ownCloud Branded Clients Manual

The ownCloud Team

Version: next, September 02, 2022

Table of Contents

Branded Clients

Creating Branded Desktop Client

Building a Branded Desktop Sync Client

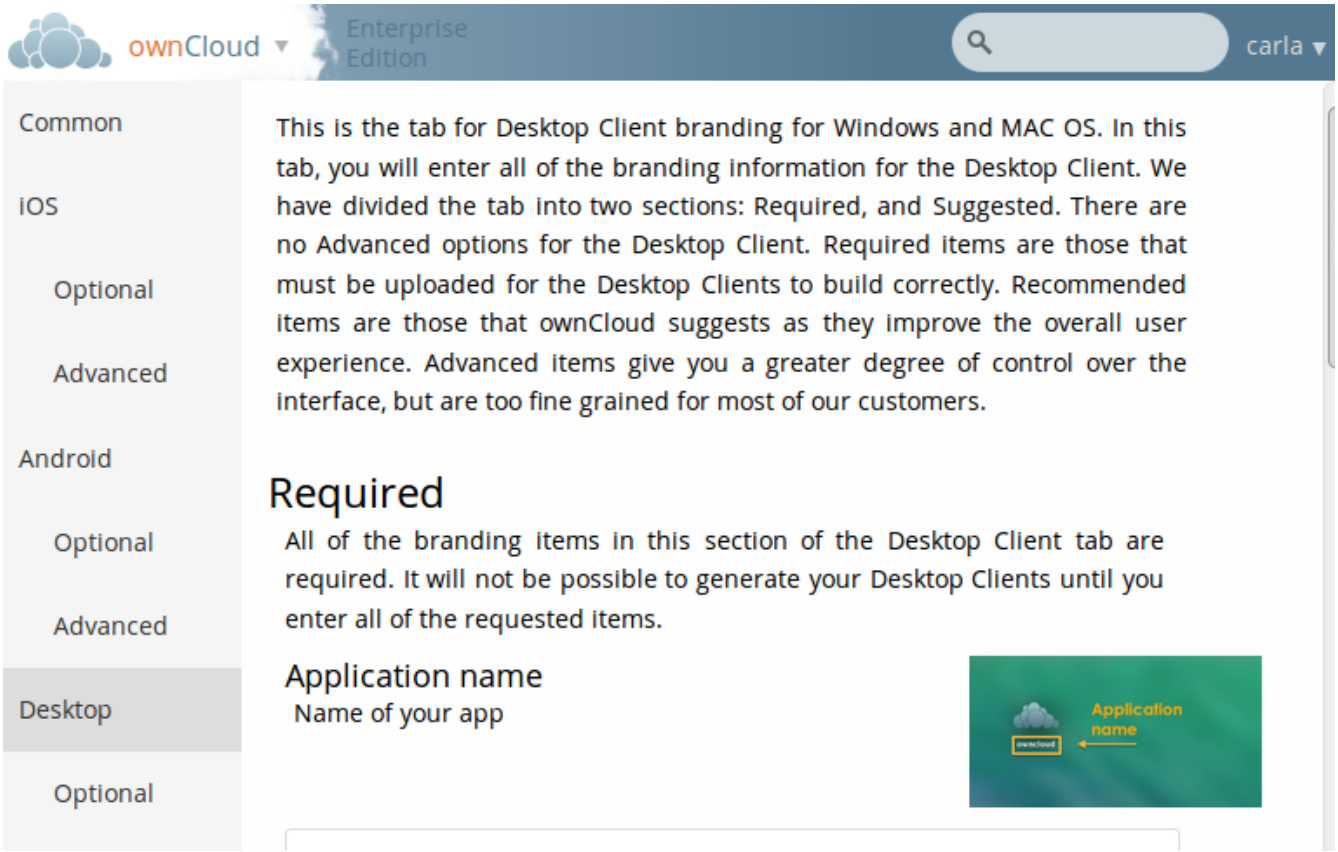
Introduction

To build a branded Desktop sync client, you need to supply your own artwork and use the ownBrander wizard in your account on customer.owncloud.com. The ownBrander wizard details the required image specifications.

Build Process

In the ownBrander wizard at your account, start with the Common section at the top, and enter information common to all clients that you can build with ownBrander. You may override any settings inside the *Common* section of the *Client* sections.

Then go to the *Desktop client section* of ownBrander, which has two sections, *Required* and *Optional*.



Work your way through the wizard, enter required elements and any optional elements you wish. When you have completed the wizard, press the **[Generate Desktop Client]** button. You will either get messages warning of any items that need to be corrected, or a success message.

It takes 24-48 hours to build your client. When finalized you will see it in your account on customer.owncloud.com.

Updating Your Branded Desktop Clients

Introduction

The Client Updater Server provides a Web service that will tell an ownCloud Desktop sync client whether or not an update is available. If an update is available, it will also provide metadata for the update, such as the Download URL, signatures or a fallback URL that the client can resort to in case the update goes wrong.

Clients for Mac OS X and Windows will update themselves automatically. Linux clients will not. You have two options for your Linux users:

- Set up your own download repository so your Linux users can update your branded clients with their package managers when they receive an update notification.
- Upload new versions of your branded client to your Web server. Your Linux users receive update notifications, then download and install the client manually.

There are times when you may want to disable update notifications. See the examples below to learn how to do this.

Prerequisites

1. Configure *Update URL* in the *Desktop* section of your ownBrander account (available for *advanced* users only).
    - Example:
      `https://mycloud.example.com/updates/`
      (note the forward slash at the end)
2. Generate branded clients.
3. Upload branded clients to your Web server.
    - Windows example:
      `https://mycloud.example.com/install/mycloud-2.1.1.240-setup.exe`
    - Mac OS X examples:
      `https://mycloud.example.com/install/mycloud-2.1.1.787.pkg`
      `https://mycloud.example.com/install/mycloud-2.1.1.787.pkg.tbz`
      `https://mycloud.example.com/install/mycloud-2.1.1.787.pkg.tbz.sig`
    - You should have a Web page with links to your branded clients, so your users can find and download them. For example, `https://mycloud.example.com/install/` with `Options +Indexes` in your ownCloud `.htaccess` file.

Install client-updater-server

1. Download `client-updater-server-0.4.tar.xz` from https://customer.owncloud.com/
2. Extract `client-updater-server-0.4.tar.xz` to your Web server. The `index.php` must be accessible at `https://mycloud.example.com/updates/index.php`.
3. Copy your ownCloud `config/ownCloud.yml` file, and name it according your **Application short**

name as configured in ownBrander.

Example: `config/mycloud.yml`

Configure client-updater-server

All configuration is done in your `config/mycloud.yml`:

```
throttle: 1 # 100% of the requests get served with the new version

platforms:
  win32msi:
    currentVersion: 2.5.0.10598
    currentVersionString: ownCloud Client 2.5.0 (build 10598)
    updateUrl: https://owncloud.com/desktop-app
    downloadUrl: http://download.owncloud.com/desktop/stable/ownCloud-2.5.0.10598.msi

  win32:
    currentVersion: 2.4.3.10188
    currentVersionString: ownCloud Client 2.4.3 (build 10188)
    updateUrl: https://owncloud.com/desktop-app
    downloadUrl: http://download.owncloud.com/desktop/stable/ownCloud-2.4.3.10188-
setup.exe

  linux:
    currentVersion: 1.8.0
    currentVersionString: ownCloud Client 1.7.1
    updateUrl: https://owncloud.com/desktop-app

  macos:
    currentVersion: 1.8.0.2139
    currentVersionString: ownCloud Client 1.8.0 (build 2139)
    downloadUrl: https://download.owncloud.com/desktop/stable/ownCloud-
1.8.0.2139.pkg.tbz
    pubDate: 2015-03-26
    signature: MCwCFFedScUKeRXYMS6vKVLw821B+/+lAhRbiCxHNzVVZFNXHSvB9GNHOuI5cw==
    minimumSystemVersion: 10.7.0
```

In earlier versions this configuration was written in PHP, which is still supported but no longer the default. The structure slightly changed and would look like this analoguely to the yml config `config/mycloud.php`:

```
<?php

$updateInfo = [
    'throttle' => 0.7, // 70% of the requests get served with the new version
    'platforms' => [
        'win32msi' => [
            'currentVersion' => '2.5.0.10598',
```

```
            'currentVersionString' => 'ownCloud Client 2.5.0 (build 10598)',
            'updateUrl' => 'https://owncloud.com/desktop-app',
            'downloadUrl' => 'http://download.owncloud.com/desktop/stable/ownCloud-
2.5.0.10598.msi',
        ],
        'win32' => [
            'currentVersion' => '2.4.3.10188',
            'currentVersionString' => 'ownCloud Client 2.4.3 (build 10188)',
            'updateUrl' => 'https://owncloud.com/desktop-app',
            'downloadUrl' => 'http://download.owncloud.com/desktop/stable/ownCloud-
2.4.3.10188-setup.exe',
        ],
        'linux' => array(
            'currentVersion' => '1.8.0',
            'currentVersionString' => 'ownCloud Client 1.7.1',
            'updateUrl' => 'https://owncloud.com/desktop-app',
        ),
        'macos' => array(
            'currentVersion' => '1.8.0.2139',
            'currentVersionString' => 'ownCloud Client 1.8.0 (build 2139)',
            'downloadUrl' => 'https://download.owncloud.com/desktop/stable/ownCloud-
1.8.0.2139.pkg.tbz',
            'pubDate' => '2015-03-26',
            'signature' =>
'MCwCFFedScUKeRXYMS6vKVLw821B+/+lAhRbiCxHNzVVZFNXHSvB9GNHOuI5cw==',
            'minimumSystemVersion' => '10.7.0',
        ),
    ]
];
```

(The former top-level config options were moved under a `platforms` key.)

Disabling Notifications

There may be times when you wish to disable update notifications. To do this, make the `'currentVersion'` and `'currentVersionString'` older than the currently installed version. To re-enable notifications, change these to release versions that are newer than the currently installed clients.

Windows

- `'currentVersion'`
  Exact version of the new client, including the build number
- `'currentVersionString'`
  Name of the new client, same as "Application name" configured in ownBrander.
- `'updateUrl'`
  Human-readable Web site with links to your new client files.
- `'downloadUrl'`
  Full URL to download the *.exe file. https needed.

Mac OS X

- `currentVersion'`
  Exact version of the new client, including the build number.
- `'currentVersionString'`
  Name of the new client, same as `Application name` configured in ownBrander.
- `'downloadUrl'`
  Full URL to download the *.pkg**.tbz** file. http**s** needed.
- `'pubDate'`
  Currently not used.
- `'signature'`
  Content of `mycloud-2.1.1.787.pkg.tbz.sig`, adds some extra security to the Mac OS X updater.
- `'minimumSystemVersion'`
  Minimum required Mac OS X version according to https://owncloud.com/desktop-app/

Linux

- `'currentVersion'`
  Exact version of the new client, including the build number
- `'currentVersionString'`
  Name of the new client, same as `Application name` configured in ownBrander.
- `'updateUrl'`
  Human-readable Web site with links to your new client files to manually install new client versions.

Debugging client-updater-server

Windows

This a example URL of a 2.5.0 client for Microsoft Windows:
`https://mycloud.example.com/updates/?version=2.5.0.10598&platform=win32&msi=true&oem=mycloud`

You should see something like the following in your Web server logs:

```
[19/Feb/2016:14:33:35 +0100] "GET
/updates/?version=2.5.0.10598&platform=win32&msi=true&oem=mycloud HTTP/1.1" 200 185 "-
"
"Mozilla/5.0 (Windows) mirall/2.5.0 (mycloud)" microsecs:530450
```

The output should look like this if you call the URL manually:

```
<?xml version="1.0"?>
   <owncloudclient>
      <version>2.5.0.10598</version>
       <versionstring>MyCloud Client 2.5.0 (build 10598)</versionstring>
```

```
        <web>https://mycloud.example.com/install/</web>
        <downloadurl>https://mycloud.example.com/install/
         mycloud-2.5.0.10598.msi</downloadurl>
    </owncloudclient>
```

Mac OS X

This a example URL of a 2.1.1 client for Mac OS X:

`https://mycloud.example.com/updates/?version=2.1.1.687&platform=macos&oem=mycloud&sparkle=true`

You should see something like the following in your Web server logs:

```
[19/Feb/2016:14:00:17 +0100] "GET
/updates/?version=2.1.1.687&platform=macos&oem=mycloud&sparkle=
true HTTP/1.1" 200 185 "-" "Mozilla/5.0 (Macintosh) mirall/2.1.1 (mycloud)"
microsecs:1071 response_size:2070 bytes_received:306 bytes_sent:2402
```

The output should look like this if you call the URL manually:

```
<?xml version="1.0" encoding="utf-8"?>
  <rss version="2.0"
  xmlns:sparkle="http://www.andymatuschak.org/xml-namespaces/sparkle"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel>
     <title>Download Channel</title>
     <description>Most recent changes with links to updates.</description>
     <language>en</language><item>
     <title>MyCloud Client 2.1.1 (build 787)</title>
     <pubDate>Mon, 23 Feb 16 00:00:00 -0500</pubDate>
     <enclosure url="https://mycloud.example.com/install/
       mycloud-2.1.1.787.pkg.tbz" sparkle:version="2.1.1.787"
       type="application/octet-stream"
       sparkle:dsaSignature="MCwCFFedScUKeRXYMS6vKVLw821B+/+
         lAhRbiCxHNzVVZFNXHSvB9GNHOuI5cw=="/>
      <sparkle:minimumSystemVersion>10.7.0</sparkle:minimumSystemVersion>
   </item>
   </channel>
  </rss>
```

Creating Branded iOS Apps

Building and Distributing Your Branded iOS App

Introduction

Building and distributing your branded iOS ownCloud app involves a large number of interdependent steps. The process is detailed in this chapter over several pages. Follow these instructions exactly and in order, and you will have a nice branded iOS app that you can distribute to your users.

Prerequisites

- A Mac OS X computer with Xcode (free download) and Keychain Access (included in Utilities). This computer is essential to the entire process and will be linked to to your iOS Developer account. You will use it create and store distribution certificates, and to upload your app to iTunes Connect.
- An iOS developer account on developer.apple.com/ios, which costs $99 per year. Or an Enterprise account for $299/yr. The developer account limits you to testing on 100 devices of each type (Apple TV, Apple Watch, iPad, iPhone, iPod Touch) which must be registered in your account. The Enterprise account allows testing on unlimited, unregistered devices.
- An ownCloud Enterprise Subscription, with the ownBrander app enabled on customer.owncloud.com
- Some iPhones or iPads for testing your app. Again, if you have the $99 developer account each device must have its UDID registered in your account on developer.apple.com.

Procedure

You need the Apple tools to build eight provisioning profiles (4 Ad Hoc and 4 App Store) and a P12 certificate. You will email the four Ad Hoc profiles and P12 certificate to support@owncloud.com after building your app with the ownBrander app on customer.owncloud.com. You must create the provisioning profiles and P12 certificate first, before building your app, because you must supply a unique **bundle ID** and an **app group** to build your app. These are created in your account on developer.apple.com, and with Keychain Access on your Mac computer.

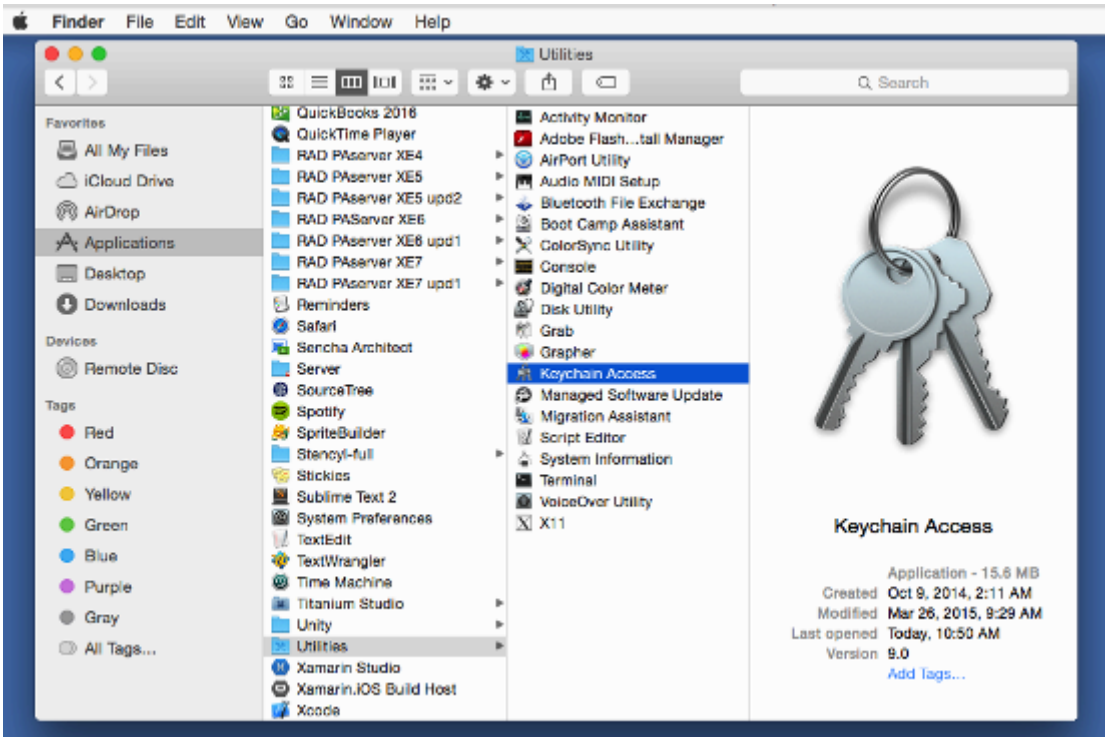
We use the 4 Ad Hoc provisioning profiles and P12 certificate to complete building your app, and then in 24-48 hours your new branded app is loaded into your account on customer.owncloud.com.

The next step is to test your app. When it passes testing, the final step is to upload it to your iTunes Connect account for distribution.

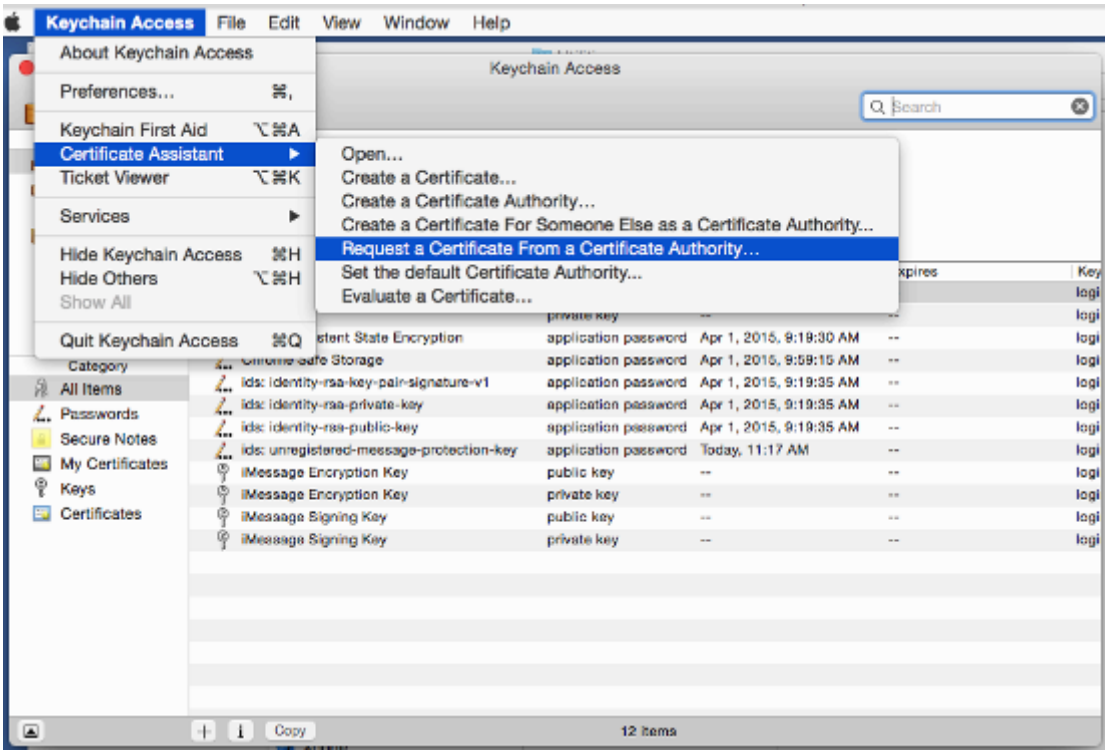
You will need a lot of graphics for building your app, and for your iTunes store listing, in specific sizes and file formats. The ownBrander app and iTunes detail all the image specifications you will need.

Create Certificate Signing Request

Start by creating a .certSigningRequest (CSR) file on your Mac, using Keychain Access. Open Finder, and then open Keychain Access from the Utilities folder.

