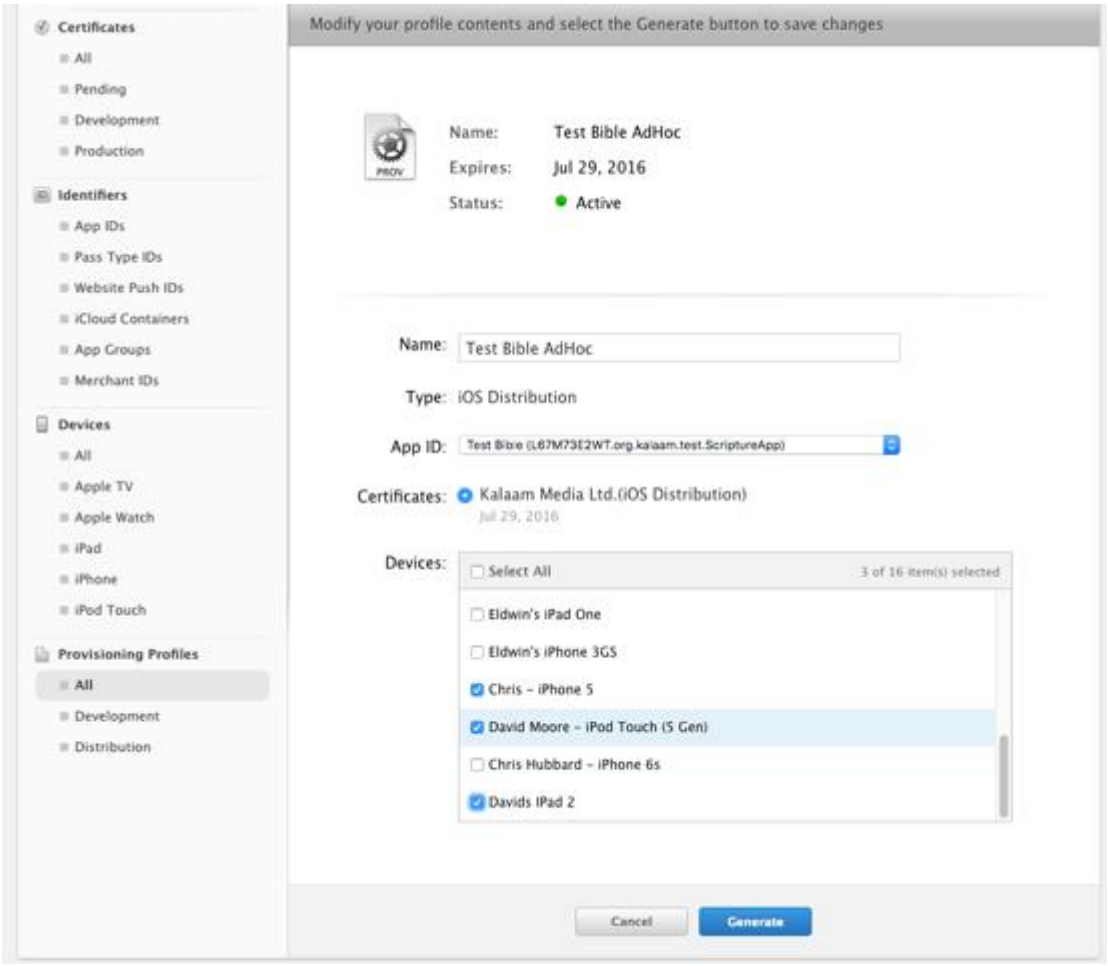


A review screen will be displayed which should list the name to be assigned to the device along with the UDID associated with the device. Review to ensure this is correct and press **Register**.

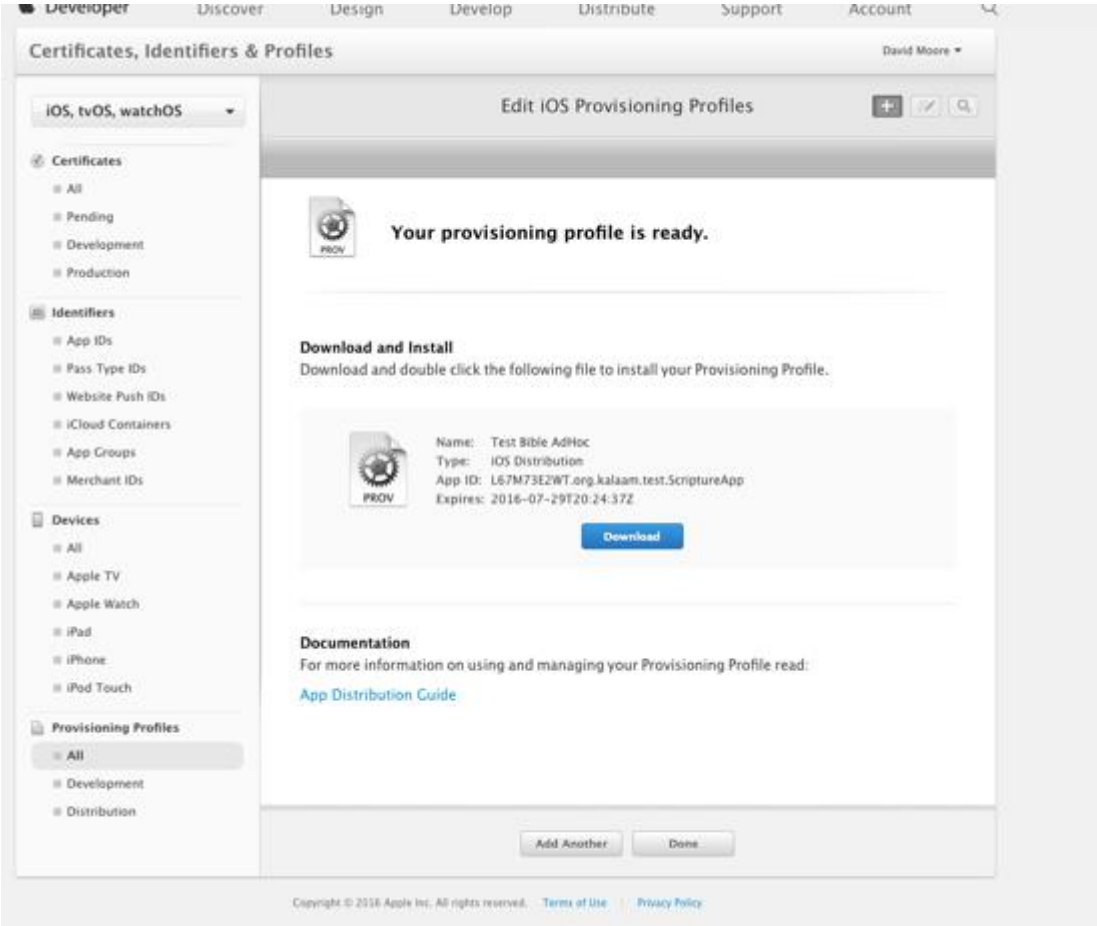


At this point your device has been registered to your Apple Developer account. You now need to add the device to the provisioning profile for your app. Select the provisioning profile being used to test the app. Make sure your test device is checked in the list of devices at the bottom of the screen. Press **Generate**.