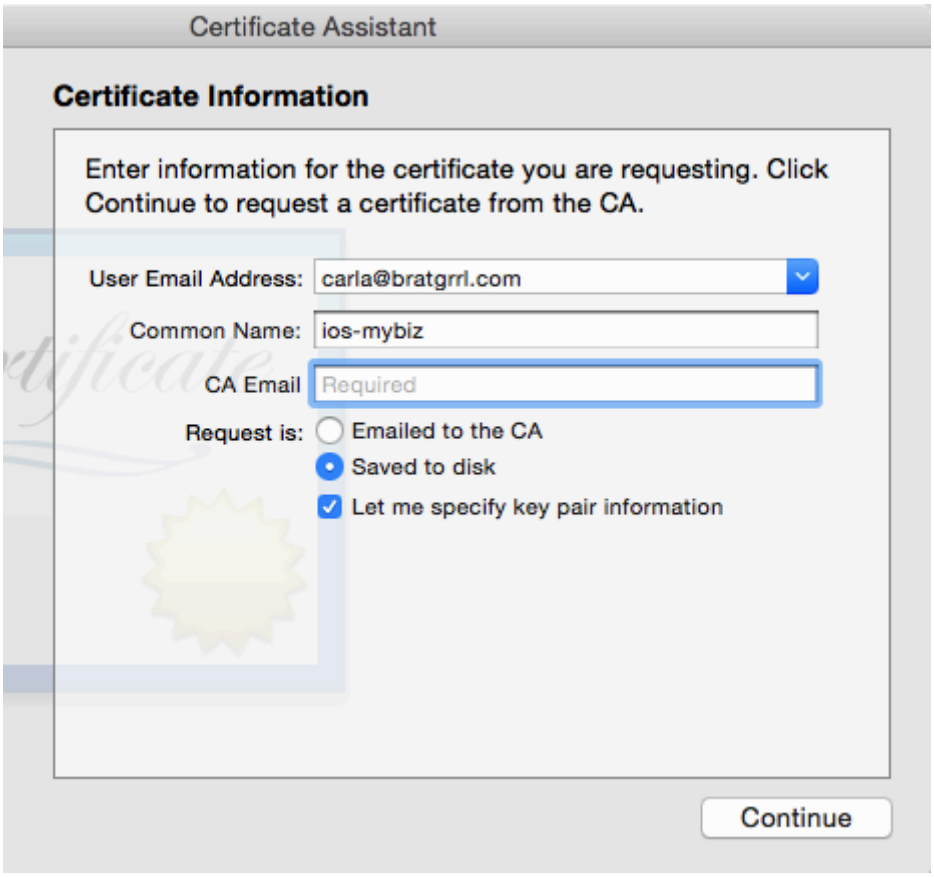


Next, open **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority**.

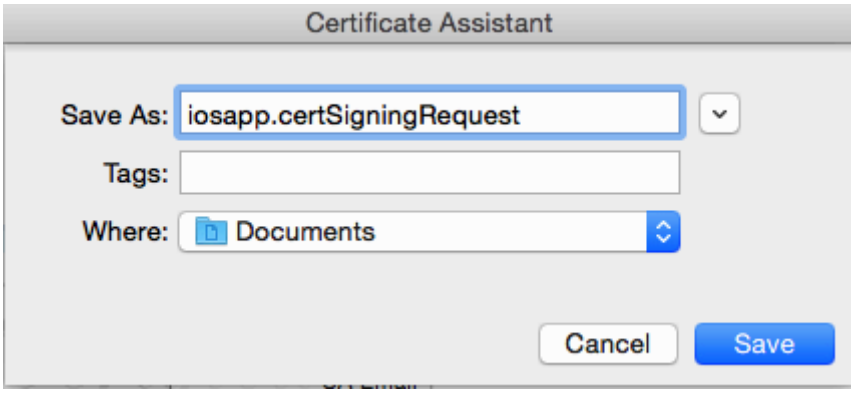


Enter the email address that you use in your Apple developer account, and enter a common name. The common name can be anything you want, for example a helpful descriptive name like "ios-mybiz". Check **[Saved to disk]** and **[Let me specify key pair information]**, then click

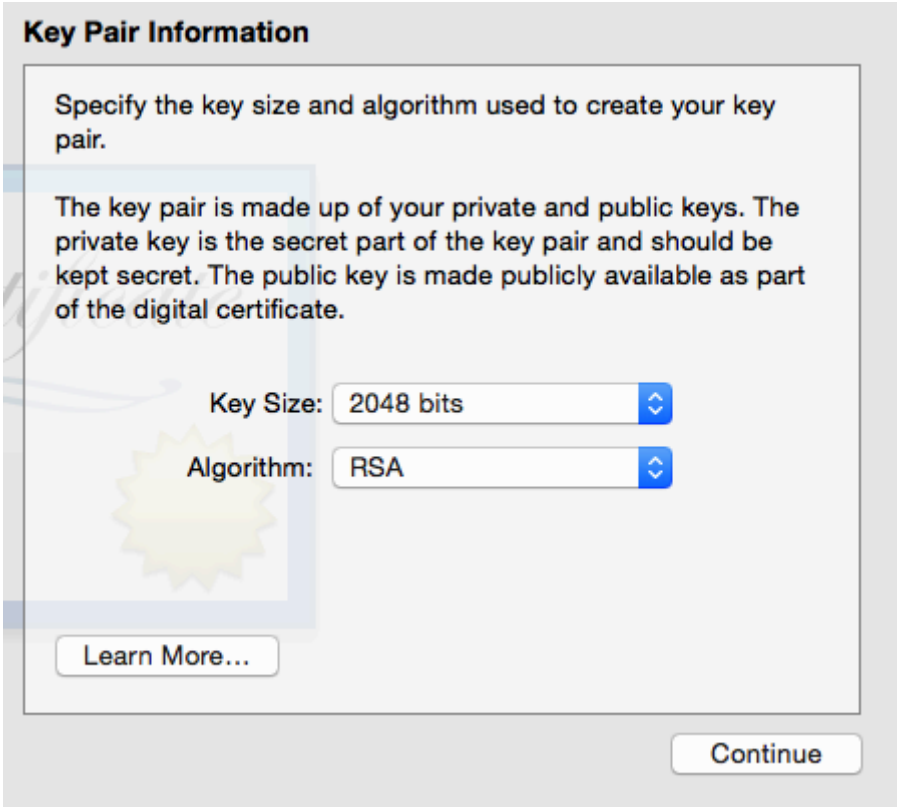
[Continue].



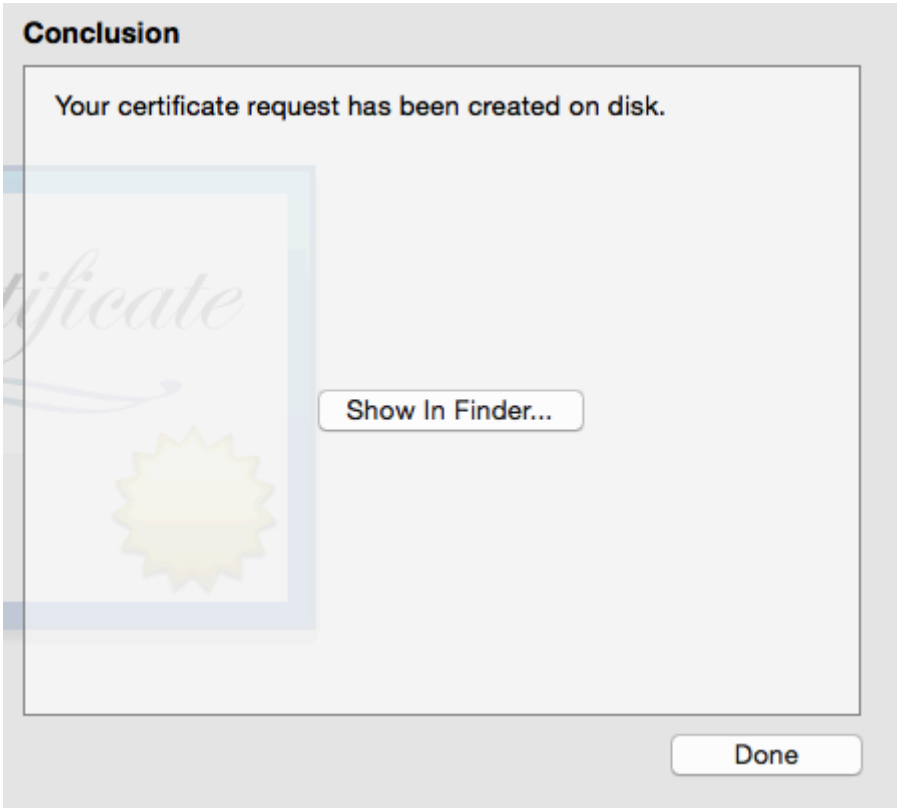
Give your CSR a helpful descriptive name, such as **iosapp.certSigningRequest**, and choose the location to save it on your hard drive, then click **[Save]**.



In the next window, set the **Key Size** value to **2048 bits** and **Algorithm** to **RSA**, and click **[Continue]**. This will create and save your certSigningRequest file (CSR) to your hard drive.

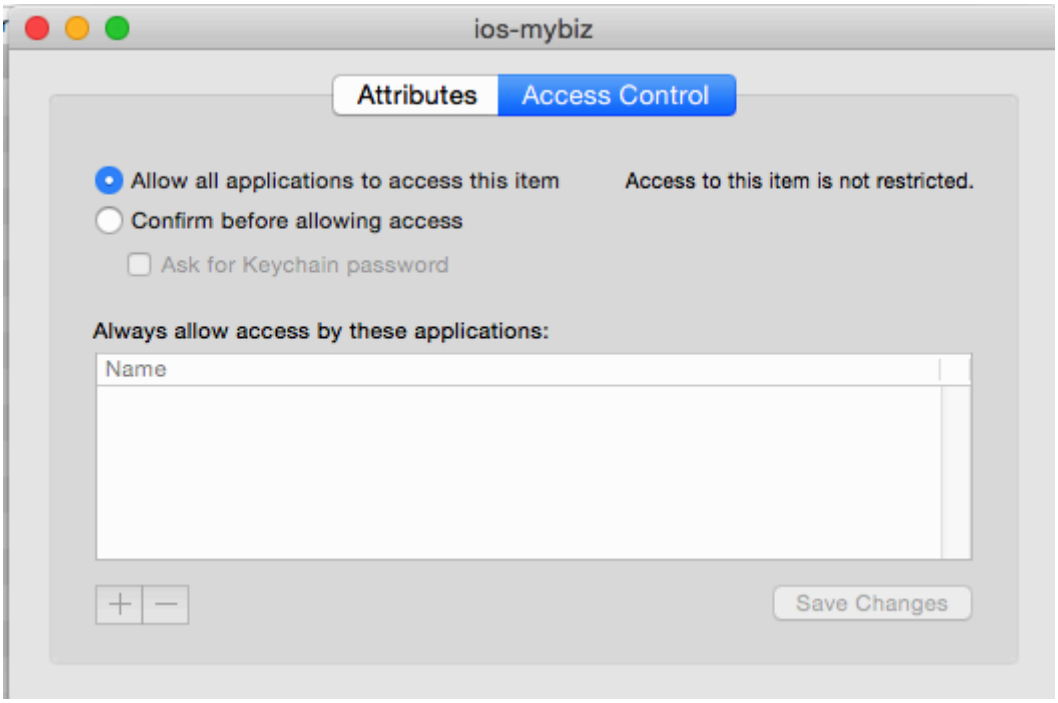


In the next screen your certificate creation is verified. Click a button to view it, or click **[Done]** to go to the next step.

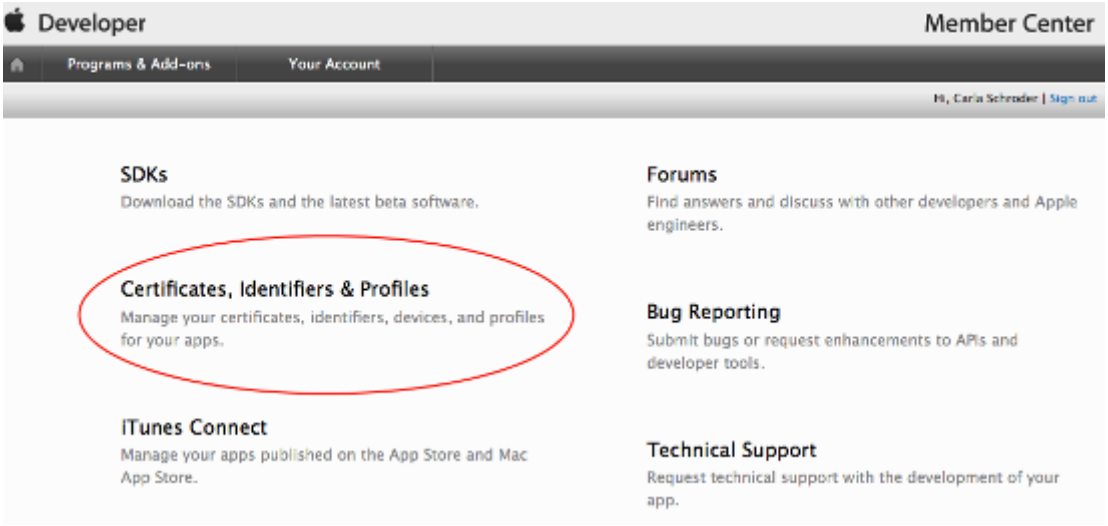


You also get a corresponding public and private key pair, which you can see in the **Login** > **Keys** section of Keychain.

Double-click on your new private key to open the Access Control dialog. Check **[Allow all applications to access this item]**.

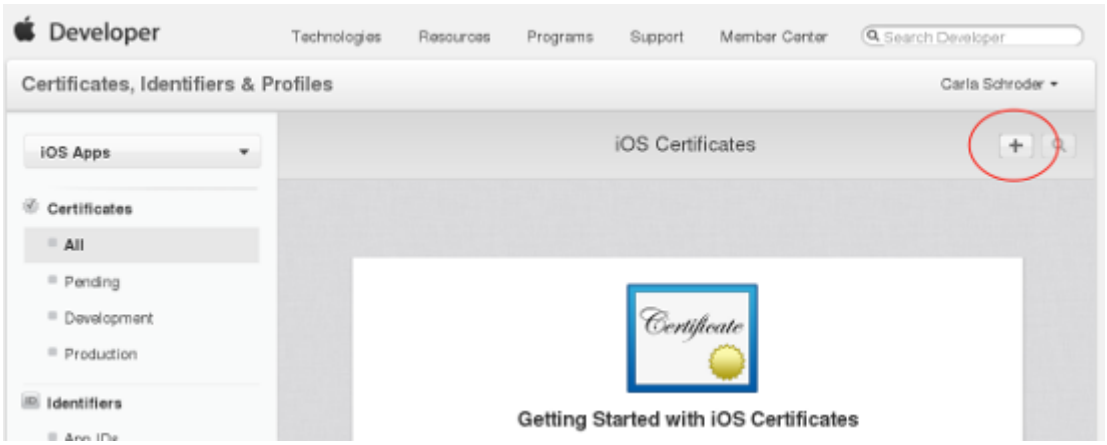


Now login to the **Member Center** on https://developer.apple.com/. Click **[Certificates, Identifiers & Profiles]**.

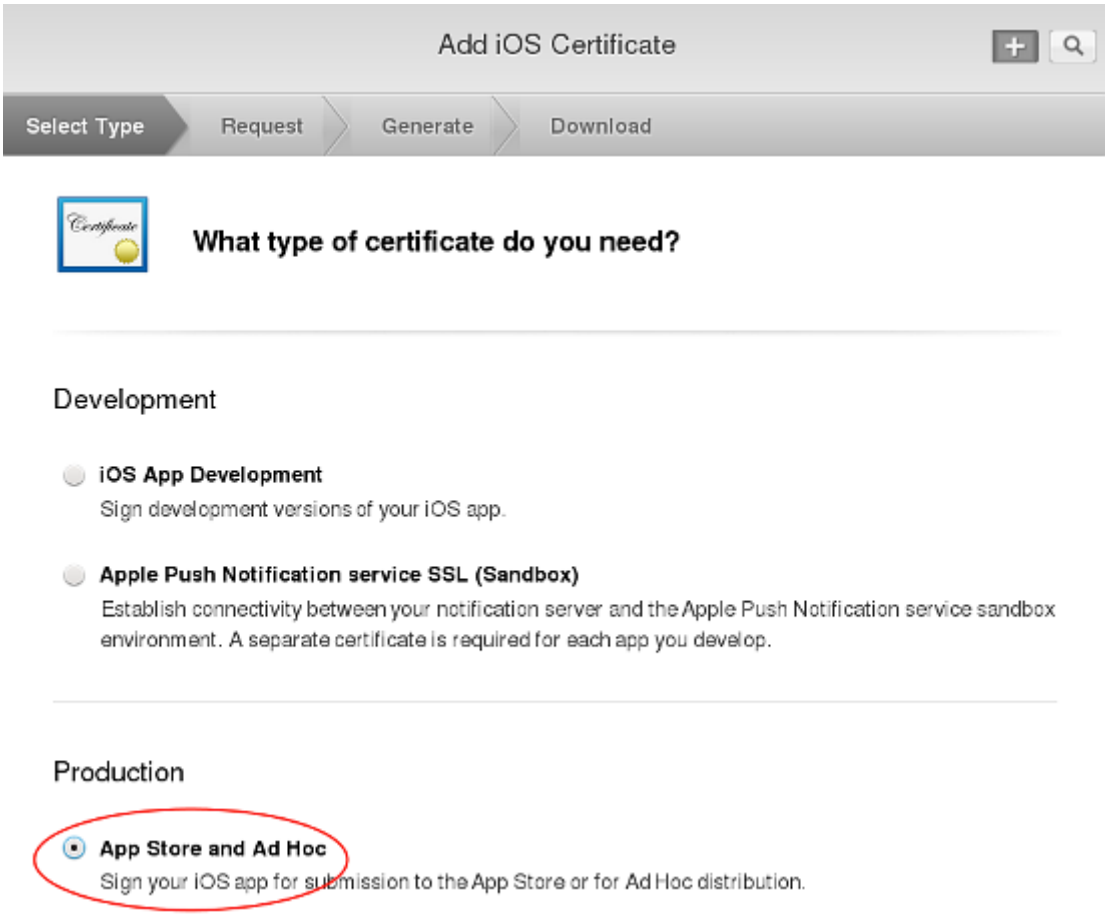


Then click **iOS Apps** > **Certificates**.

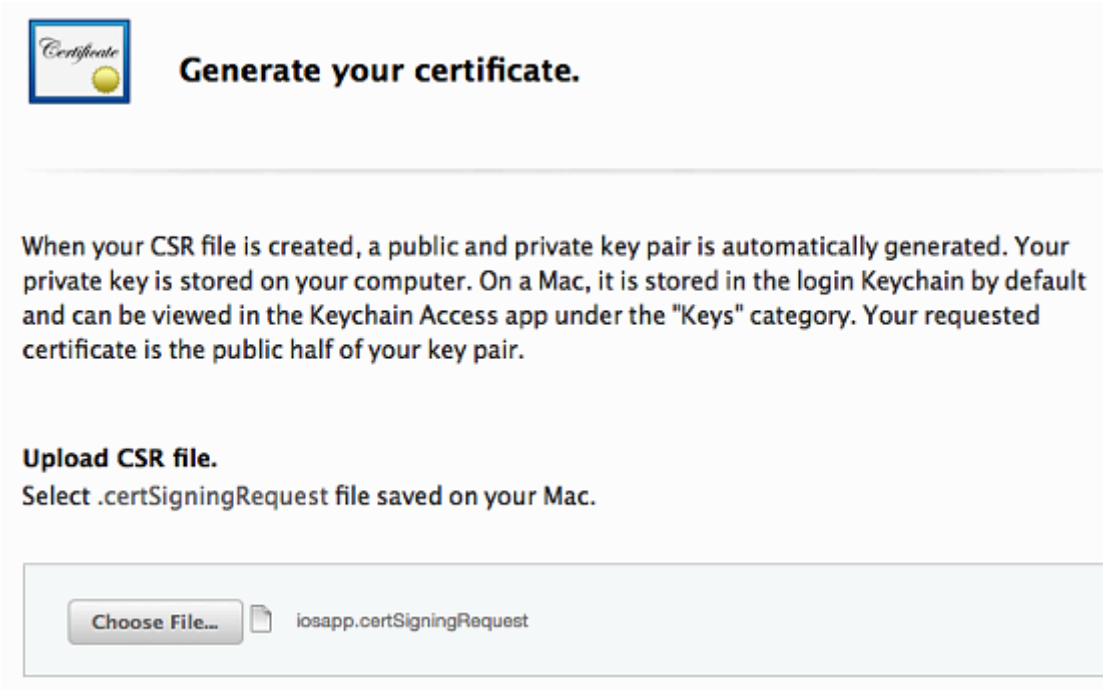
Next, click the **[add]** button (the little plus sign) in the top right corner of the **iOS Certificate** page.



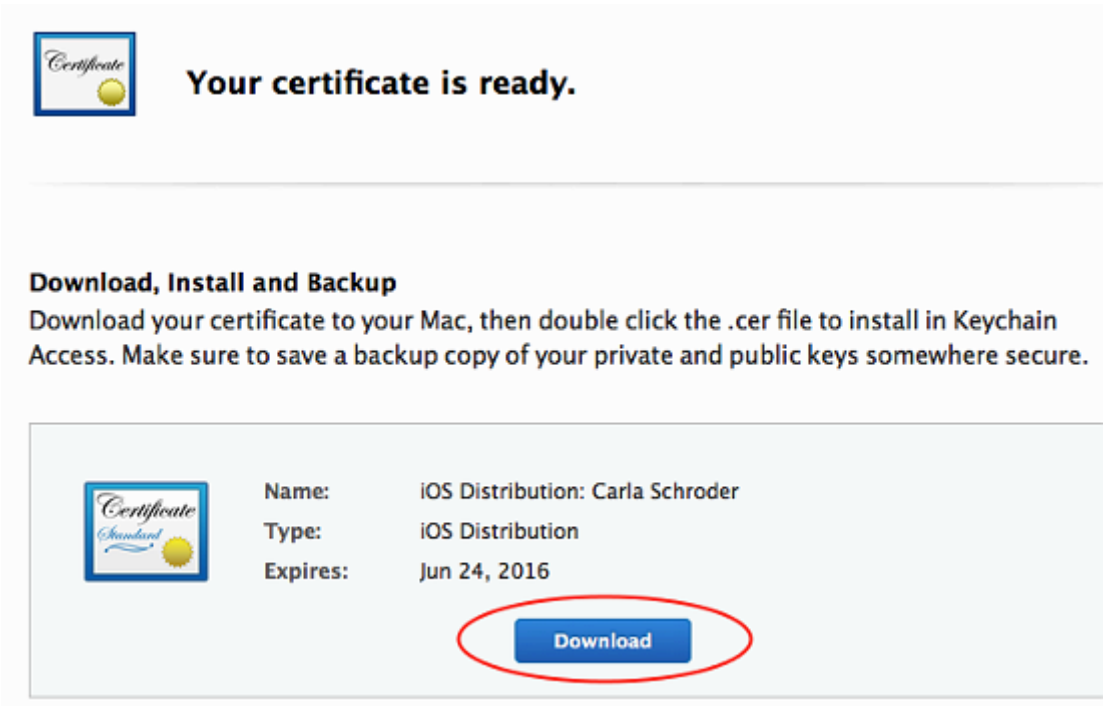
Under "What type of certificate do you need?" check **[App Store and Ad Hoc]**, then click the **[Continue]** button at the bottom of the page.



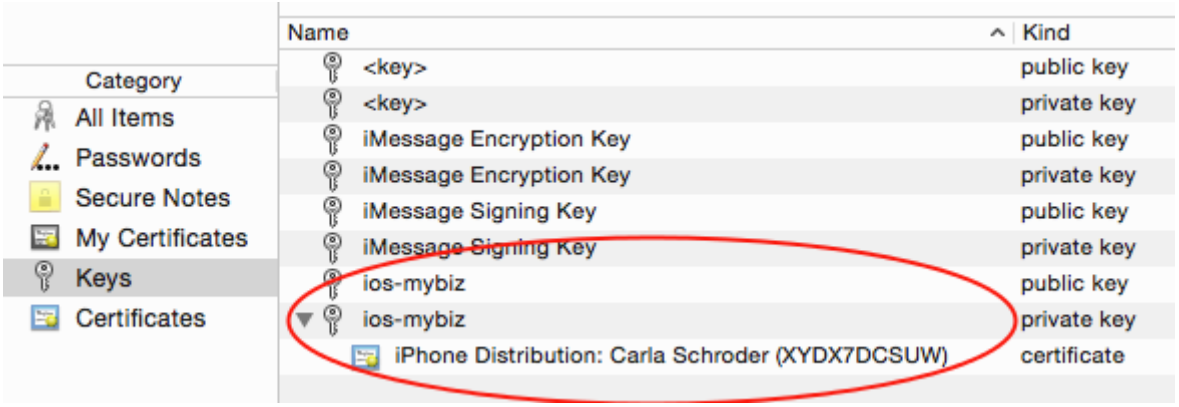
The next screen, **About Creating a Certificate Signing Request (CSR)** has information about creating a CSR in Keychain Access. You already did this, so go to the next screen. "Add iOS Certificate", to upload the CSR you already created, then click the **[Generate]** button.



Your new certificate is named **ios_distribution.cer**. Download it to your Mac; then find it and double-click on it to install it properly in Keychain.



After installing it, you should see it stored with its corresponding private key in Keychain.



Remember to make backups of your keys and certificates and keep them in a safe place.

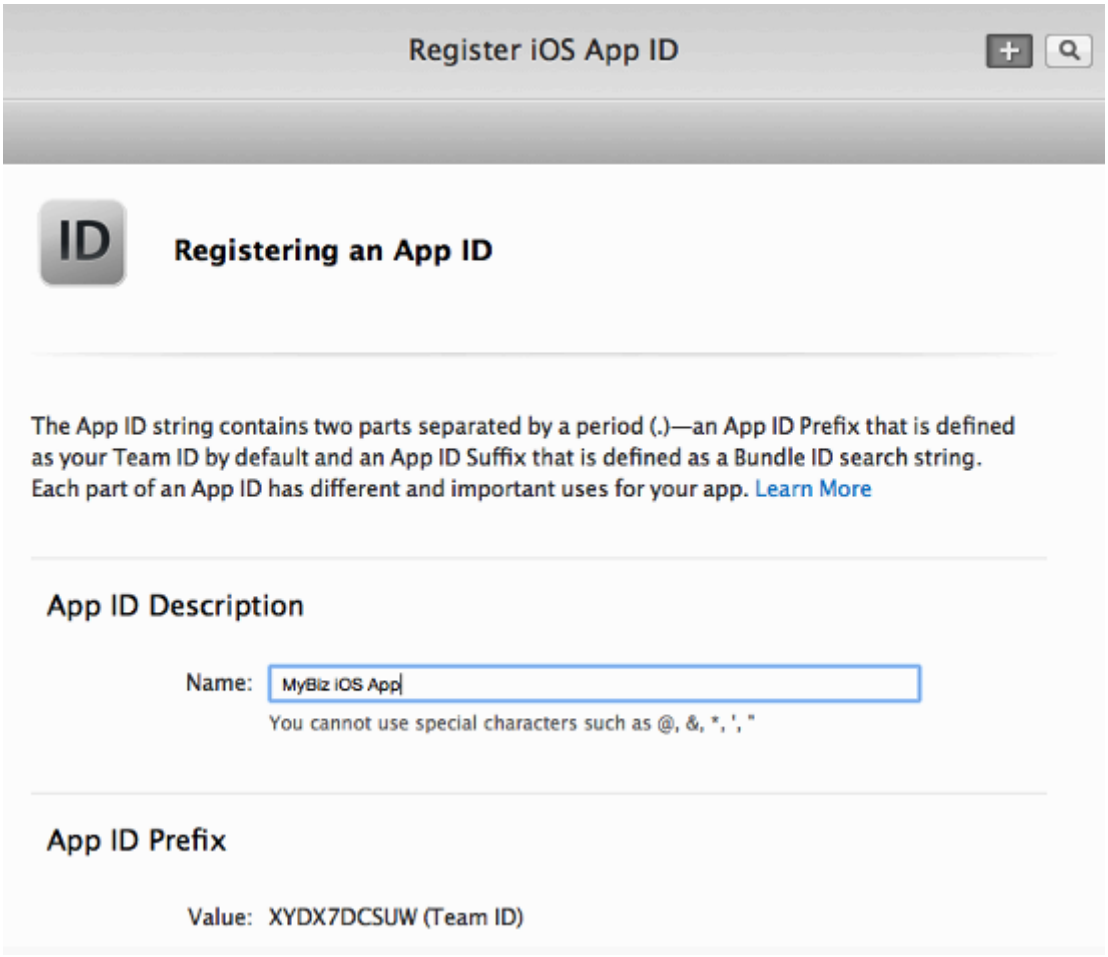
Create Bundle IDs

Create Bundle IDs

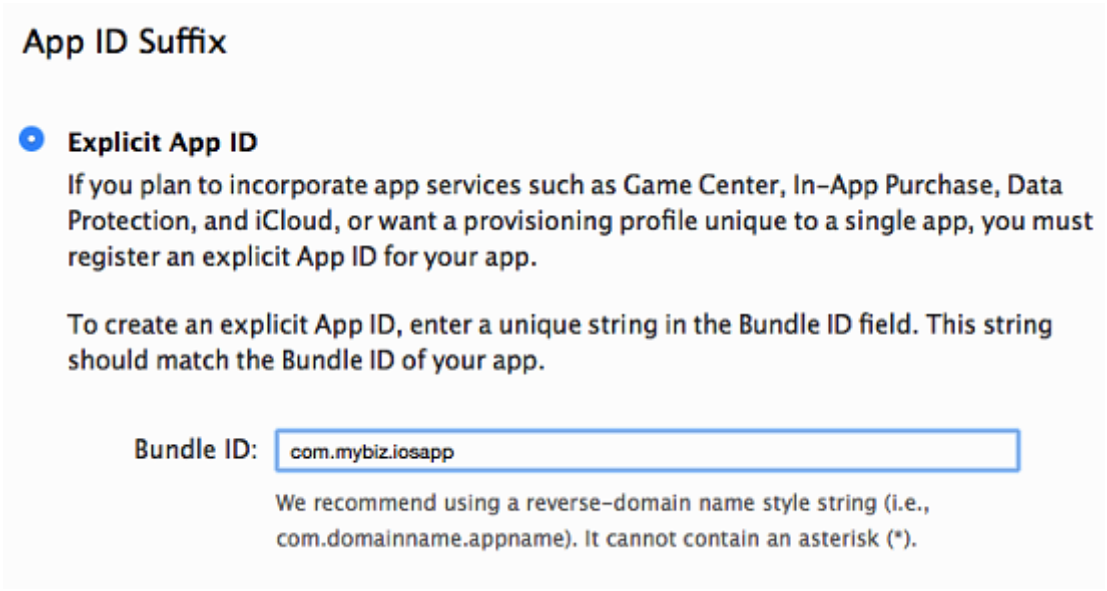
The next step is to create four **Bundle IDs**. These are unique identifiers for your branded iOS app. You must also create an **App Group** and place your three **Bundle IDs** in your **App Group**. You will need your base **Bundle ID** and **App Group** when you build your app with the ownBrander app on customer.owncloud.com.

Create App ID

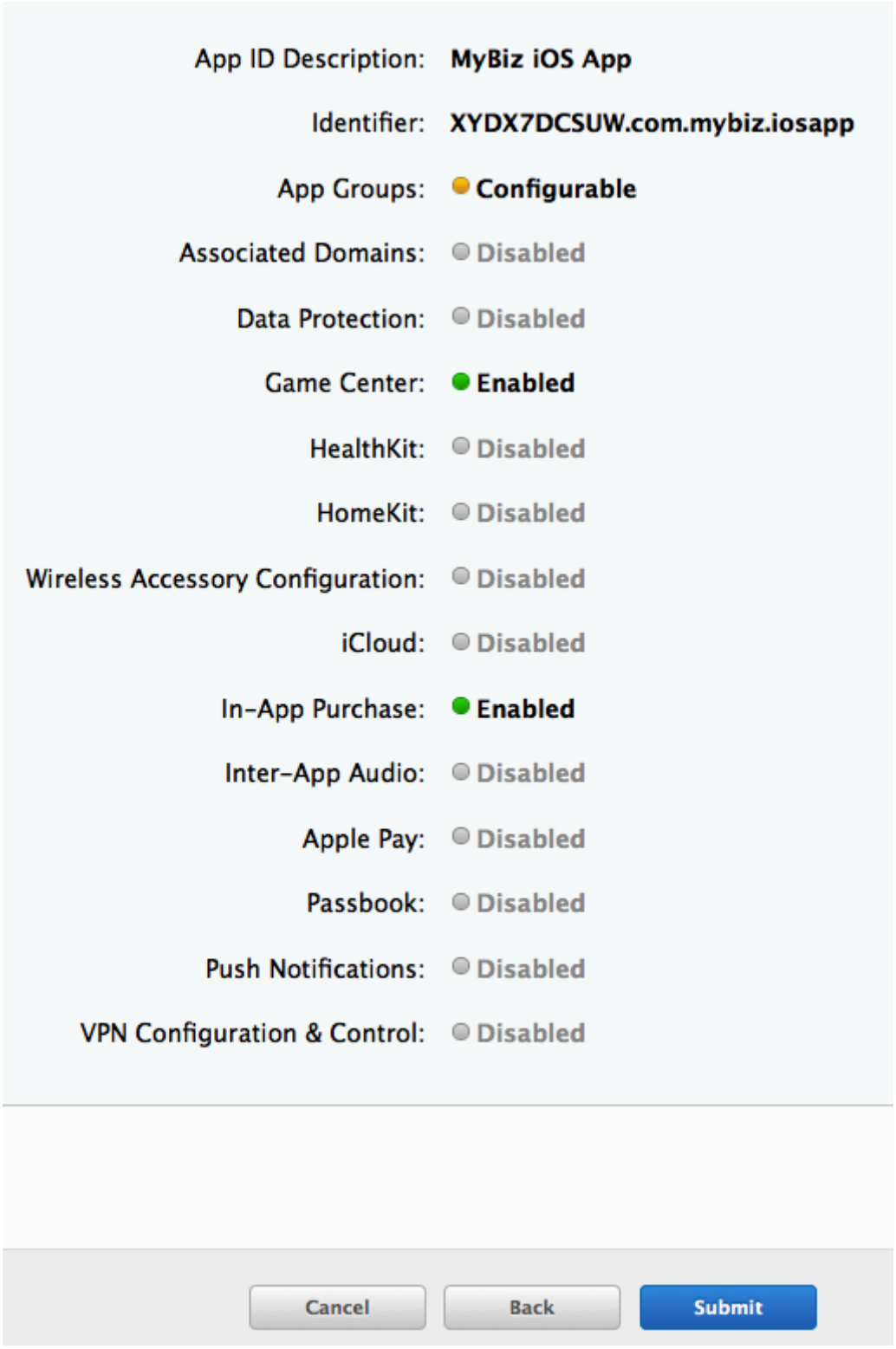
Now you must create your App ID. Go to **Identifiers** > **App IDs** and click the **[plus button]** (top right) to open the "Register iOS App ID" screen. Fill in your **App ID Description**, which is anything you want, so make it helpful and descriptive. The **App ID Prefix** is your Apple Developer Team ID, and is automatically entered for you.



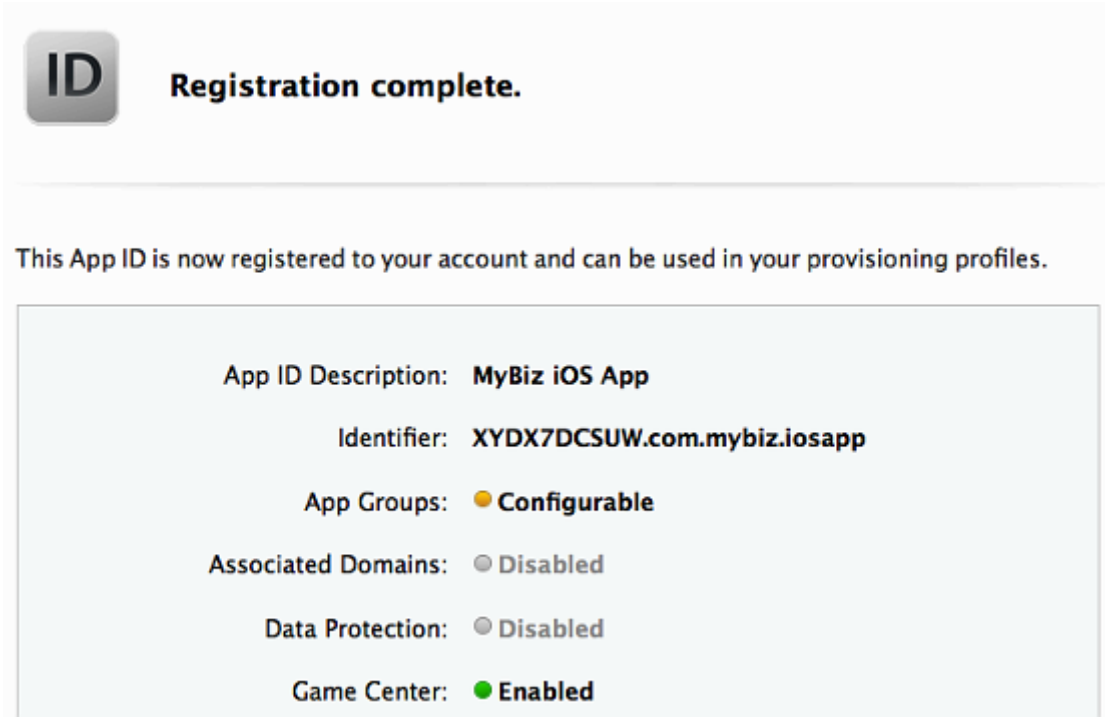
Scroll down to the **App ID Suffix** section and create your **Bundle ID**. Your **Bundle ID** is the unique identifier for your app. Make a note of it because you will need it as you continue through this process. The format for your **Bundle ID** is reverse-domain, e.g. *com.MyCompany.MyProductName*.



The next section, **App Services**, is where you select the services you want enabled in your app. You can edit this anytime after you finish creating your **App ID**. Check **App Groups**, make your other selections and then click the **[Continue]** button at the bottom. Now you can confirm all of your information. If everything is correct click **[Submit]**; if you need to make changes use the **[Back]** button.

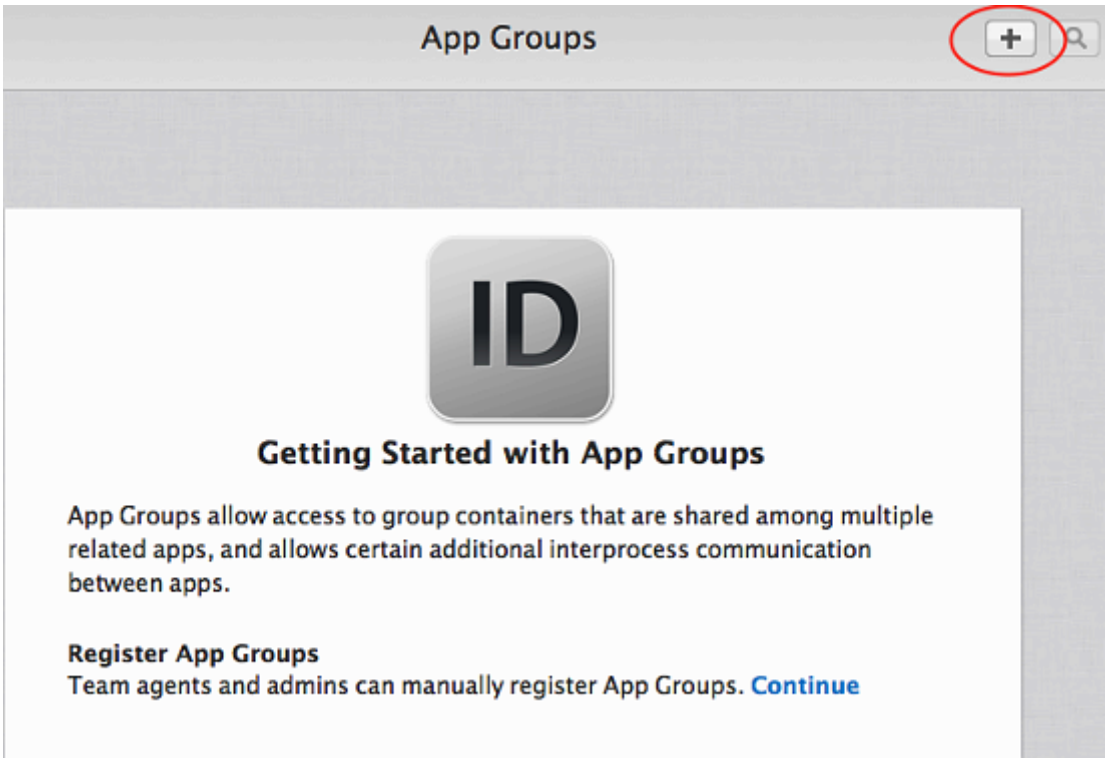


When you are finished you will see a confirmation. Click the **[Done]** button at the bottom.