Note: For the purposes of testing with DeployGate, an AdHoc type of provision profile must be created and used. If you have not selected AdHoc, the list of devices will not be available on the screen.



The following screen will display:



Now you can download the new provisioning profile that has been generated.

Next you need to rebuild the iOS app using the new profile and upload it to DeployGate again. This time when you attempt to install it, the **Install** button should be enabled and will allow you to install your app.

13. Uploading iOS App to Apple App Store

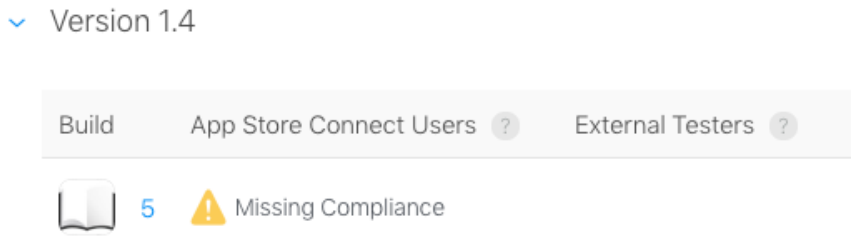
Before attempting to upload the app, you will need to create an entry in App Store Connect (https://appstoreconnect.apple.com).

Select the appropriate distribution **Signing Identity** and **App Store** provisioning profile on the **Signing (iOS)** tab and click Build iOS App. This will create an IPA file in the IPA Output Folder.

Launch **Transporter** and click **Sign In** using the Apple ID for your Apple Developer Account. Drag and drop the IPA file from the IPA Output Folder to Transporter. Once the app is added to Transporter, click on the **Deliver** button. After it is uploaded, you can click on the **…** button and select **View in App Store Connect**. Selecting the Activity tab will show the status of the processing of the upload.

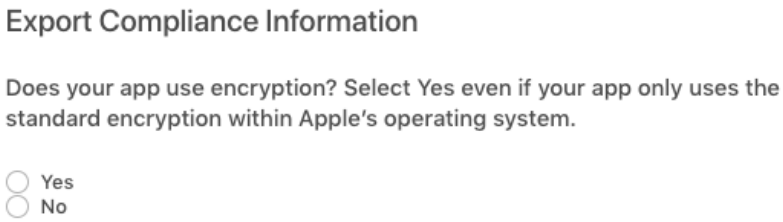
14. Using Test Flight to Test an iOS App

It will take a little bit of time (around 20 minutes) for the app to be processed in App Store Connect. You will receive an email when the app is done processing. Switching to the Test Flight tab in App Store Connect, you will see that app is Missing Compliance.



Click on the build number and you will be taken to a page where you can click on **Provide Export Compliance Information**. The app uses an encryption algorithm to protect the text of the app.

Click **Yes** to the first dialog that the app uses encryption and then click **Next**.



Click **Yes** to the second dialog to indicate that the app qualifies for an exemption due to **(b) Limited to intellectual property and copyright protection** and then click **Start Internal Testing**.

You can add App Store Connect users (normally users in your organization) to test the app. There is a link at the left for **Add External Testers**. This will require the app to go through Beta App Review.

15. Apple Privacy Policy

Prior to publishing your app through **App Store Connect**, you will be required to complete the privacy policy details section. The answers to these questions will be used to create the privacy policy section in your app store entry that describes to the user how you are using their data. The types of data associated with each question is documented by Apple at https://developer.apple.com/app-store/app-privacy-details/. The answers associated with the SAB app depend upon the use of analytics and what data is saved if you have user registration defined for your app.

15.1. Data Types

Under the initial *Data Collection* screen in this section, if the app does not use analytics or user registration, you may indicate that the app does not collect any data. If either of these features are in use, then you should respond that we do collect data from this app. If you answer *Yes* to this question, then you need to specify in the next screen the types of data collected. For an app that has analytics enabled, *Product Interaction* under *Usage Data* and *Crash Data* under *Diagnostics* should be selected. If the app has User Registration enabled, then some of the fields under *Contact Info* may need to be checked depending upon the fields you have configured for the user to enter.

15.2. Product Interaction

This section should only be required if analytics or user registration are enabled. If analytics is enabled, then the *Analytics* entry should be checked to indicate that the data is being used to track user behavior. If user registration is enabled, you may also want to check the *App Functionality* entry. On the next screen, you should check the "*No, product interaction data collected from this app is not linked to the user's identity*" entry. On the final screen, regarding tracking, you may click the "*No, we do not product interaction data for tracking purposes".*

15.3. Diagnostics

This section should only be required if analytics is enabled. The entry for *Analytics* should be checked.

16. Building from Terminal

Scripture App Builder has a command line interface which allows you to create a new app and build it, or load an existing app and build it.

The base command calls java to access the jar file within the Scripture App Builder application followed by a series of options described below. The base command is:

```
java -jar "/Applications/Scripture App
Builder.app/Contents/Java/bin/scripture-app-builder.jar"
```

The available parameters are:

Option	Description
`-new`	Create a new app project
`-load <project>`	Load an existing app project
`-build`	Build app project (use with either -new or -load)
`-no-save`	Do not save changes to app (use with -load)
`-resign`	Resign iOS Template App (use with either -new or -load)
`-?`	Show command line help
`-n <app-name>`	Set app name. Enclose the name in "double quotes" if it contains spaces.
`-p <package-name>`	Set package name, e.g. com.myorg.language.appname
`-b <filename>`	Add book or bundle file. This could be a USFM file or a zipped set of USFM files. It could also be a Digital Bible Library text release bundle.
`-i <filename>`	Include additional parameters file. Use the full path of the file and enclose it in "double quotes" if there is a space in the path.
`-a <filename>`	Set about box text, contained in text file. Use the full path of the file and enclose it in "double quotes" if there is a space in the path.
`-f <fontname>`	Set font name or filename, e.g. "Charis SIL Compact", "c:\fonts\myfont.ttf" The font name must be one of the items in the list of fonts in the New App wizard. For other fonts, specify the full path to the font filename.
`-g`	Use Grandroid