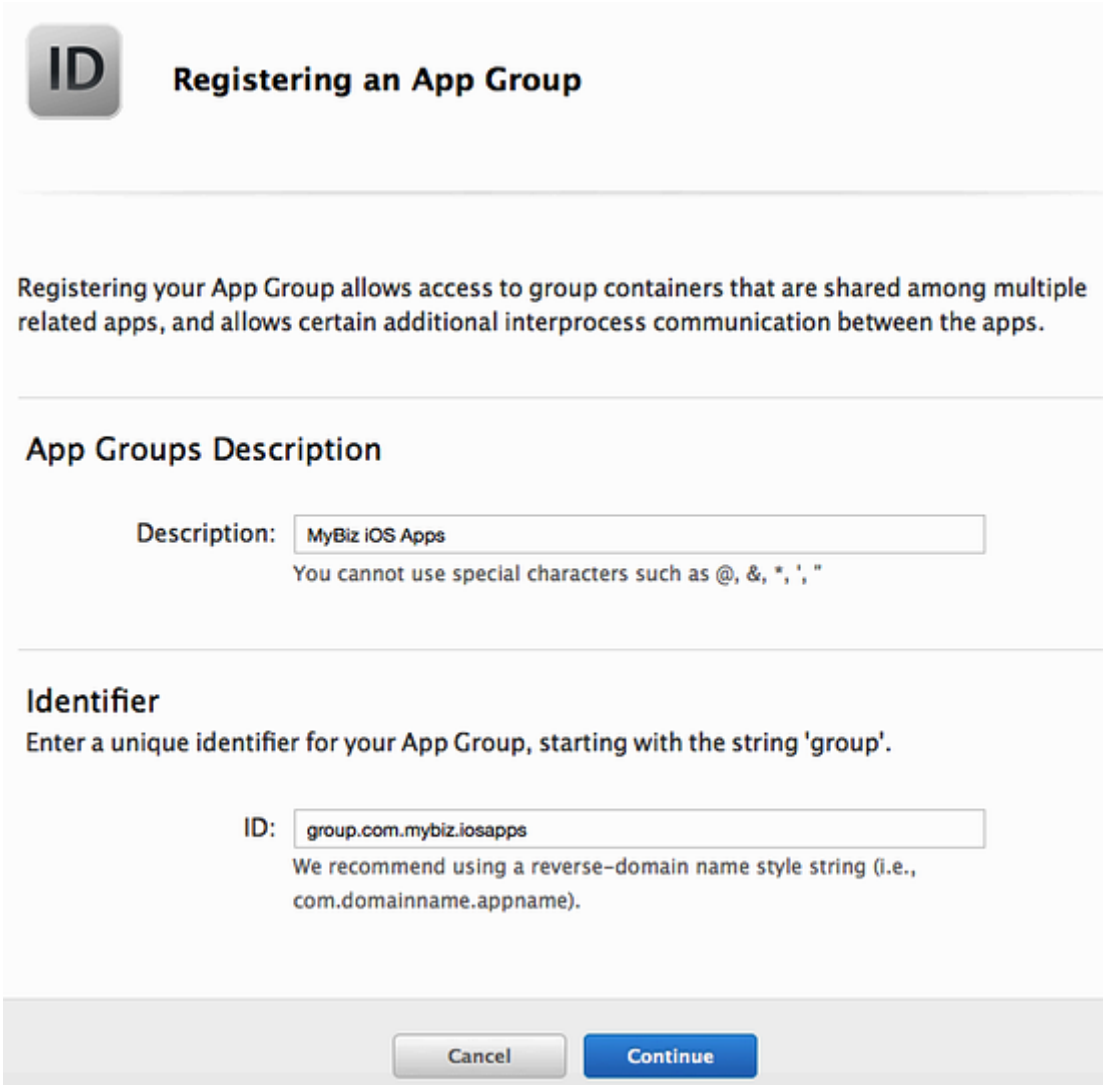


Create App Group

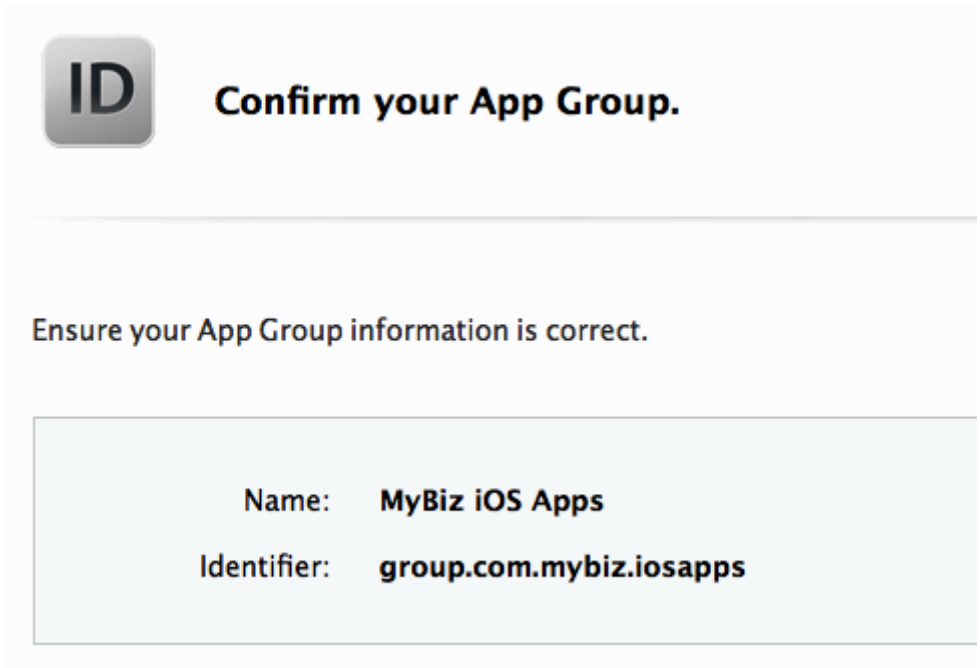
The next step is to create an App Group and put your App ID in it. Go to **Identifiers** > **App Groups** and click the **[plus button]** (top right).



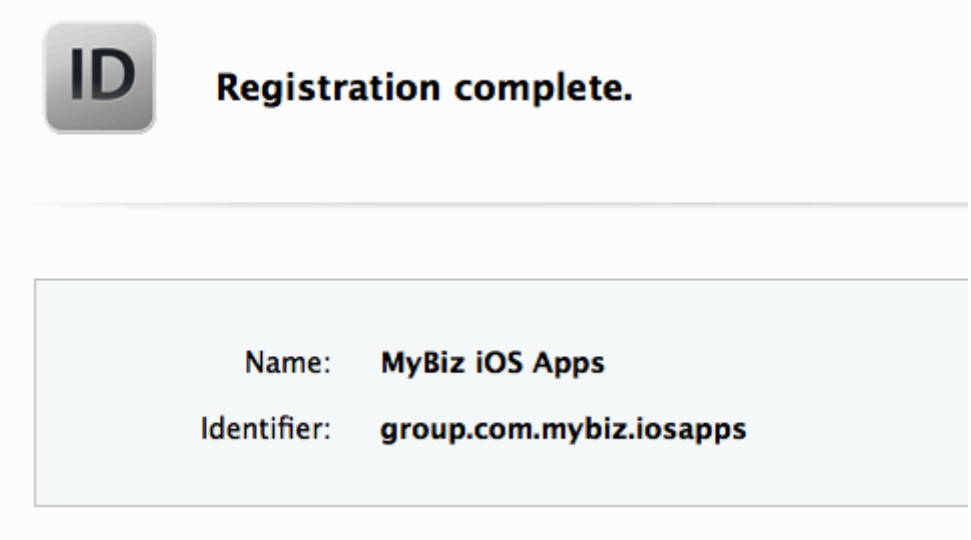
Create a description for your app group, and a unique identifier in the format *group.com.MyCompany.MyAppGroup*. Then click **[Continue]**



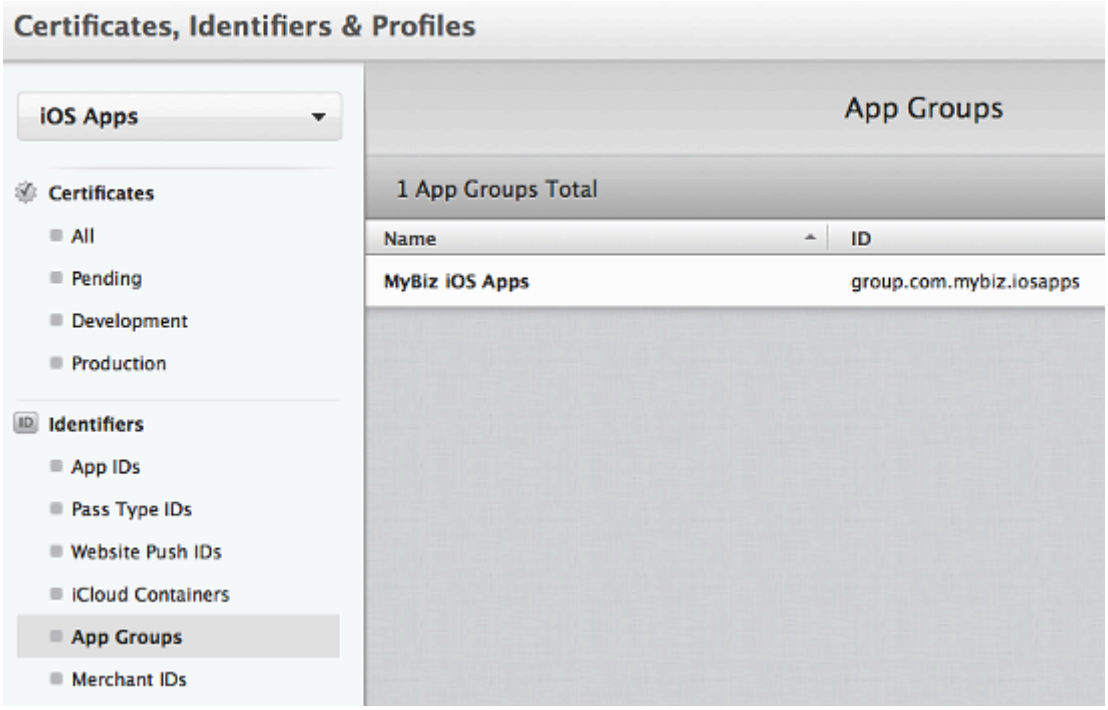
Review the confirmation screen, and if everything looks correct click the **[Register]** button.



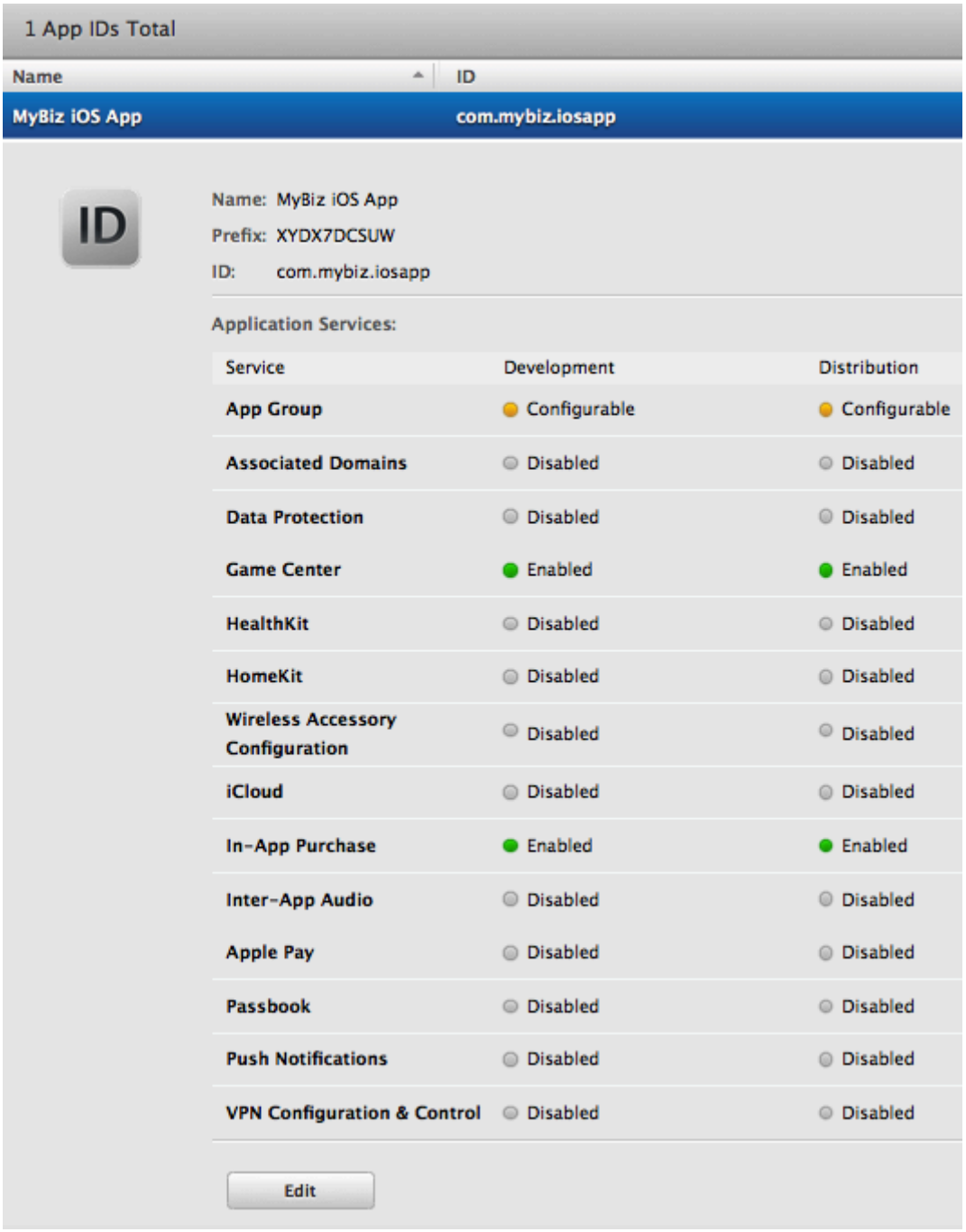
You'll see a final confirmation screen; click **[Done]**.



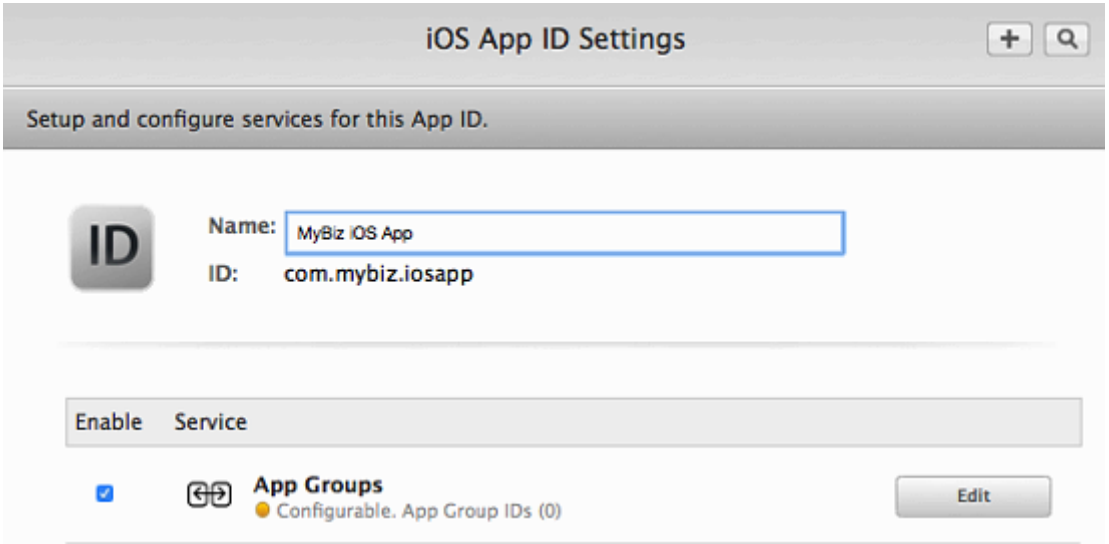
When you click on **[App Groups]** you will see your new app group.



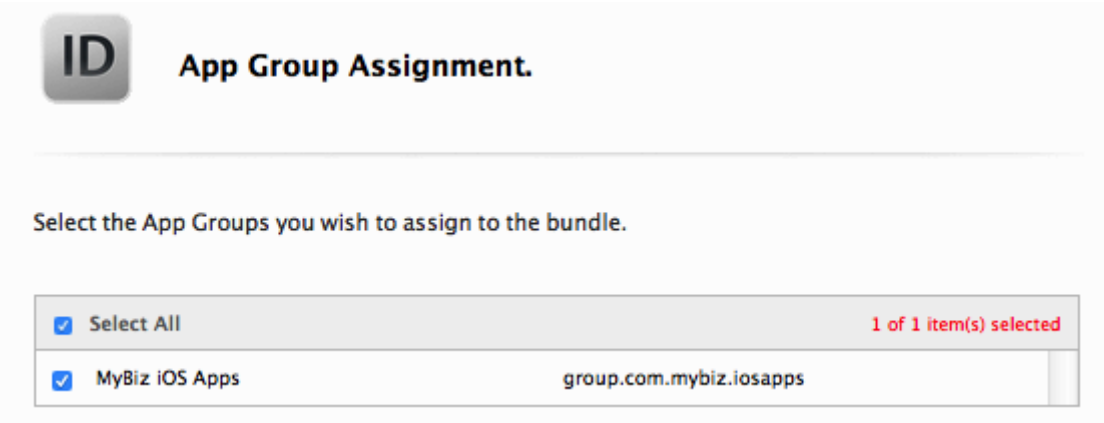
Now go back to **Identifiers** > **App IDs** and click on your **[App ID]**. This opens a screen that displays all your app information. Click the **[Edit]** button at the bottom.



Click the **[Edit]** button next to **[App Groups]**.



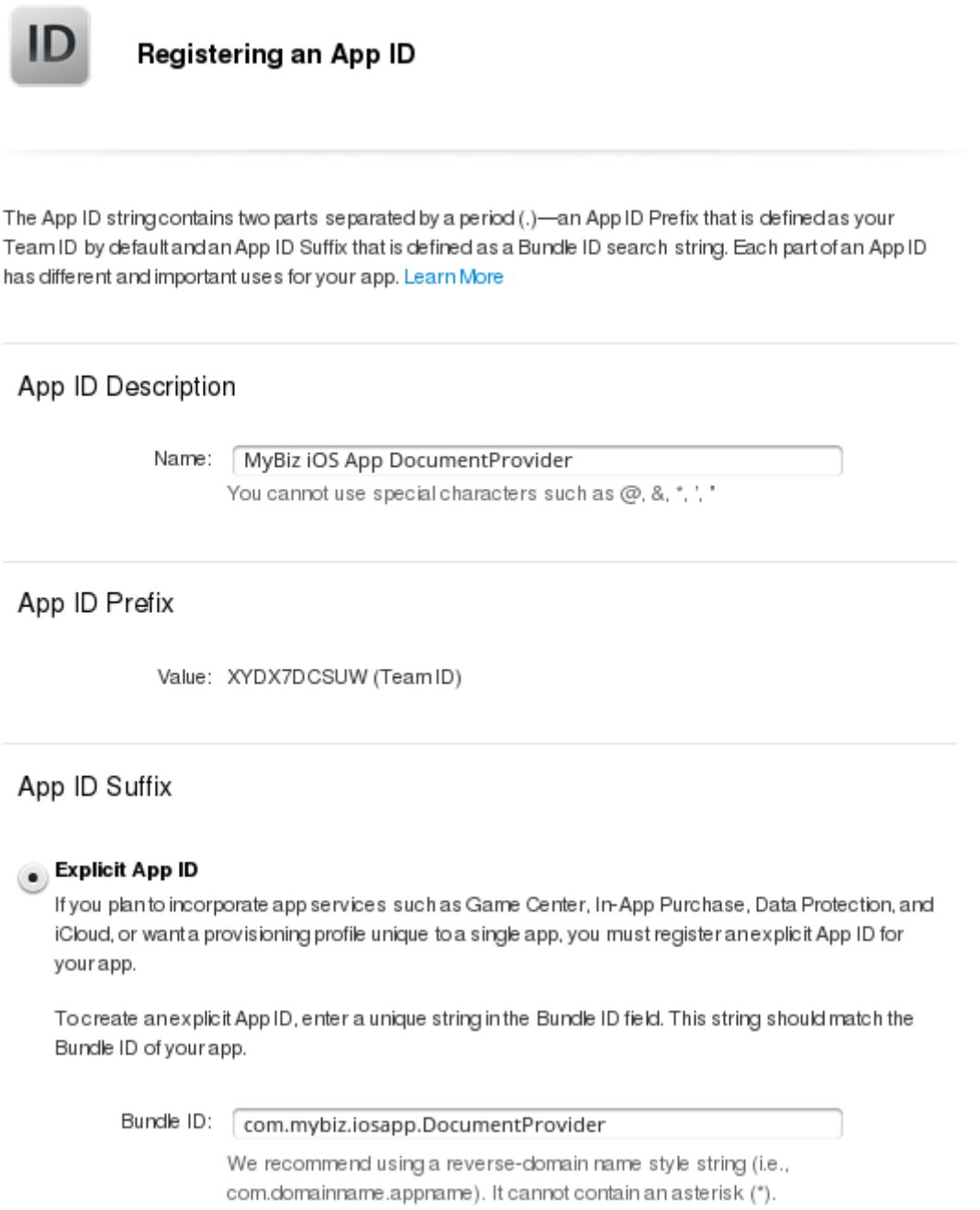
Check your app and click the **[Continue]** button.



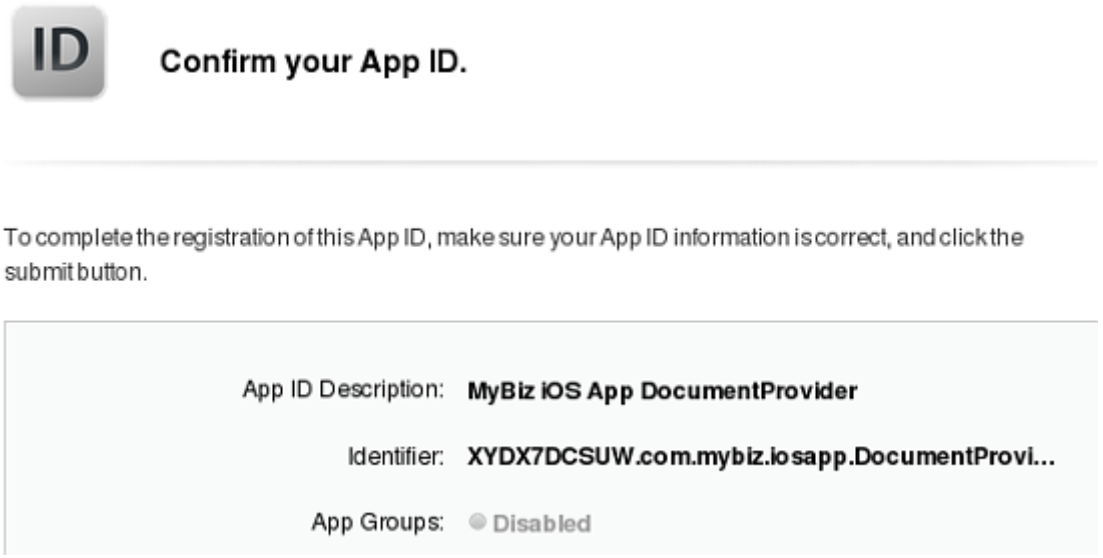
The next screen asks you to "Review and confirm the App Groups you have selected". Click the **[Assign]** button to confirm. The next screen announces "You have successfully updated the App Groups associations with your App ID", and you must click yet another button, the **[Done]** button at the bottom.

Create a DocumentProvider Bundle ID

Now you must return to **Identifiers** > **App IDs** and click the **[plus button]** to create a DocumentProvider Bundle ID. Follow the same naming conventions as for your App ID, then click **[Continue]**.

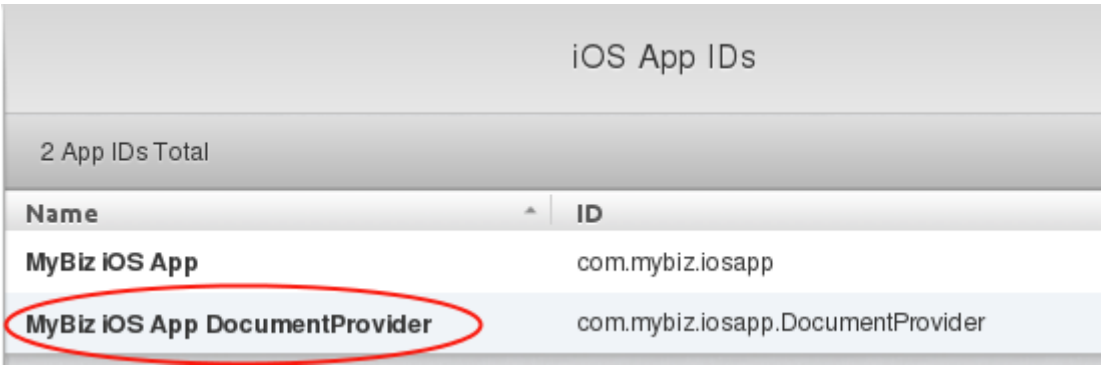


Confirm your new App ID and click **[Submit]**.

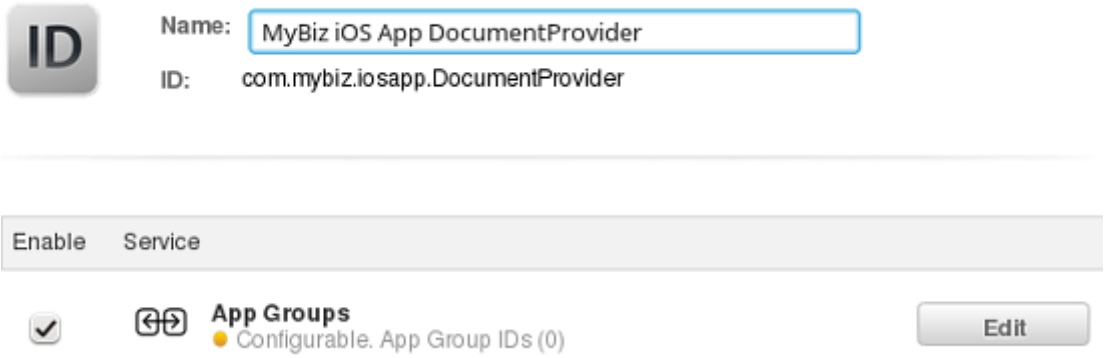


You will see one more confirmation: "Registration complete. This App ID is now registered to your account and can be used in your provisioning profiles." Click **[Done]**.

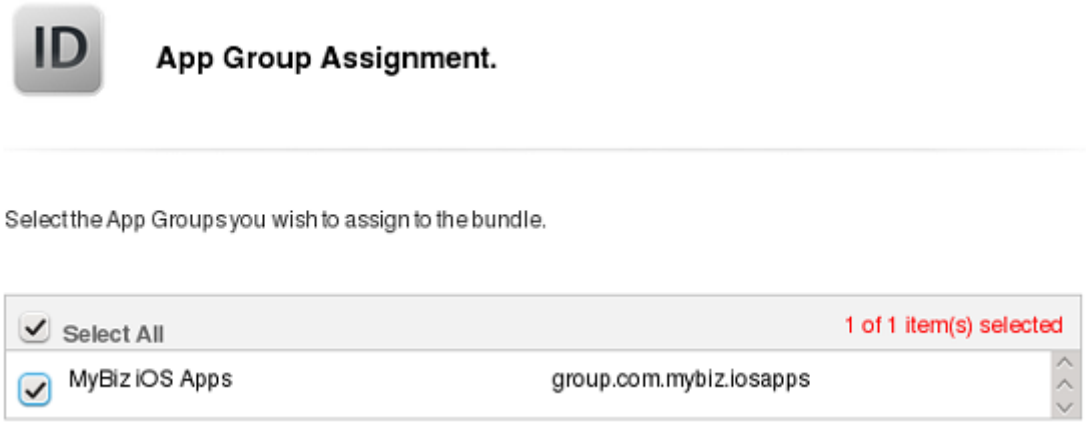
Now you need to add it to your App Group. Go to **Identifiers** > **App IDs** and click on your new **[DocumentProvider Bundle ID]** to open its configuration window, and then click the **[Edit]** button at the bottom.



Select **[App Groups]** and click the **[Edit button]**.



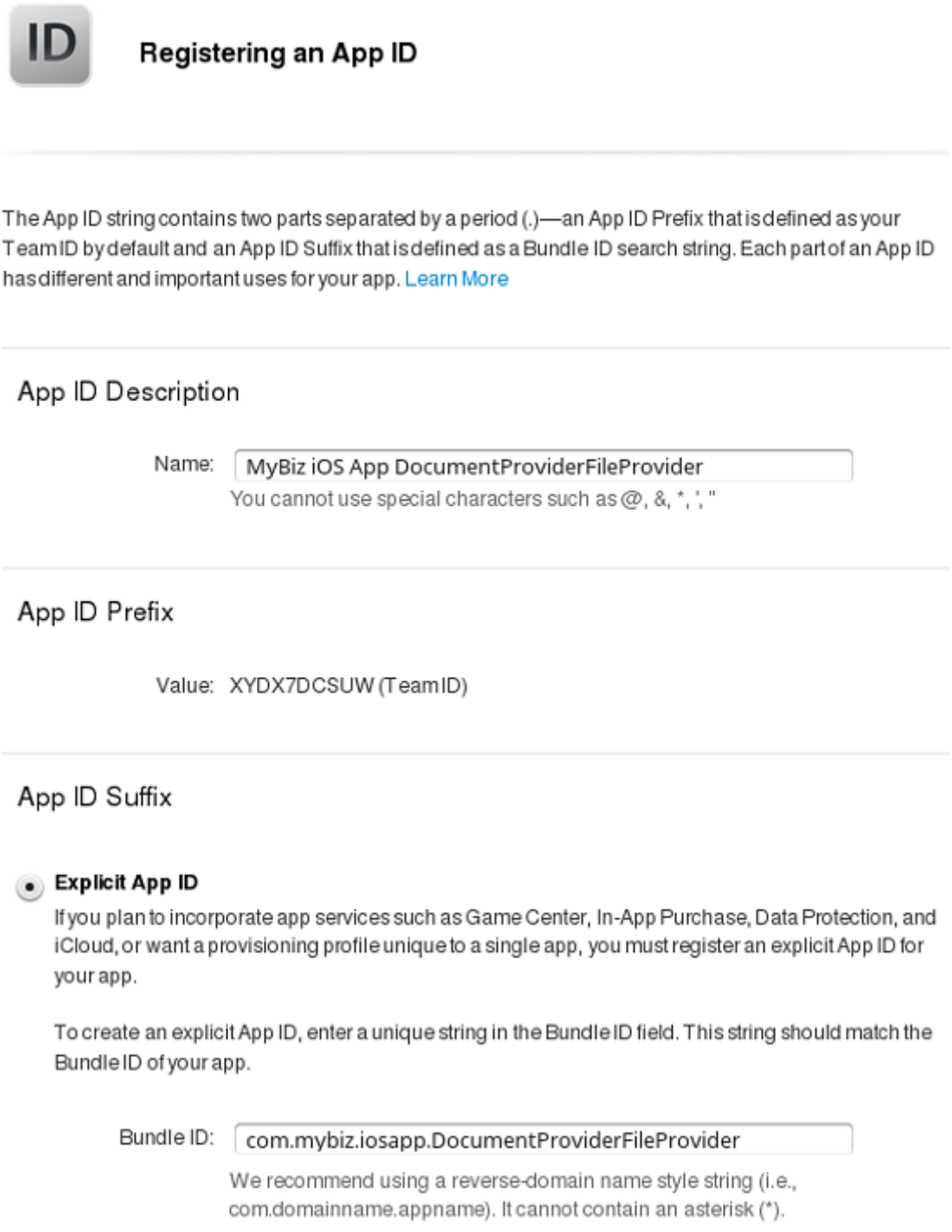
Select your group and click **[Continue]**.



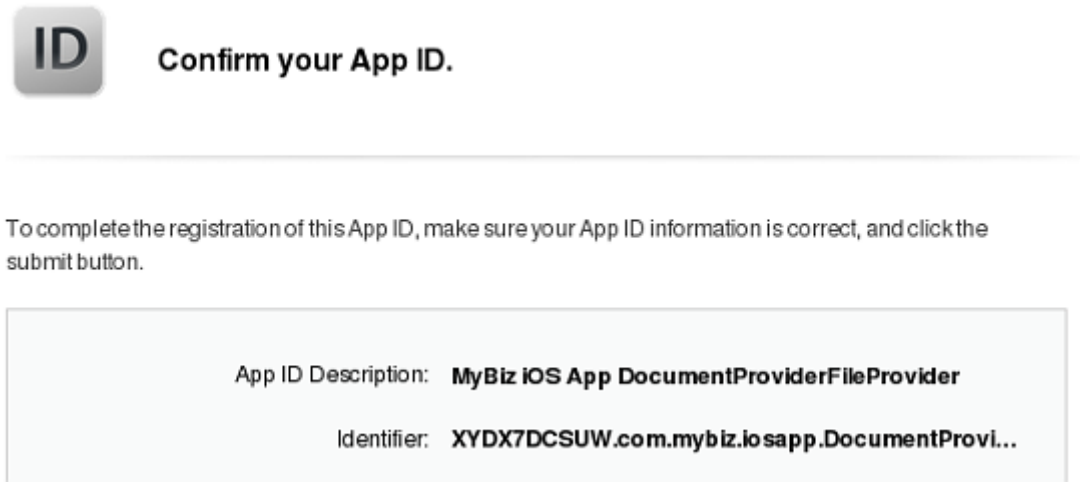
Once again you will asked if you really mean it. On the confirmation screen click **[Assign]**, and you'll see the message "You have successfully updated the App Groups associations with your App ID."

Create a DocumentProviderFileProvider Bundle ID

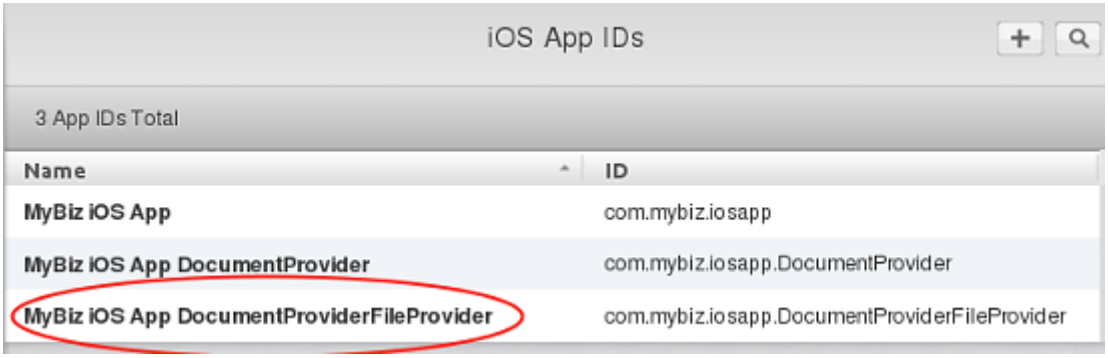
One more time, go to **Identifiers** > **App IDs** and click the **[plus button]** to create a DocumentProviderFileProvider Bundle ID. Follow the same naming conventions as for your App ID, then click **[Continue]**.



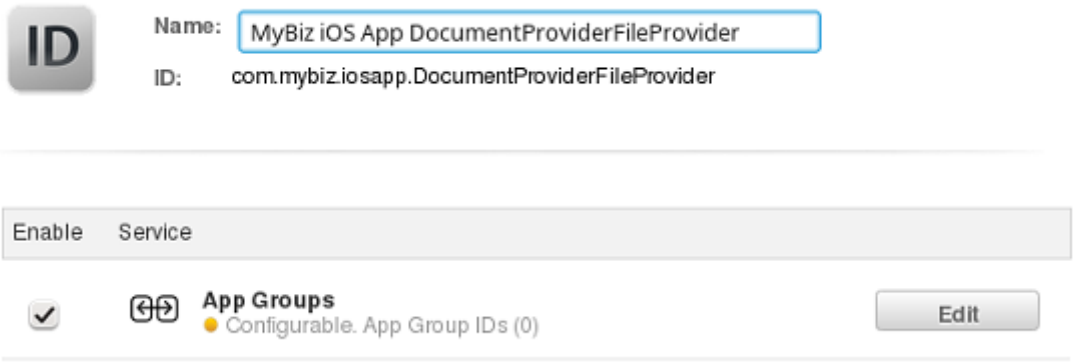
Confirm your new App ID and click **[Submit]**.



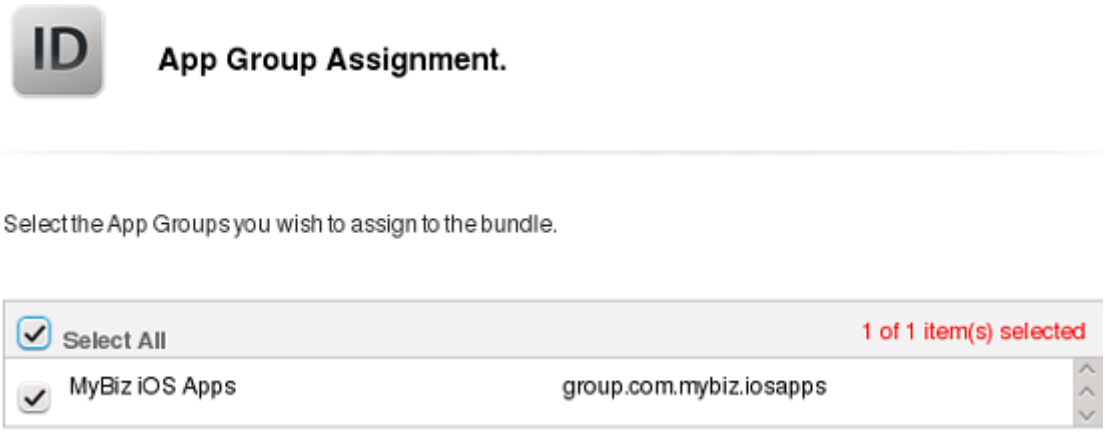
You will see one more confirmation; review it and click **[Done]**. Now you need to add it to your App Group. Go to **Identifiers** > **App IDs** and click on your new **[DocumentProviderFileProvider Bundle ID]** to open its configuration window, and then click the **[Edit]** button.



Select **[App Groups]** and click the **[Edit]** button.



Select your group and click **[Continue]**.

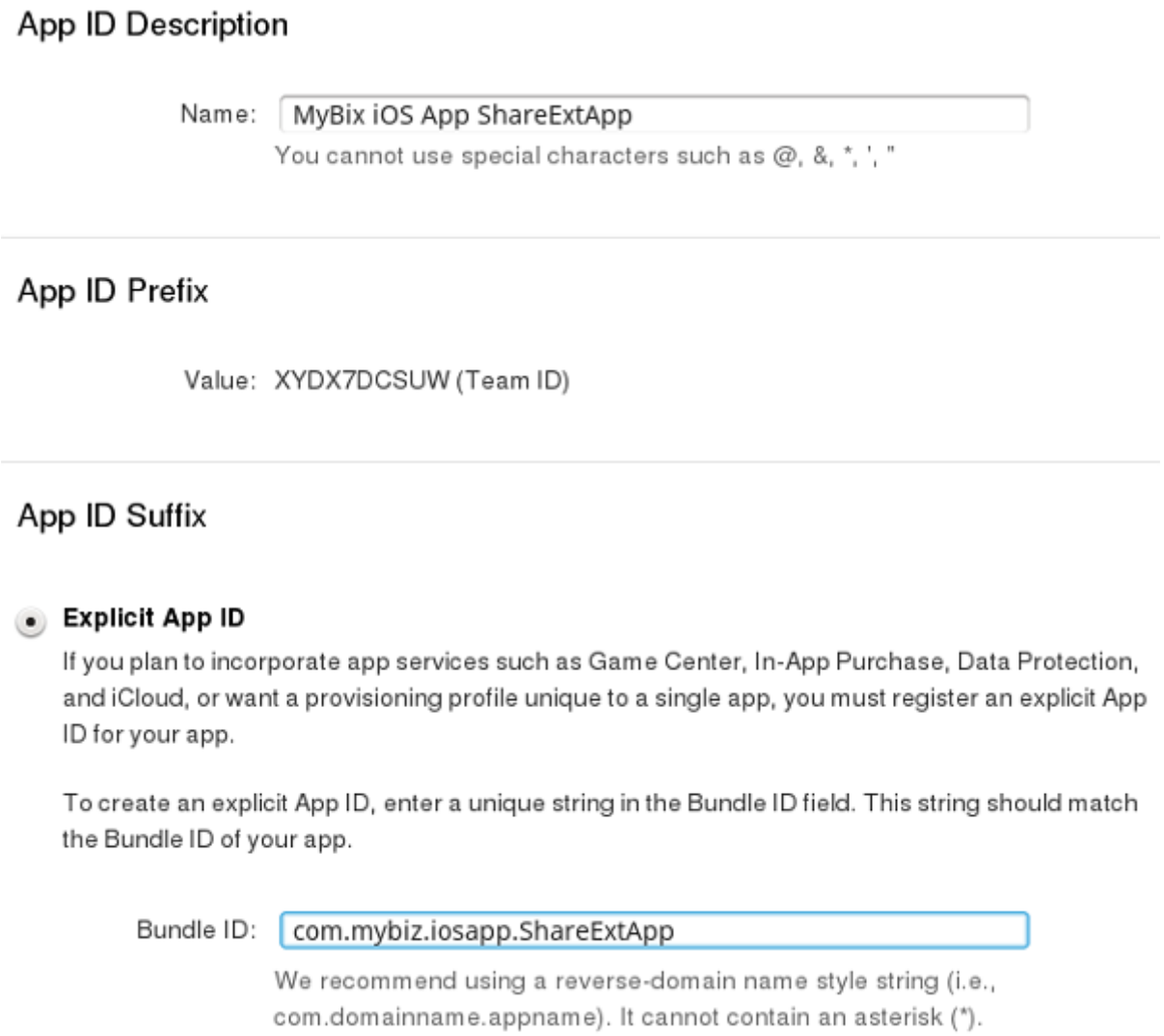


On the confirmation screen click **[Assign]**, and you'll see the message "You have successfully updated the App Groups associations with your App ID."