-ic <filename>	Add launcher icon (one or more .png files). Use the full path of the files and enclose them in "double quotes" if there is a space in the path.
-l <lang-code>	Set language for menu items and settings, e.g. en, fr, es
-ft <feature=value>	Set a feature, e.g. book-select=grid
-vc <integer>	Set version code, e.g. 1, 2, 3, or +1 to increment the current version code by 1.
-vn <string>	Set version name, e.g. 1.0, 2.1.4, or use +1, +0.1, +0.0.1 to increment the current value.
-ks <filename>	Set keystore filename. Use the full path of the file and enclose it in "double quotes" if there is a space in the path.
-ksp <password>	Set keystore password
-ka <alias>	Set key alias
-kap <password>	Set key alias password
-fp <folder=path>	Set a folder path, e.g. "app.builder=c:\Scripture App Builder".
-ta <target-api>	Set Target API, e.g. 21 for Android 5.0, 22 for Android 5.1.
-si <signing identity>	Set Signing Identity to use for iOS Resigning
-pp <provisioning profile>	Set full path to provisioning profile for iOS resigning
-bn <integer>	Set build number for ipa file, e.g. 1, 2, 3, or +1 to increment by 1
-vs <string>	Set version string for ipa file, e.g. 1.0, 2.1.4 or +1, +0.1, +0.0.1

Examples:

```
Java -jar

"/Applications/Scripture App Builder.app/Contents/Java/bin/scripture-app-builder.jar" -load "Mali" -resign -bn "5" -vs "2.3.2" -si "iPhone Distribution: Summer Institute of Linguistics, Inc (SIL) (4YF5X97M4H) " -pp
```

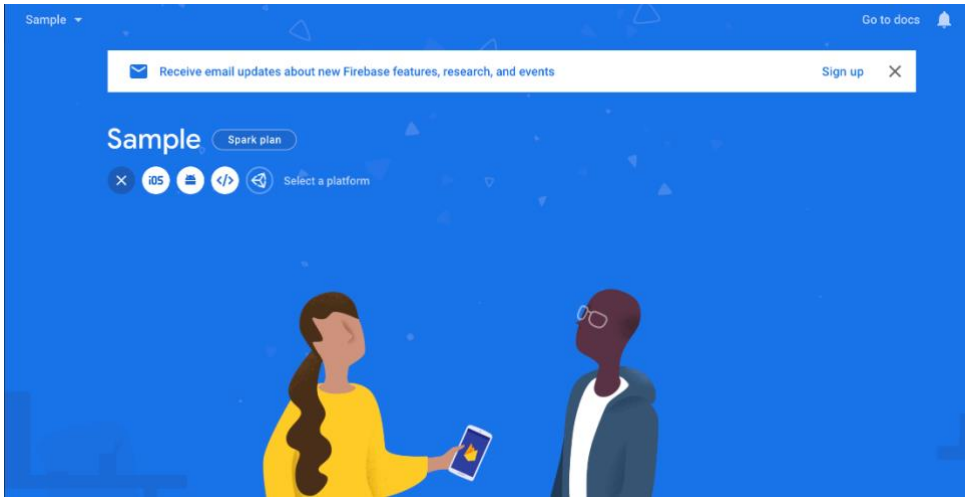
```
"/Users/builder/Documents/MobileProvision/AdHoc_org.wycliffe.app.mali.mobileprovision"
```

17. Using Firebase in an iOS App

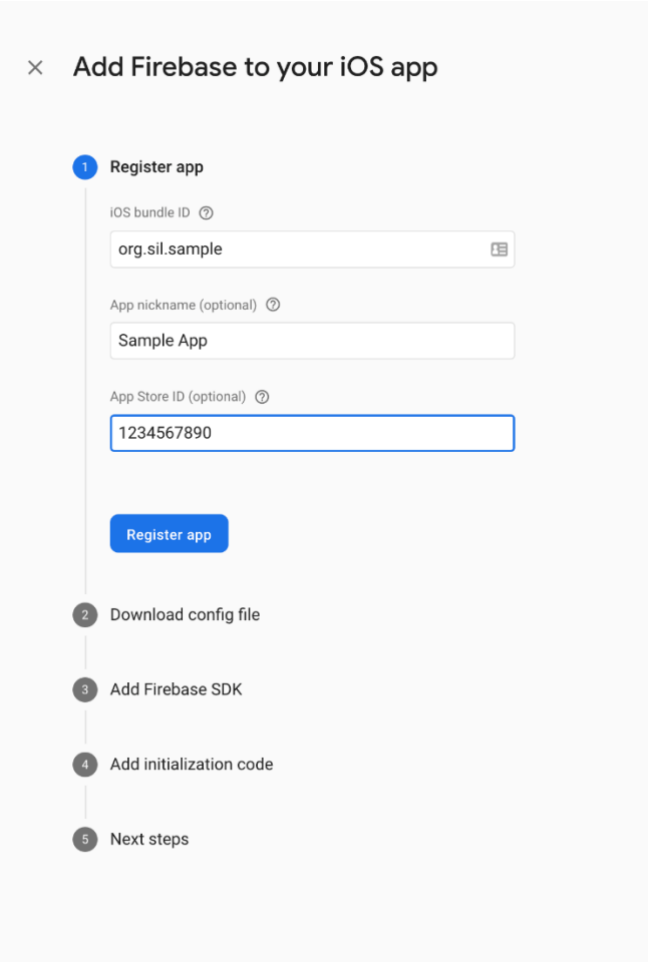
The *Building Apps* document contains general information about setting up the app to support Firebase Analytics, Crashlytics, Messaging and Real Time Database. This section will focus on the steps required to make these features work with an iOS app.

17.1. Adding an iOS App

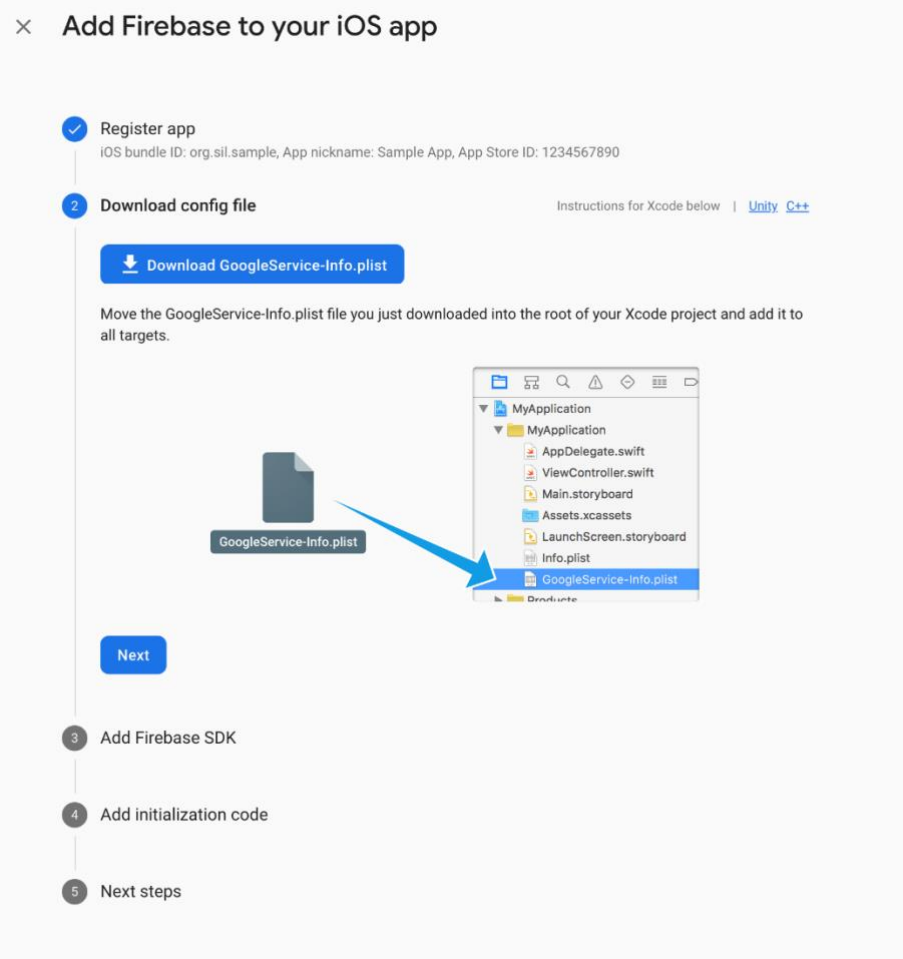
The first step is to go to your Firebase Console (https://console.firebase.google.com/u/0/) and select the entry for the application you are working on. Press the "Add App" button, then select the iOS button.



The next screen will request general information about your application. The iOS bundle ID should match the SAB Package setting from the Package Tab. The App Nickname can be set to whatever you want this app to be referenced as within Firebase. The App Store ID is the same as the Apple ID field in the SAB IPA tab described in Section 9. Once these fields are entered press “Register App”.

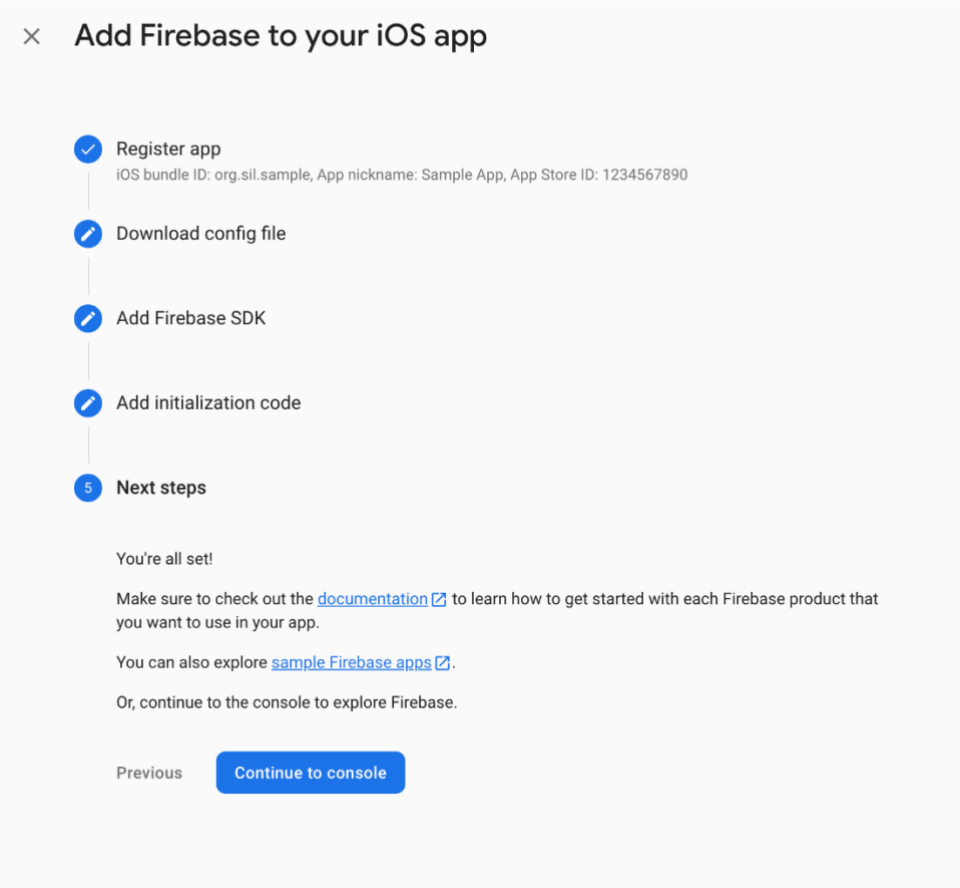


The next screen allows you to download the config file for the application. Press “Download GoogleServiceInfo.plist” to download the Firebase configuration file to your local machine. It should be mentioned here that if your app has the Real Time Database option set in SAB and you have not yet configured the Real Time Database within Firebase Console, you should either do that first or download the plist file again after that configuration has been completed. Download the file to a location where you can locate it. Later in this section, instructions will be provided for adding this file into your SAB application. After downloading the file, press “Next” to continue.



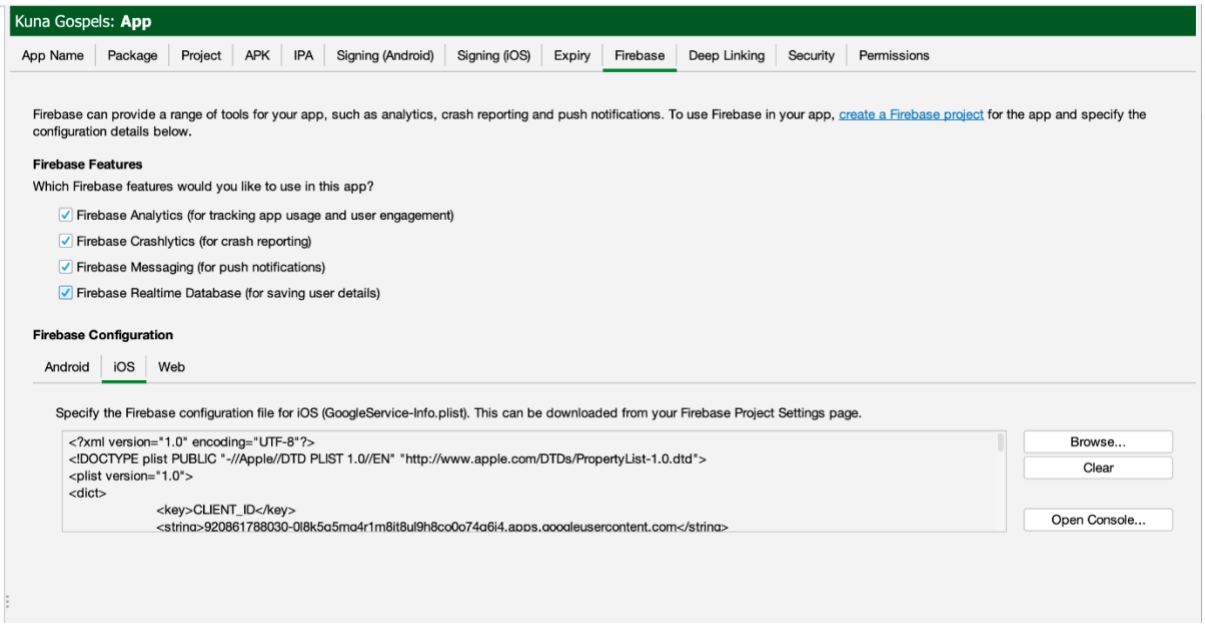
The screens which follow the download step are all instructional screens providing information about programming changes that need to be made to add Firebase to an app. This work has already been done within SAB and can be ignored here.