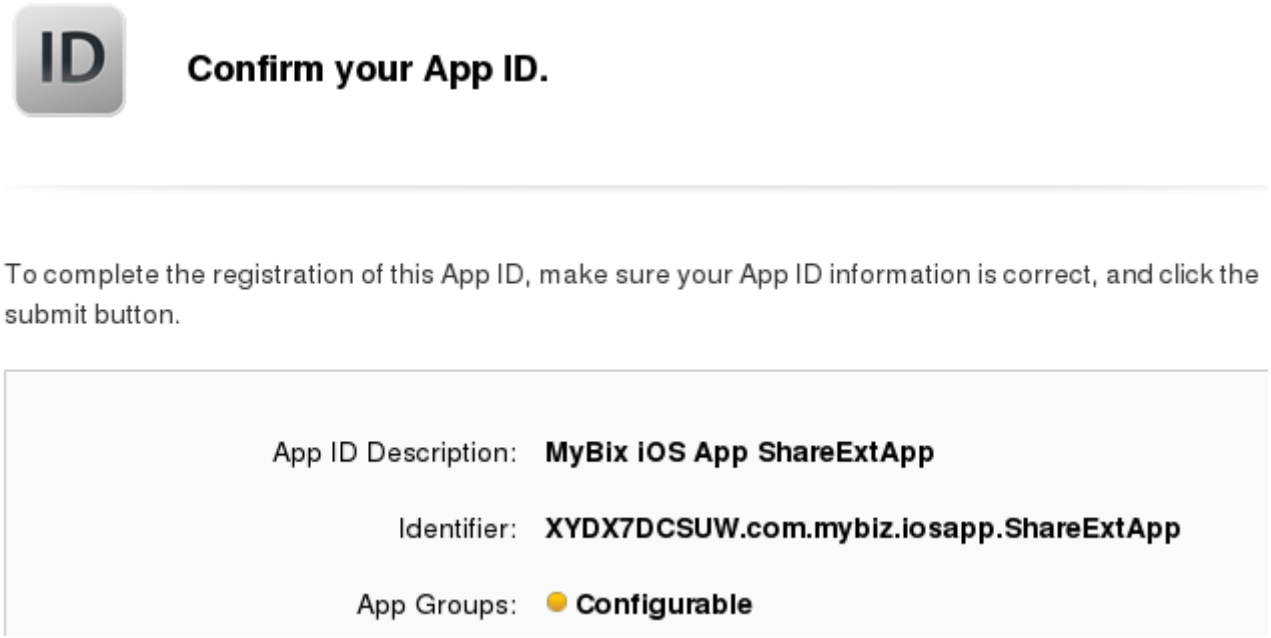
Create a ShareExtApp Bundle ID

This supports Apple's ShareIN extension.

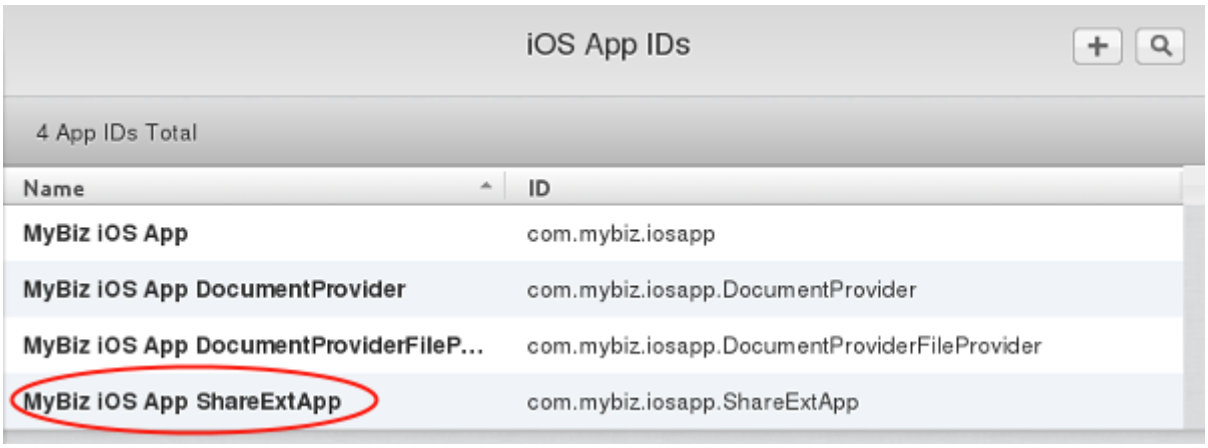
Yet again, go to **Identifiers** > **App IDs** and click the **[plus button]** to create a ShareExtApp Bundle ID. Follow the same naming conventions as for your App ID, then click **[Continue]**.



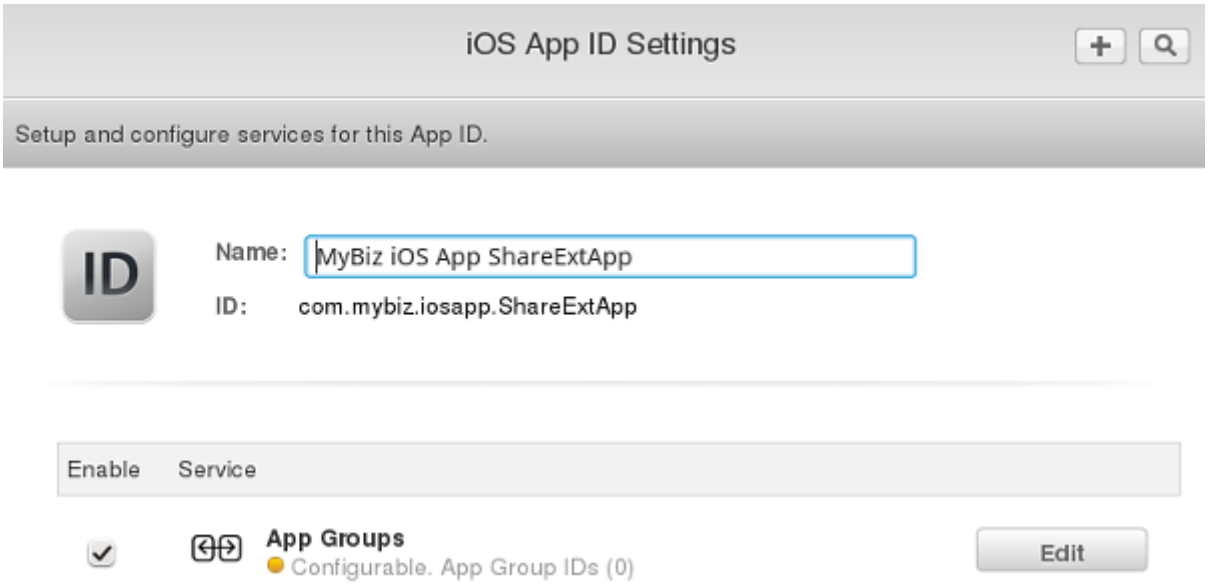
Confirm your new App ID and click **[Submit]**.



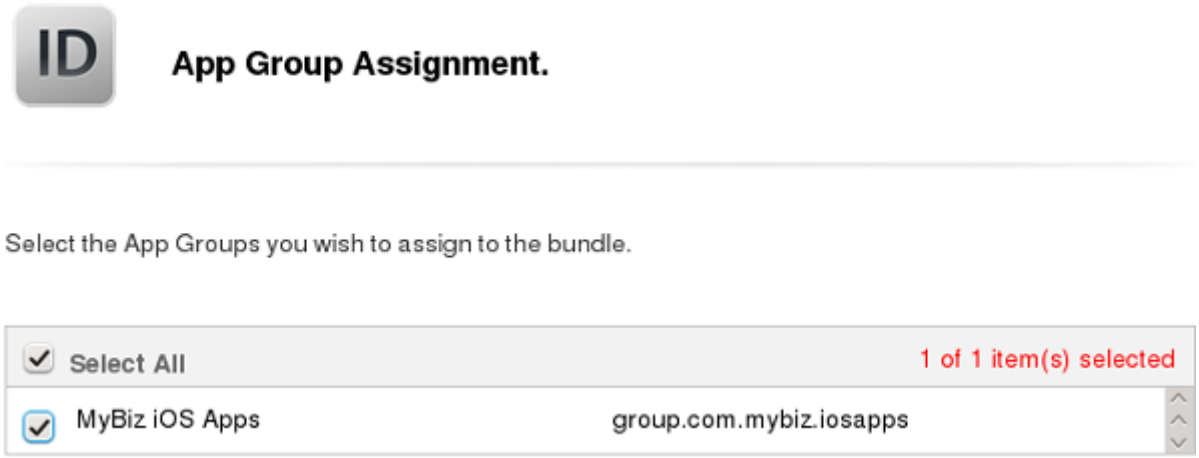
You will see one more confirmation; review it and click **[Done]**. Now you need to add it to your App Group. Go to **Identifiers** > **App IDs** and click on your new **[ShareExtApp Bundle ID]** to open its configuration window, and then click the **[Edit]** button.



Select **[App Groups]** and click the **[Edit]** button.



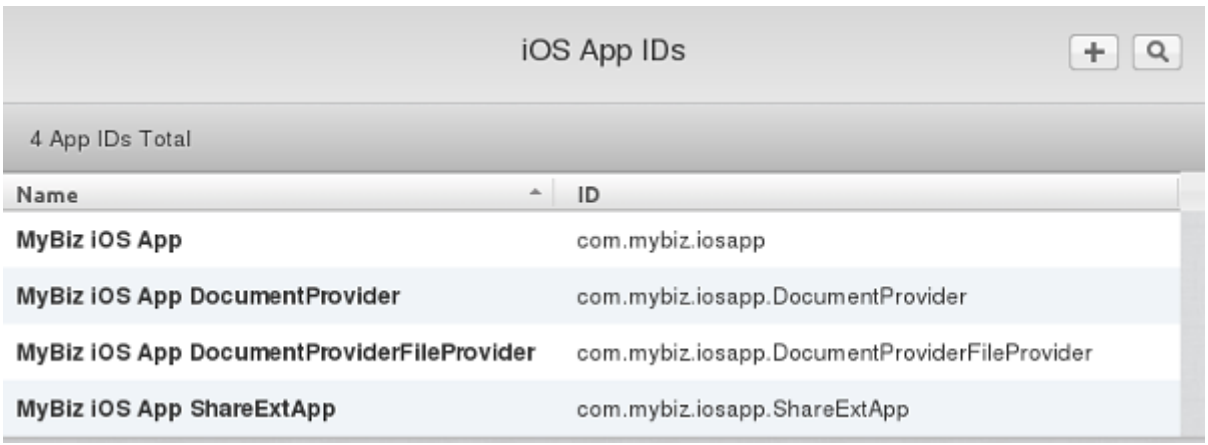
Select your group and click **[Continue]**.



On the confirmation screen click **[Assign]**, and you'll see the message "You have successfully updated the App Groups associations with your App ID."

Four Completed App IDs

Now you should have four new App IDs, and all of them should belong to your App Group.



Setting up Testing Devices

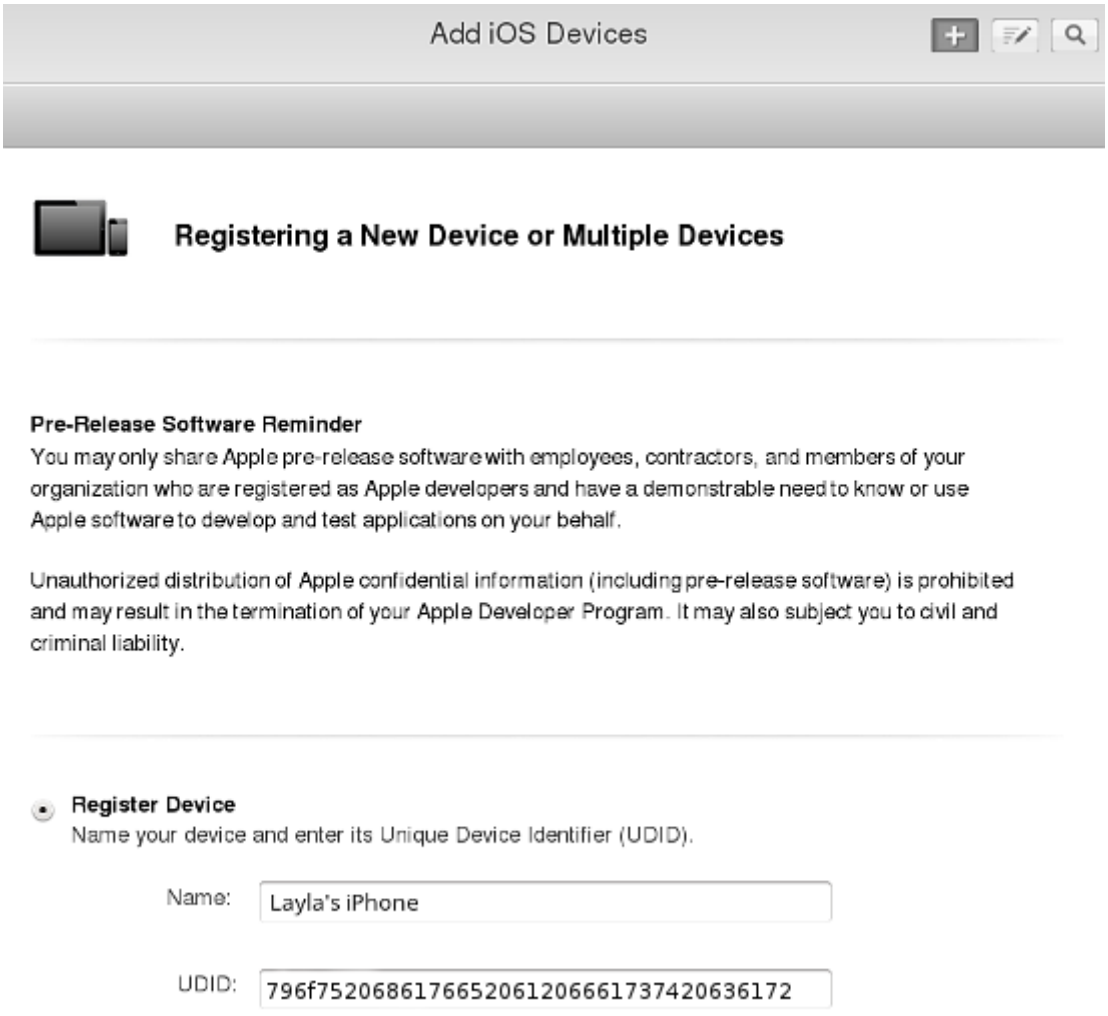
The $99 Apple Developer account allows you to test your iOS apps on a maximum of 100 devices of each type:

```
Apple TV     100
Apple Watch  100
iPad         100
iPhone       100
iPod Touch   100
```

And you must register the UDID of each device in your Apple developer account. If you have the $299 Enterprise account then you can install your app on any device without registering it.

The easiest way to find UDIDs is to connect to your iTunes account. Then connect your iOS device to your Mac computer. Your device will appear on the left sidebar in iTunes. Click on this to display your device information. Then click on the serial number, and you will see your UDID.

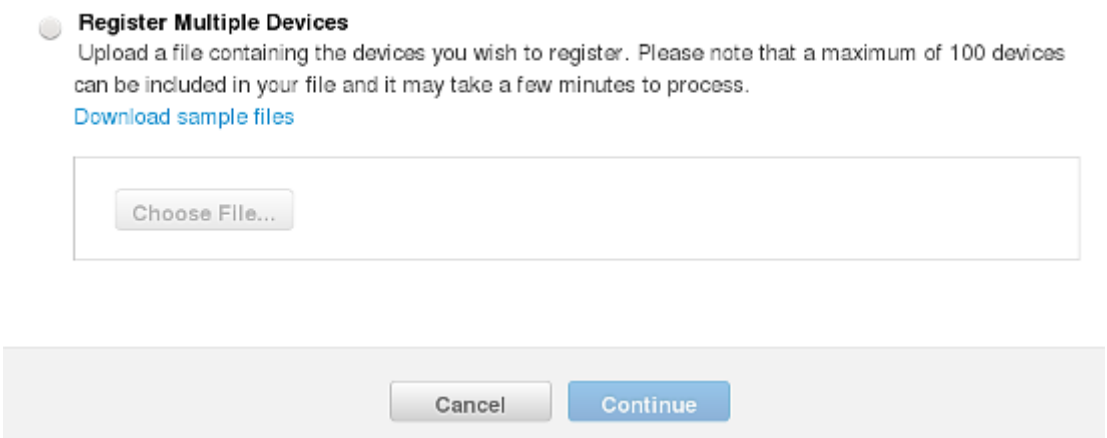
Return to your account on Developer.apple.com, go to **IOS Apps** > **Devices** > **All**, and click the plus button on the top right to register a new device. You can make the name anything you want, and the UDID must be the UDID copied from iTunes.



If you have a large number of devices to register, you may enter them in a text file in this format, and then upload the file:

```
Device ID  Device Name
A1234567890123456789012345678901234567890123456789   NAME1
B1234567890123456789012345678901234567890123456789   NAME2
```

Click **Download sample files** to see examples of plain text and markup files.



When you are finished entering your device IDs click the **Continue** button. Verify, and then click **Done**.

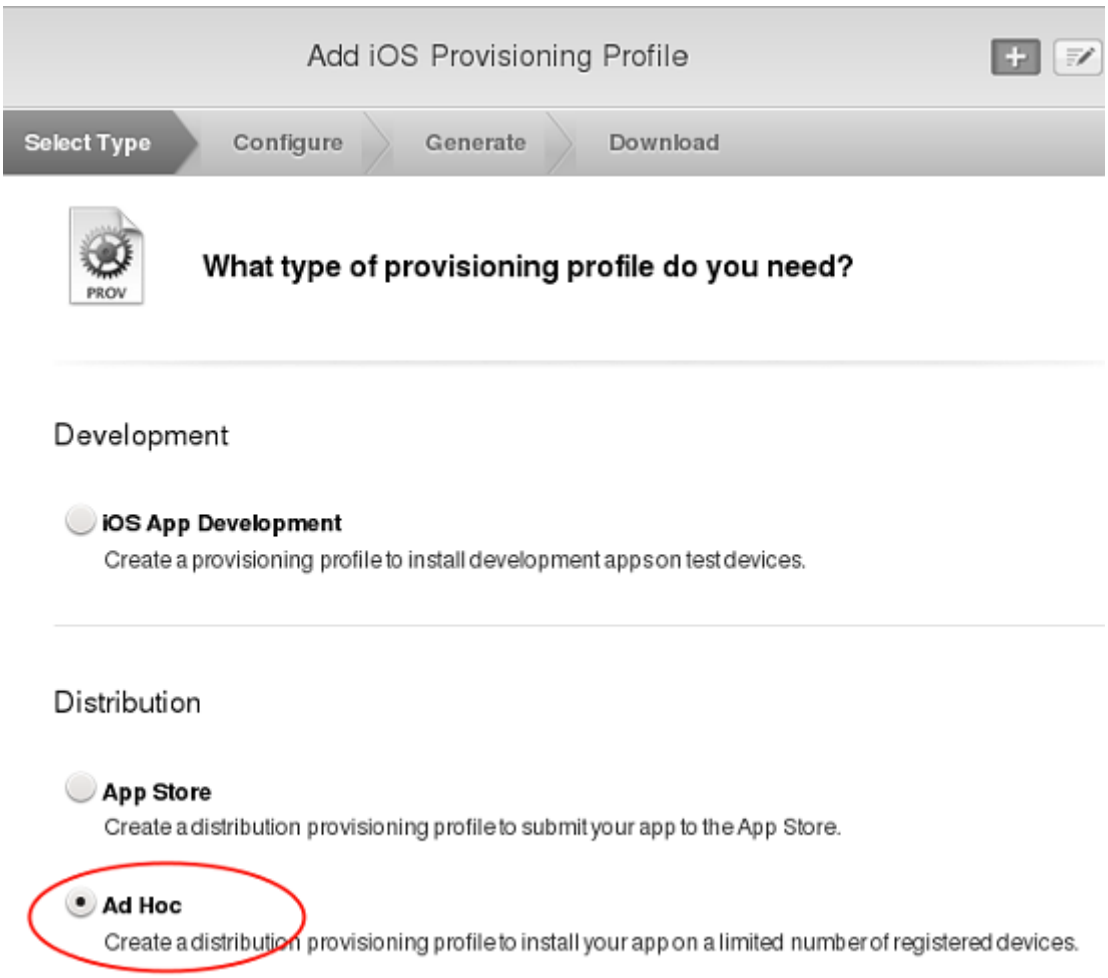
Create Provisioning Profiles

Next Step

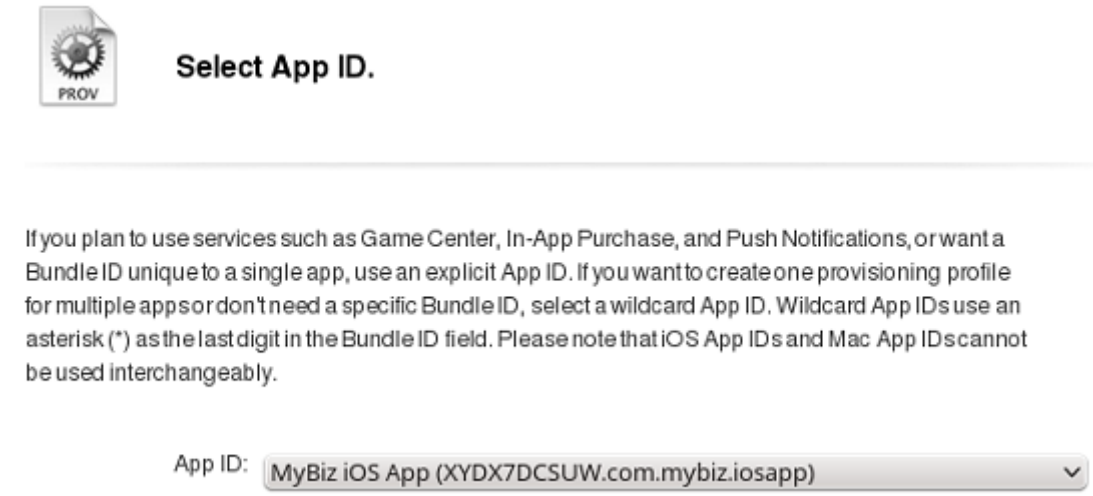
The next phase of this glorious journey is to create eight provisioning profiles: 4 Ad Hoc and 4 App Store <app_store_profiles_label>. You will email the four Ad Hoc profiles, and your P12 certificate <publishing_ios_app_6> (which you will create after your provisioning profiles), to support@owncloud.com after building your branded app with the ownBrander app on customer.owncloud.com. **Do not send us the App Store profiles.** All eight of these profiles must be stored on your Mac PC.