The final screen just provides a means to return to the console. The addition of the iOS app to the Firebase console is complete.



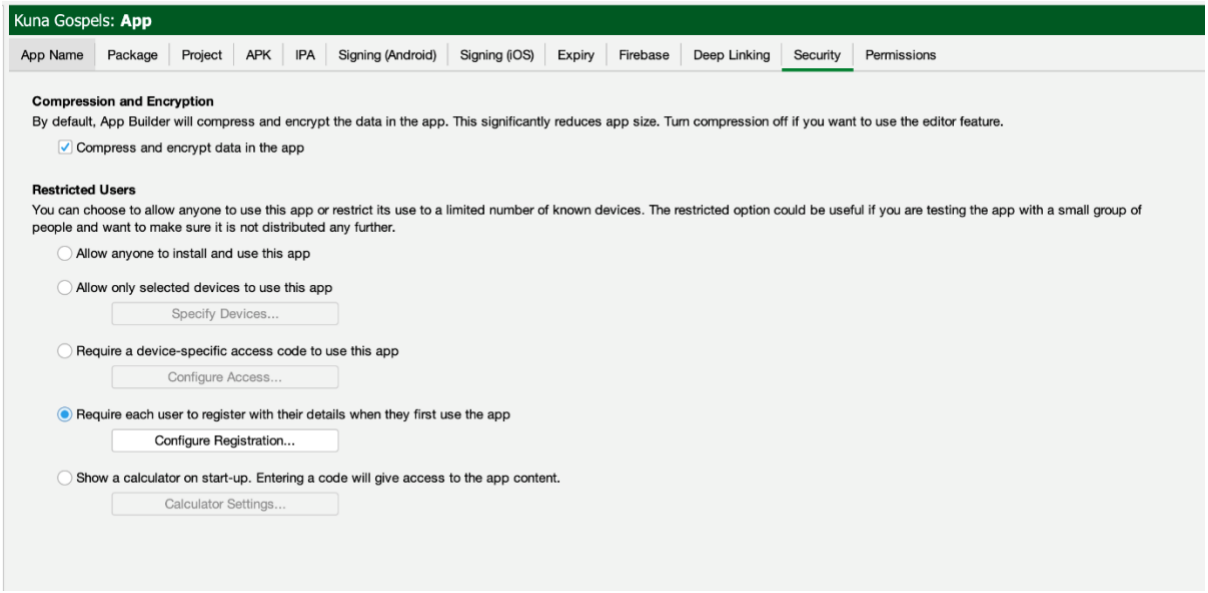
17.2. iOS Configuration for Firebase in SAB

After the app has been added to the Firebase configuration through Firebase Console, the configuration file that was downloaded needs to be added to SAB. On the **Firebase** tab, in the **Firebase Configuration** section on the bottom half of the screen, press the **iOS** tab. Press the **Browse** button to locate the *GoogleService-Info.plist* file that was downloaded during the Firebase configuration process above. Select that file to add it to the app configuration. Note that if Firebase features are checked as enabled, the iOS app will not build until the plist file has been added.



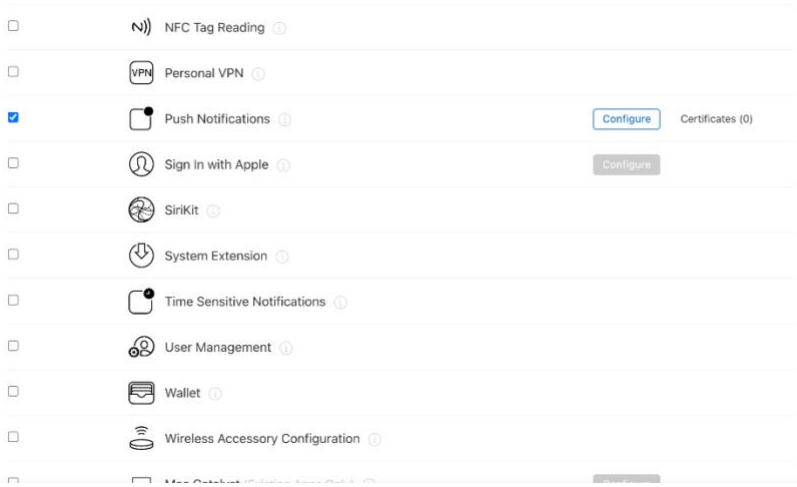
17.3. Security Feature Support in iOS App

The iOS app supports a subset of the security features available in SAB. These features can be accessed through the **Security** tab within SAB. iOS only supports the default *Allow anyone to use this app* and the *Require each user to register with the details when they first use the app* features. The other features are available for Android apps but are not implemented in the iOS app at this time. The registration configuration for this feature works identically for both Android and iOS. No special steps have to be taken specifically for iOS to configure this option if it is selected.



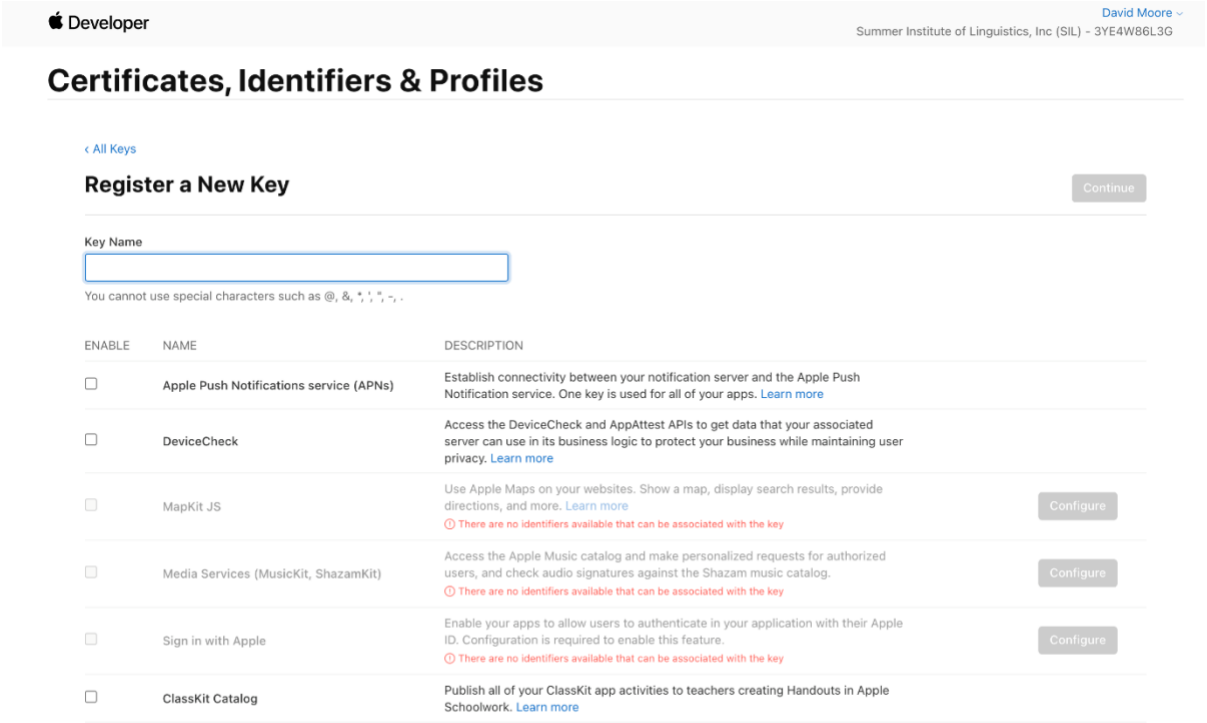
17.4. Firebase Messaging

If Firebase Cloud Messaging is configured to allow push notifications for the app, several changes are required for the app's configuration information in the Apple App Store. The first thing that is required is that the App's configuration must be changed in Apple Developer to include Push Notifications. Under *Certficates, Identifiers, and Profiles*, select the app that is being configured. Enable the Push Notifications option from the list.

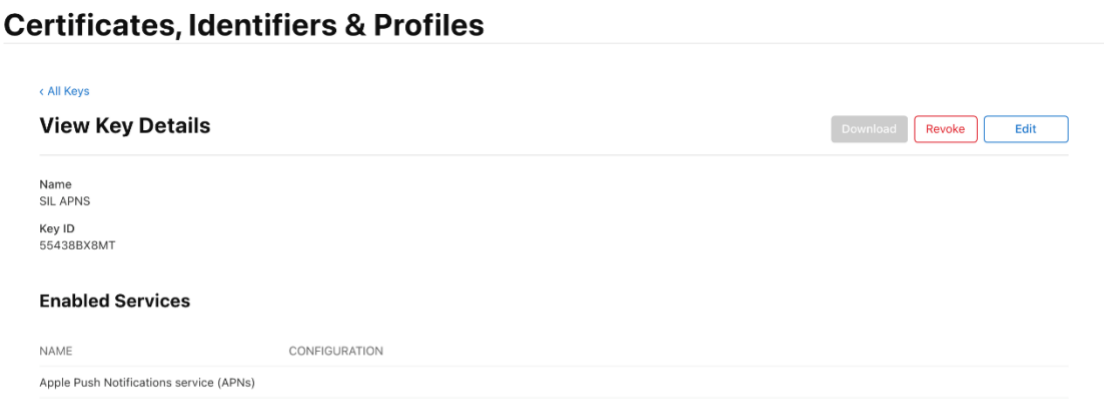


After making these changes, the provisioning profile for the app should be updated for the new capability and the entry updated in SAB.

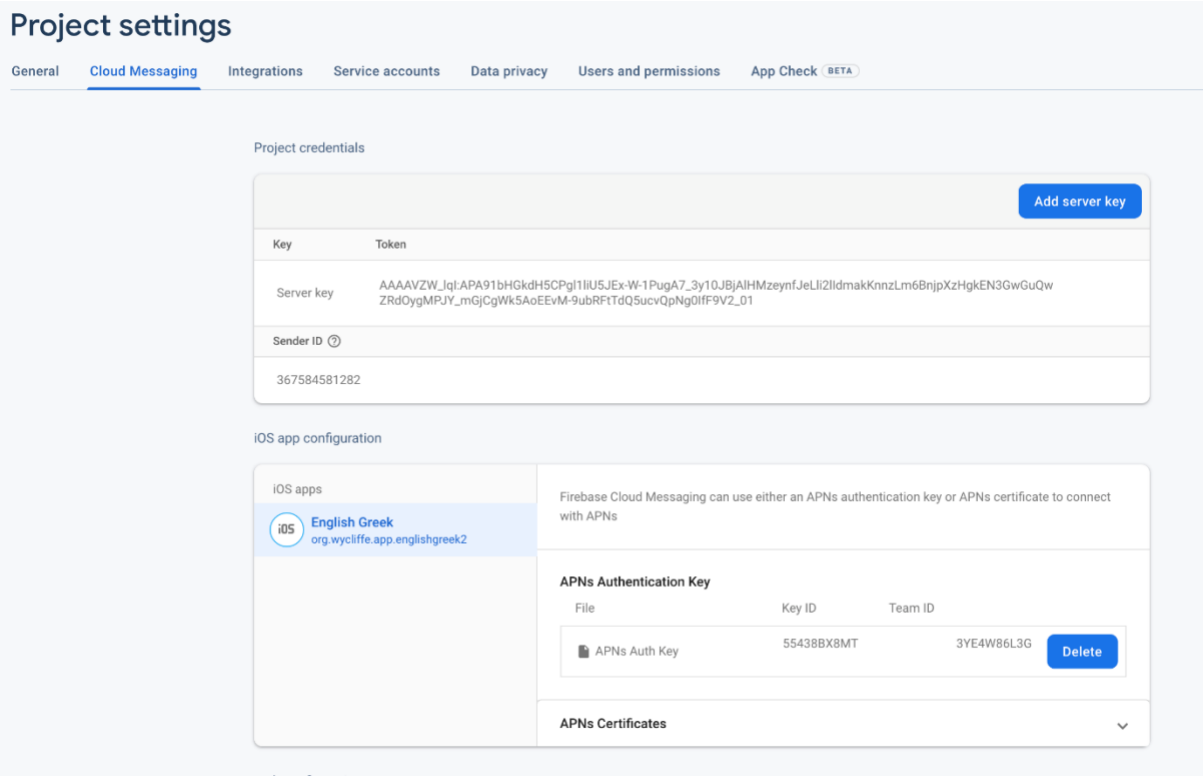
For Firebase to send notification to the app, it will need to have a key uploaded to it. The preferred manner is to go to the *Keys* section in Apple Developer and select to create an Apple Push Notifications service key, using the first option on the screen below. One key of this type is created to be used with all apps associated with this organization. If a key has already been created, then use that.