First Ad Hoc Provisioning Profile

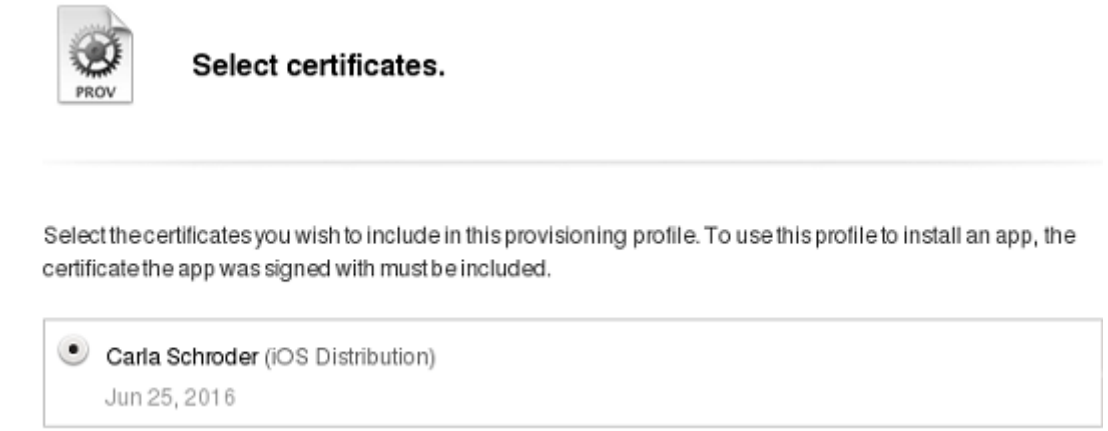
Go to **Provisioning Profiles** > **All**, then click the **[plus button]** (top right) to open the *Add iOS Provisioning Profile* screen. Select **[Ad Hoc]** and click **[Continue]**.



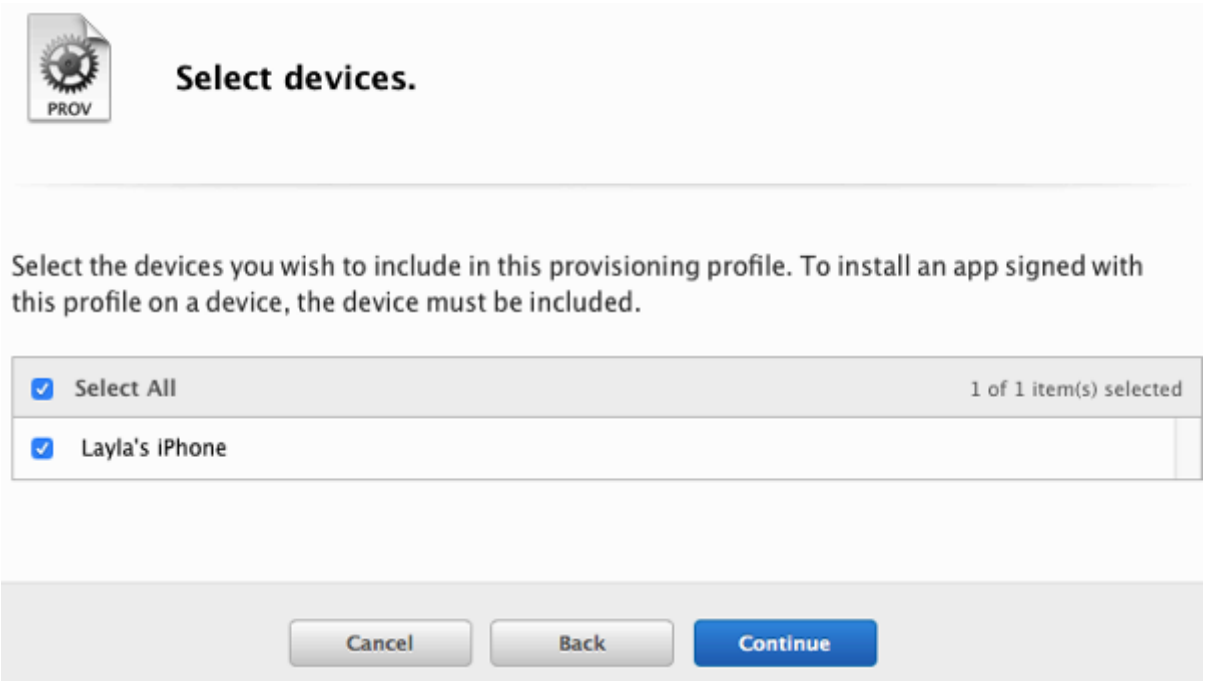
On the **Select App ID** screen select the first of the three App IDs that you created and click **[Continue]**. (The first one has the shortest name, if you followed the naming conventions in this manual.)



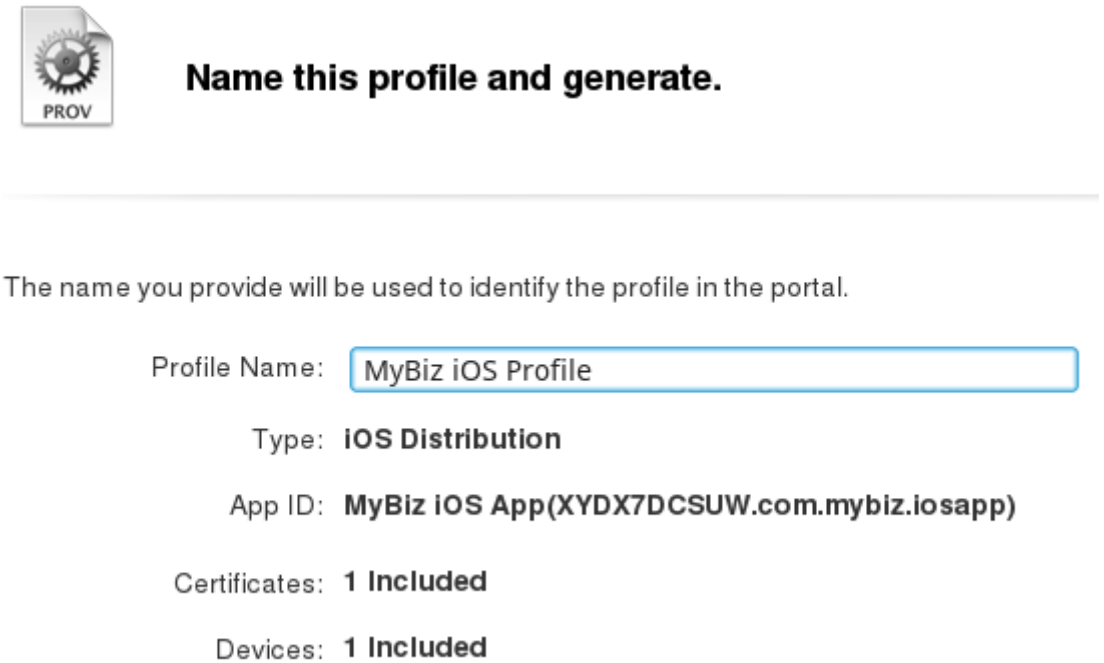
Select the certificate that you created at the beginning of this process and click **[Continue]**.



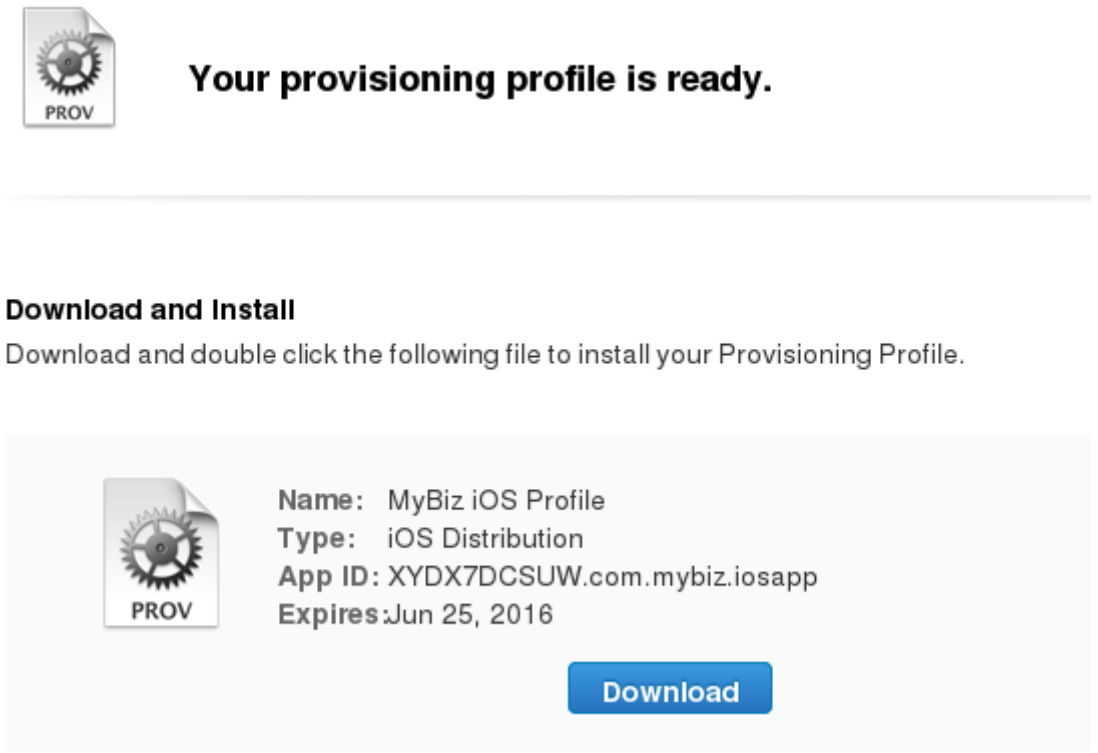
Select the devices that you want to install and test your app on, then click **[Continue]**.



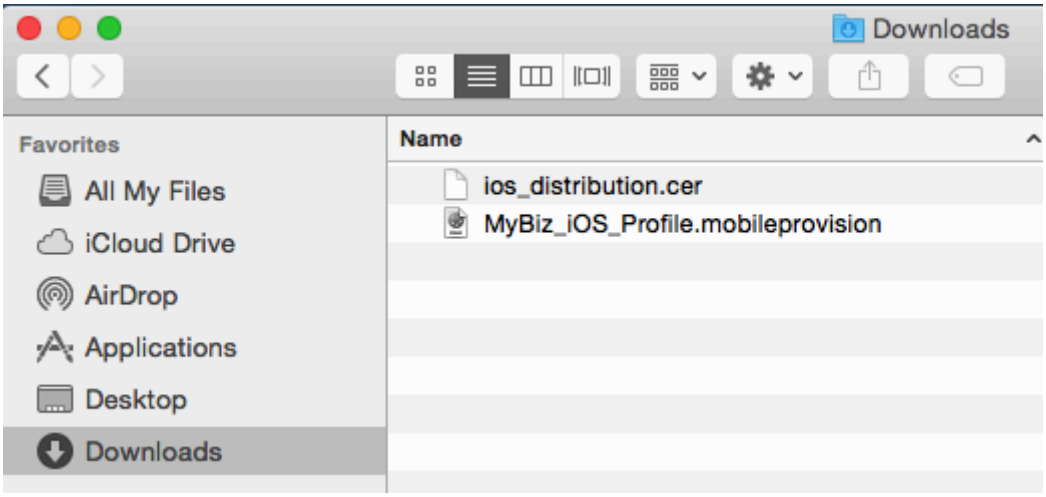
Name your provisioning profile with a descriptive **Profile Name** and click **[Generate]**.



When it has generated, download your new profile to your Mac computer.

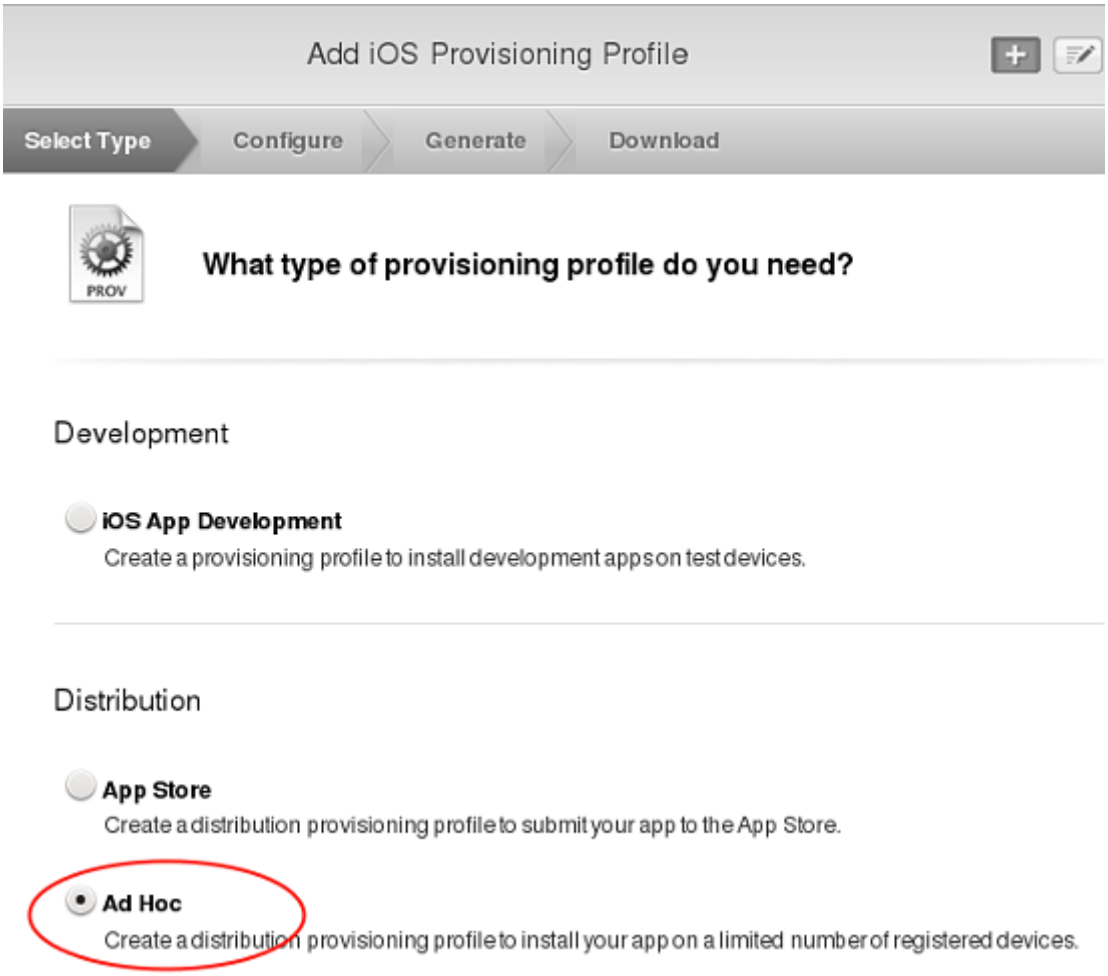


Find it on your Mac (usually the Download folder) and double-click to install it in Xcode.

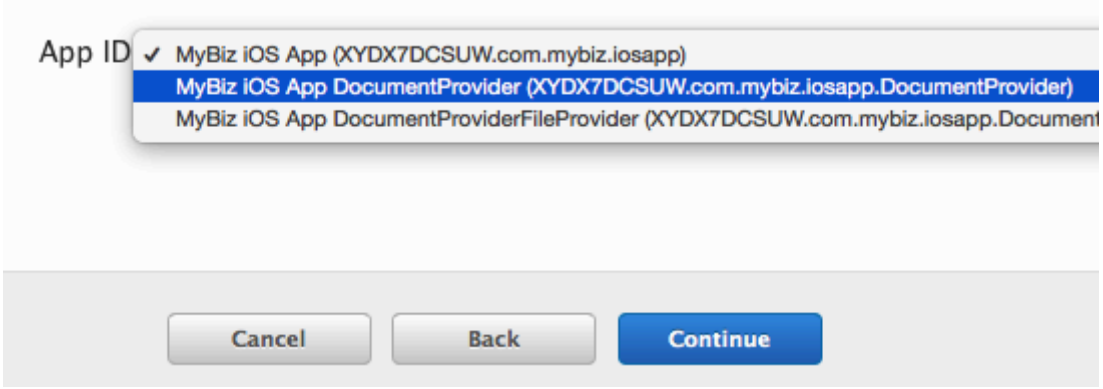


Second Ad Hoc Provisioning Profile

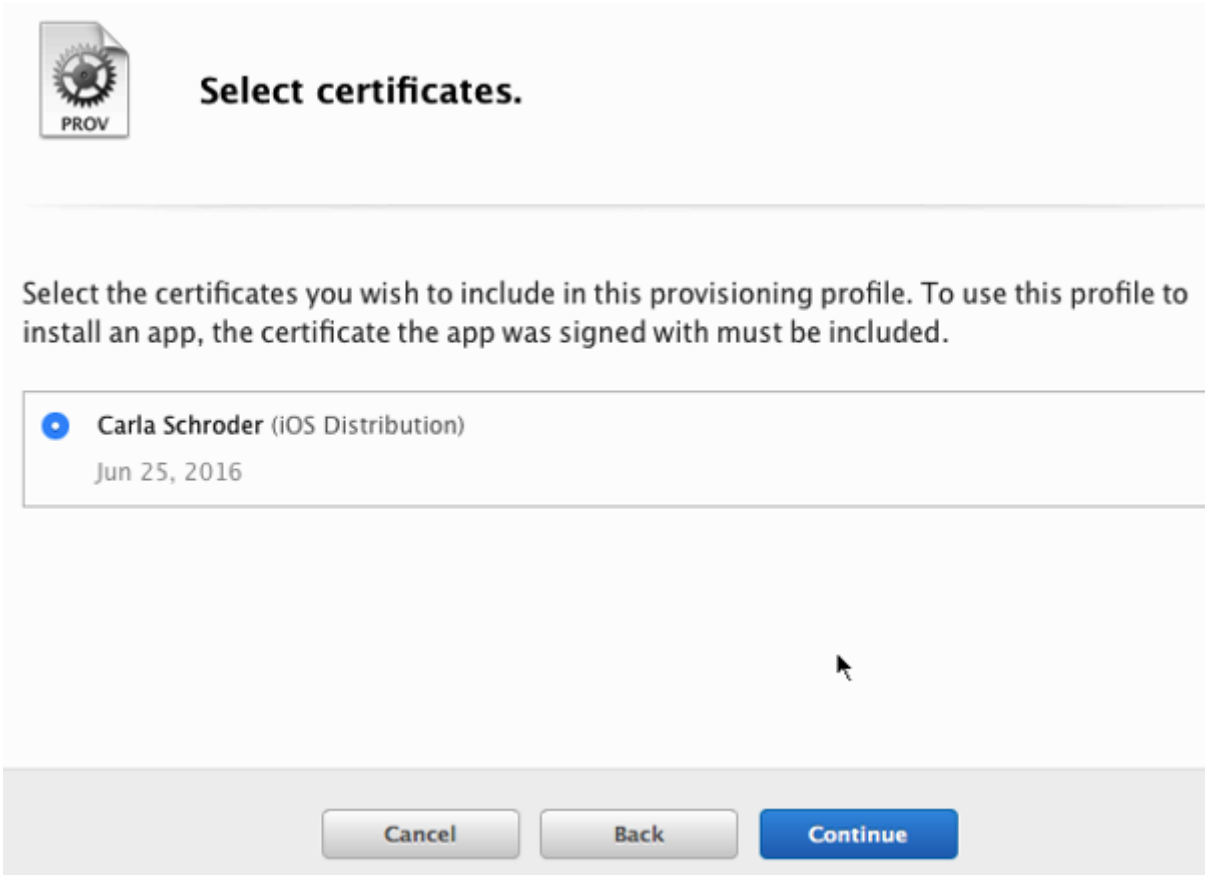
Return to the "Your provision profile is ready" screen, scroll to the bottom and click **[Add Another]**. On the following screen select **[Ad Hoc]** and click **[Continue]**.