At the time the key is created, a .p8 file is generated which can be used for all apps. You are allowed to make two keys through this interface, so that you can create a new key and revoke the old one, but only one is active at a time.



Once the p8 file for this key has been downloaded, the file can be uploaded to the Firebase configuration for cloud messaging for this app.



17.5. Firebase Crashlytics for iOS

If Crashlytics is configured for the app in Firebase, the dSYM file associated with the app should be uploaded to get the detailed information about any crashes. To upload dSYMs, you'll need to use the *upload-symbols* command line tool that ships with the Crashlytics SDK. One way to obtain the tool is to install the Crashlytics SDK and locate the tool in the "FirebaseCrashlytics" folder where you installed the SDK. The tool has also been uploaded to an App Builder share drive. The link to this file is:
https://drive.google.com/file/d/1RJIDZyQUbfhoqg9IGZg1YggNMJQQVeue/view?usp=sharing
After downloading this file, cd to the download location and enter "chmod +x upload-symbols" to make the script executable. Otherwise a "Permission denied" may be encountered when attempting to run the script.

The dSYM file that is required is supplied as part of the build process if Crashlytics is configured for the application. The dSYM file is created in the ipa output folder with the same name as the ipa file, with a dSYM extension.

Upload the dSYM file using the *GoogleService-Info.plist* file associated with this application by opening a Terminal window. The command line to upload the dSYM file is:

```
/path/to/upload-symbols -gsp /path/to/GoogleService-Info.plist -p ios
/path/to/TemplateApp.app.dSYM
```

For the example where all three files have been placed in the same directory and the Terminal window has cd'd to that directory, the command would be:

```
./upload-symbols -gsp ./GoogleService-Info.plist -p ios ./TemplateApp.app.dSYM
```

18. Deep Linking for iOS

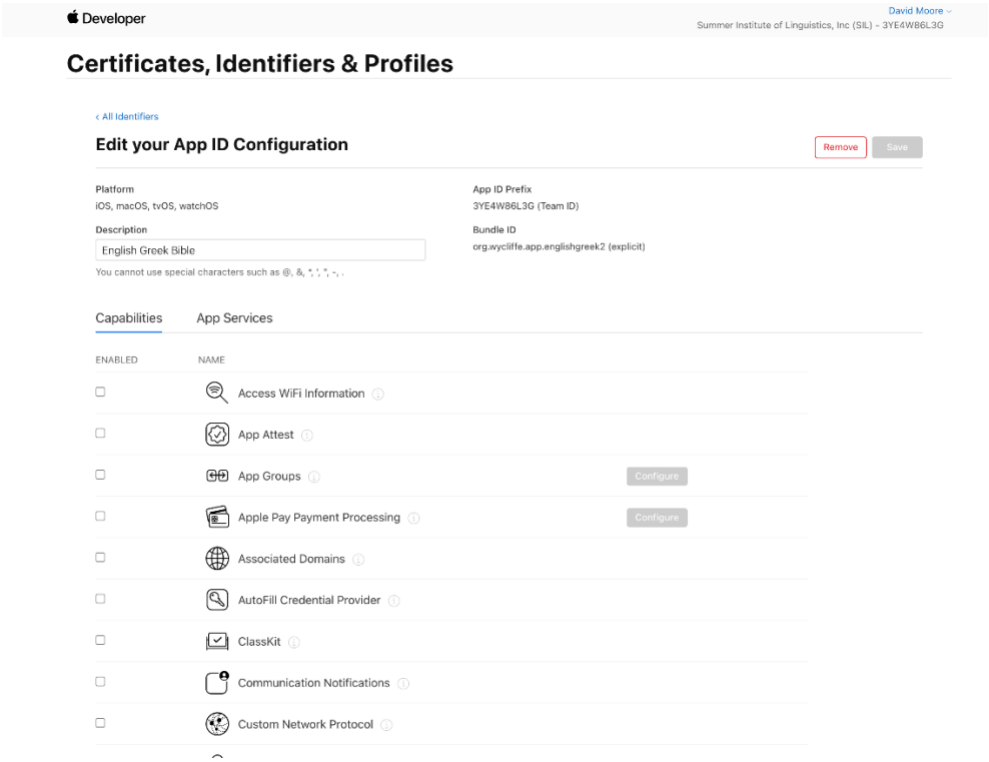
The iOS version of SAB now supports Deep Linking. Within SAB, the setup of this feature for an app is identical to what is described in the Scripture App Builder: Building Apps section on Deep Linking. Additional steps are required for iOS depending upon the deep linking configurations required.

18.1. Provisioning Profile Changes (Associated Domains)

Any type of deep linking configuration will require updates to the app configuration in Apple Developer. The definition of the app in Apple Developer must be updated to include the Associated Domains capability. To update an app, perform the following steps:

- Log into your Apple Developer Account
- Select "Certificates, IDs and Profiles" from the main menu
- Select "Identifiers" from the menu on the screen that appears. This will provide a list of the apps that have already been created.

- Select the app that you are adding the Deep Linking capability to.
- A screen similar to the one below should display:



- Check the “Enabled” box in front of “Associated Domains” and then press “Save”. This will generate a warning that adding or removing capabilities will invalidate your current provisioning profiles. Press “Confirm” to continue
- Select “Profiles” from the menu on the left to get access to the Provisioning Profile for the app.
- Select the provisioning profile associated with this app (which should be listed as “Invalid”) and press “Edit”
- No changes should be required. “Associated Domains” should appear in the list of “Enabled Capabilities”. Press “Save”.
- Press “Download” to download the provisioning profile to the system where SAB is installed and update the “Profile File” entry on the “Signing iOS” tab to point to the updated provisioning profile.

18.2. Deep Linking using URL Scheme

A URL that supports deep linking for an iOS app must contain an Associated Domain File. The section “Add the Associated Domain File to Your Website” in https://developer.apple.com/documentation/xcode/supporting-associated-domains describes the required changes. As is stated in this article, the page must be accessible via https://.

An example apple-app-site-association file that was used for testing is:

{

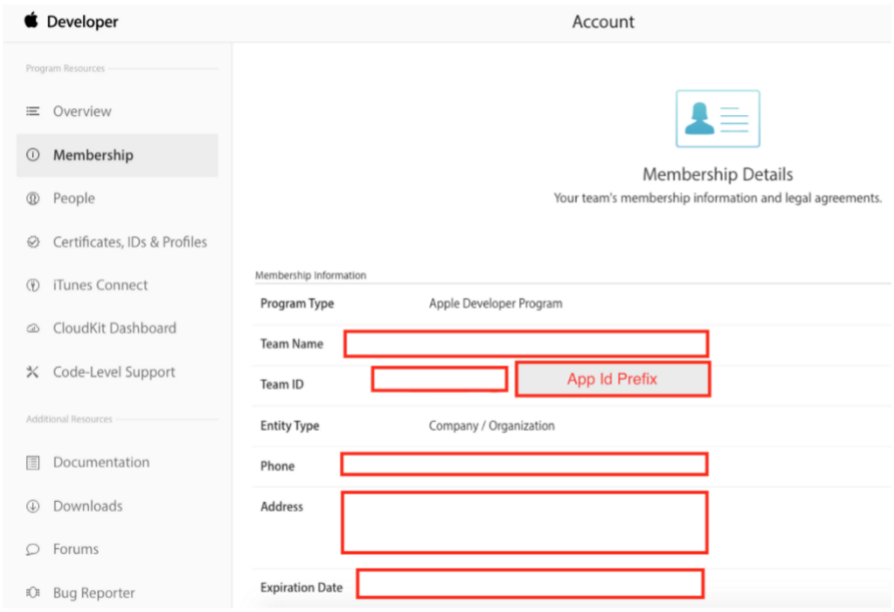
```
  "applinks": {
    "apps": [],
    "details": [
      {
        "appID": "3YE4W86L3G.org.wycliffe.app.sabtest",
        "paths": ["*"]
      }
    ]
  }
}
```

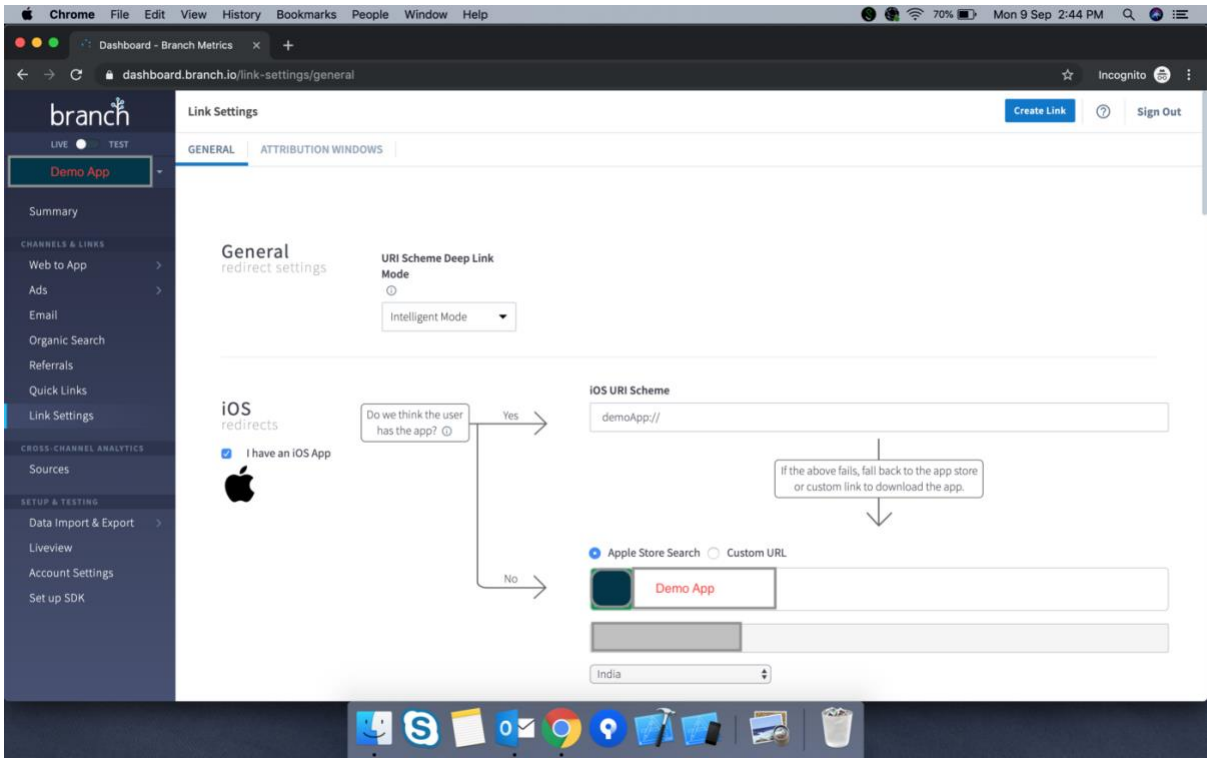
18.3. Deferred Deep Linking – Using Branch

When Branch.io is used to implement deferred deep linking, the Branch dashboard must be filled out to indicate that you have an iOS app. The following is an excerpt from a web page that describes all of the steps needed to use Branch.io for an iOS app. Most of the information there is already done by either SAB or the template app.
https://medium.com/flawless-app-stories/ios-universal-links-and-app-referrals-using-branchio-integration-31dd474be20

On branch dashboard under link settings, tick iOS and enter below details:

- **iOS Url Scheme:** If the app is registered with some scheme then we need to specify here. When a referral/universal link is clicked then it will use scheme to launch the app.
- **Apple Store Search:** If the app is not installed then you can redirect the user to the App Store for that particular app. Search your app here by name.
- **Custom URL:** You can also redirect the user to a particular website instead of the App Store. This might be a case where either of one platform(iOS or Android) is supported and for the other, you want to redirect the user to a website for further communication.
- **Bundle Id:** Give the bundle id of the app.
- **Apple App Prefix:** The App prefix can be retrieved from app developer account or even from Xcode -> General -> Signing. It is just a Team Id.


Developer
Account
Program Resources
Overview
Membership
People
Certificates, IDs & Profiles
iTunes Connect
CloudKit Dashboard
Code-Level Support
Additional Resources
Documentation
Downloads
Forums
Bug Reporter
Membership Details
Your team's membership information and legal agreements.
Membership Information
Program Type
Apple Developer Program
Team Name
Team ID
App Id Prefix
Entity Type
Company / Organization
Phone
Address
Expiration Date


Chrome File Edit View History Bookmarks People Window Help
Mon 9 Sep 2:44 PM
Dashboard - Branch Metrics
dashboard.branch.io/link-settings/general
Incognito
branch
LIVE TEST
Demo App
Summary
CHANNELS & LINKS
Web to App
Ads
Email
Organic Search
Referrals
Quick Links
Link Settings
CROSS-CHANNEL ANALYTICS
Sources
SETUP & TESTING
Data Import & Export
Liveview
Account Settings
Set up SDK
Link Settings
Create Link
Sign Out
GENERAL
ATTRIBUTION WINDOWS
General
redirect settings
URI Scheme Deep Link Mode
Intelligent Mode
iOS
redirects
I have an iOS App
Do we think the user has the app?
Yes
No
iOS URI Scheme
demoApp://
If the above fails, fall back to the app store or custom link to download the app.
Apple Store Search
Custom URL
Demo App
India