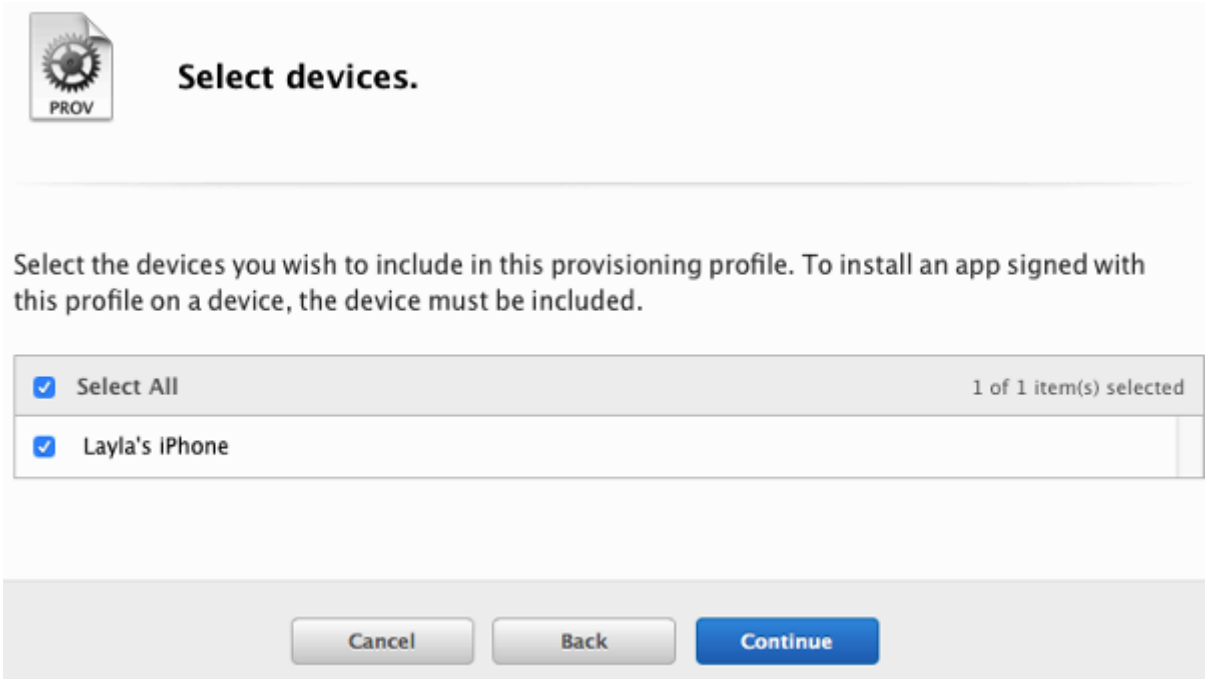
This time select the **.DocumentProvider** app ID and click **[Continue]**.



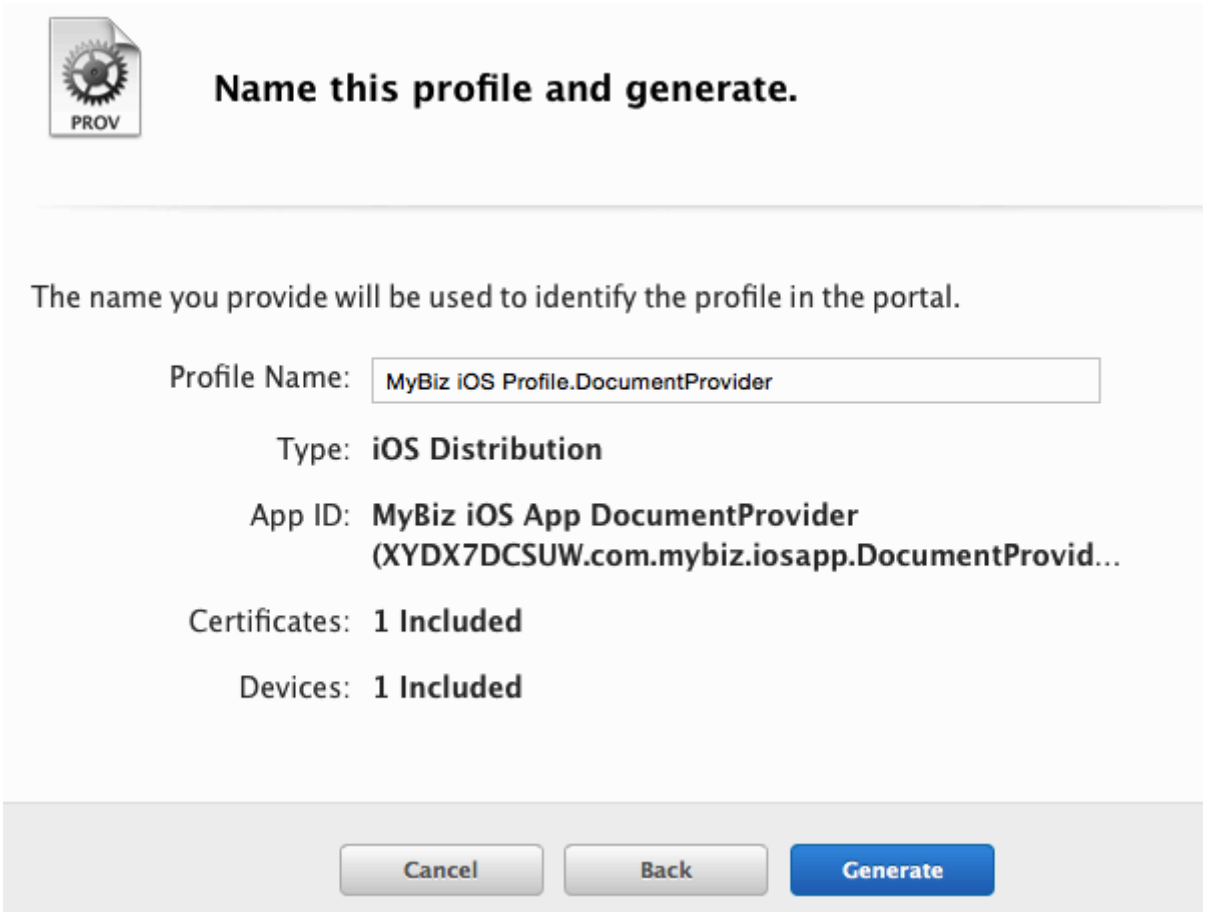
Select the certificate that you created at the beginning of this process and click **[Continue]**.



Select the devices that you want to install and test your app on, then click **[Continue]**. These must be the same devices you selected for the first provisioning profile.



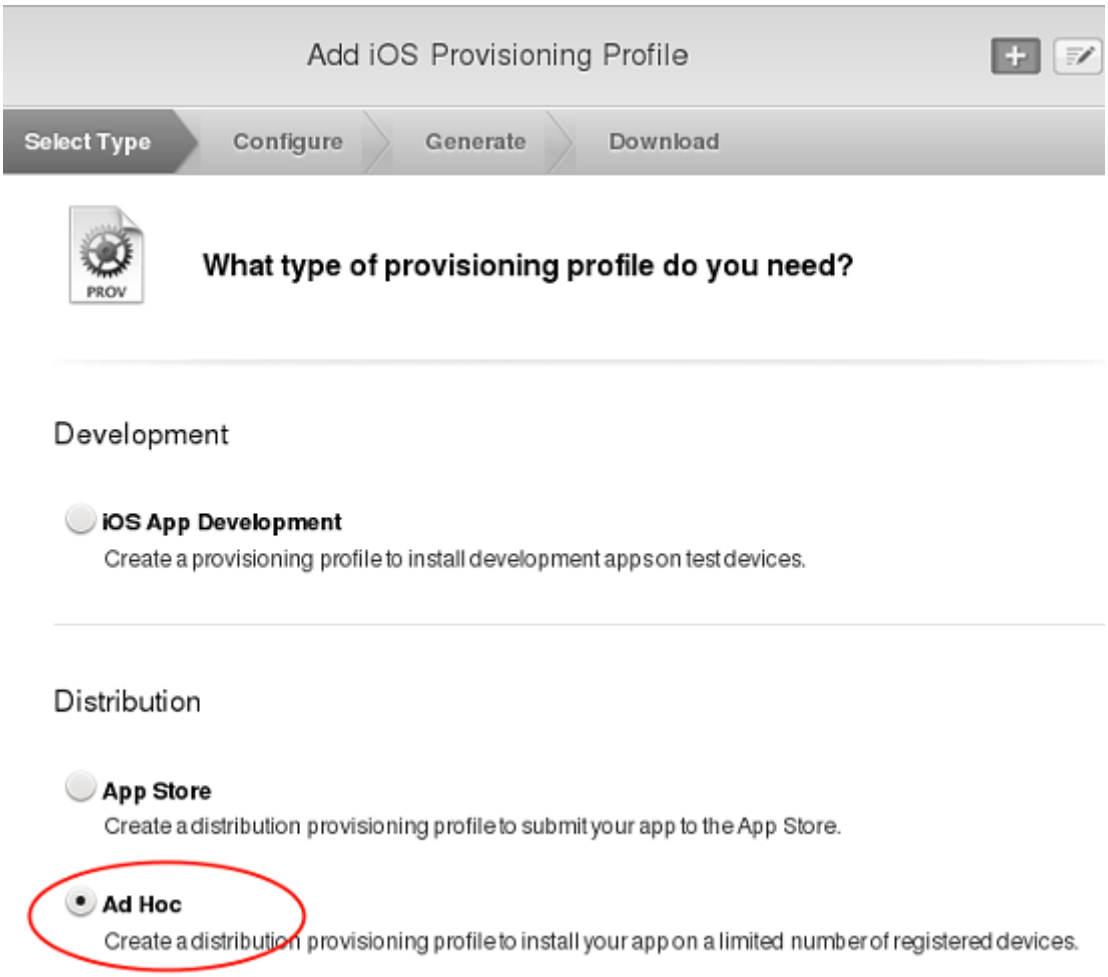
Give this provisioning profile the same name as your first profile, plus **.DocumentProvider** and click **[Generate]**.



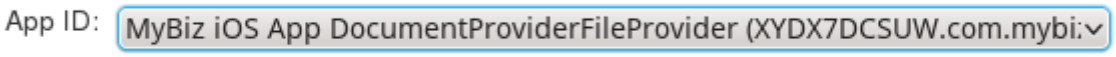
Just like the first provisioning profile, download it to your Mac computer, and then double-click to install it in Xcode.

Third Ad Hoc Provisioning Profile

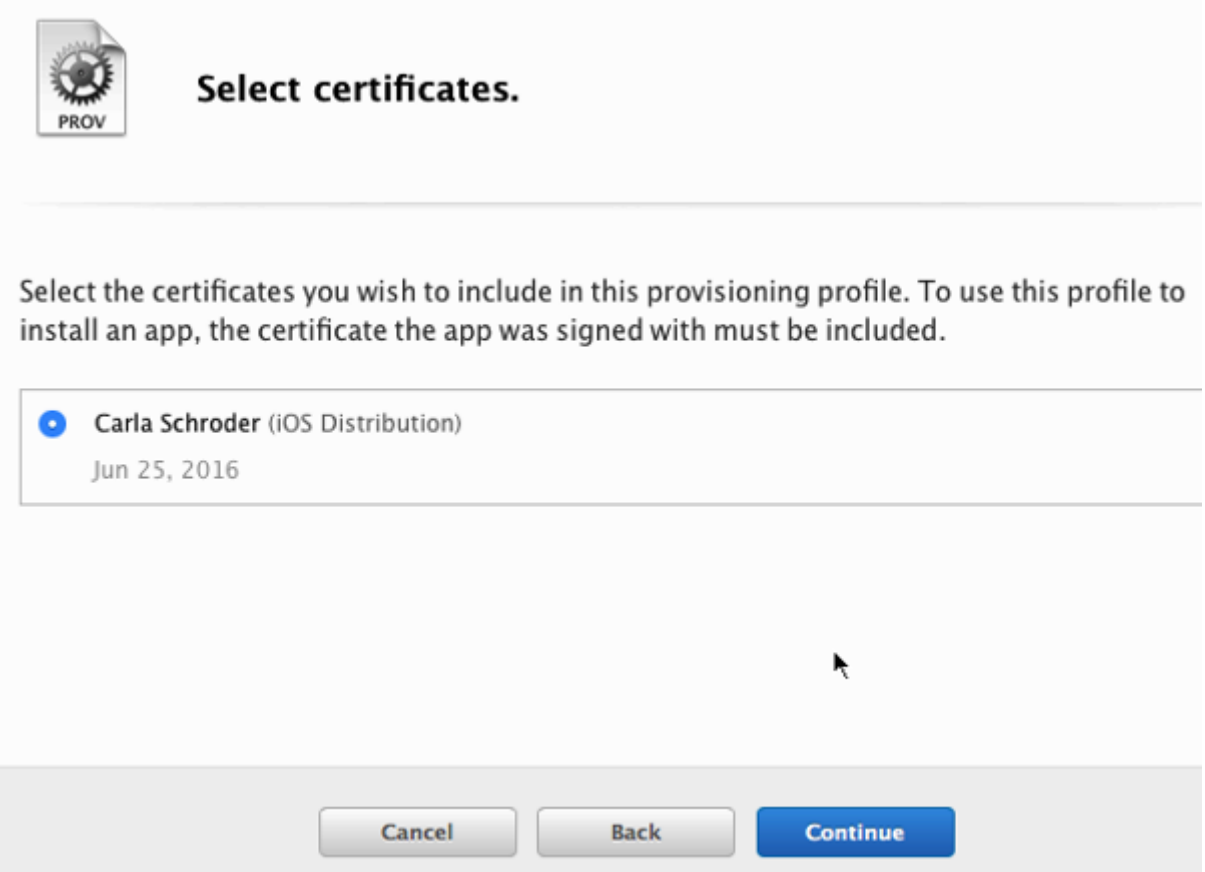
Return to the "Your provision profile is ready" screen, scroll to the bottom and click **[Add Another]**. On the following screen select **[Ad Hoc]** and click **[Continue]**.



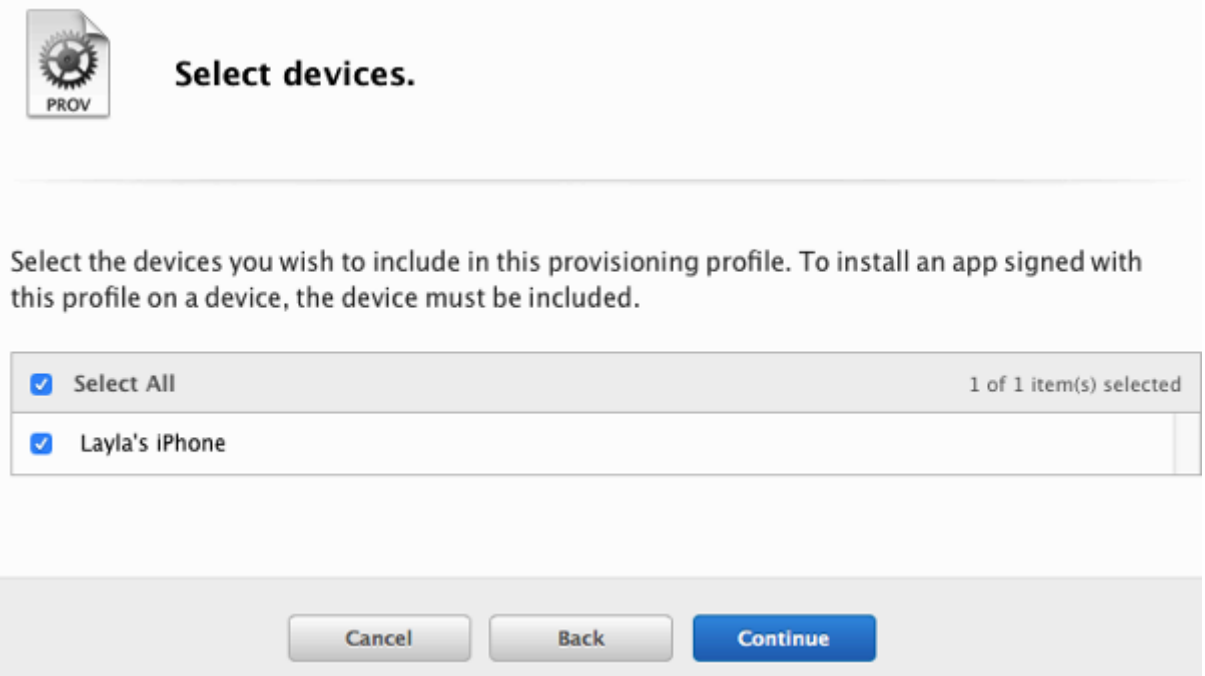
This time select the **.DocumentProviderFileProvider** app ID and click **Continue**.



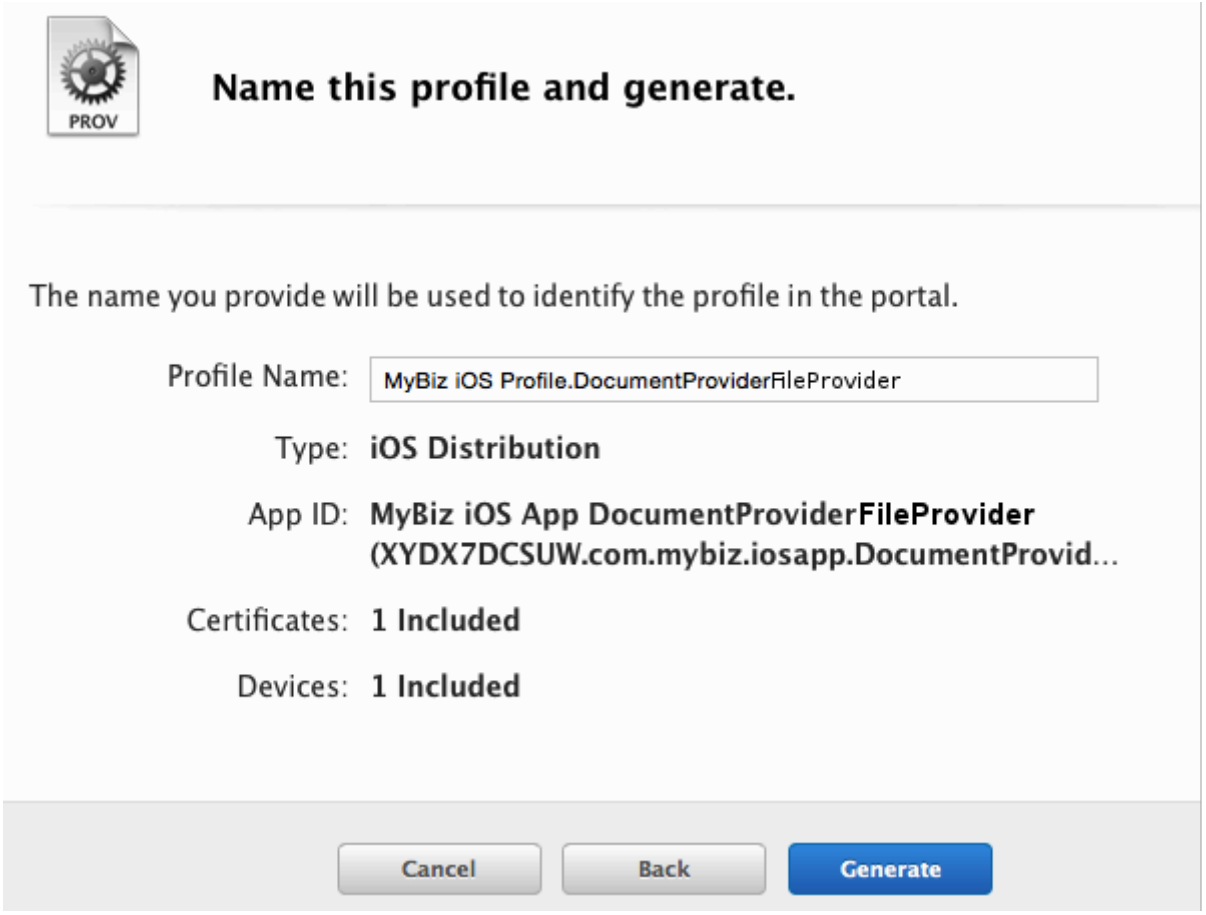
Select the certificate that you created at the beginning of this process and click **[Continue]**.



Select the devices that you want to install and test your app on, then click **[[Continue]**. These must be the same devices you selected for the first provisioning profile.



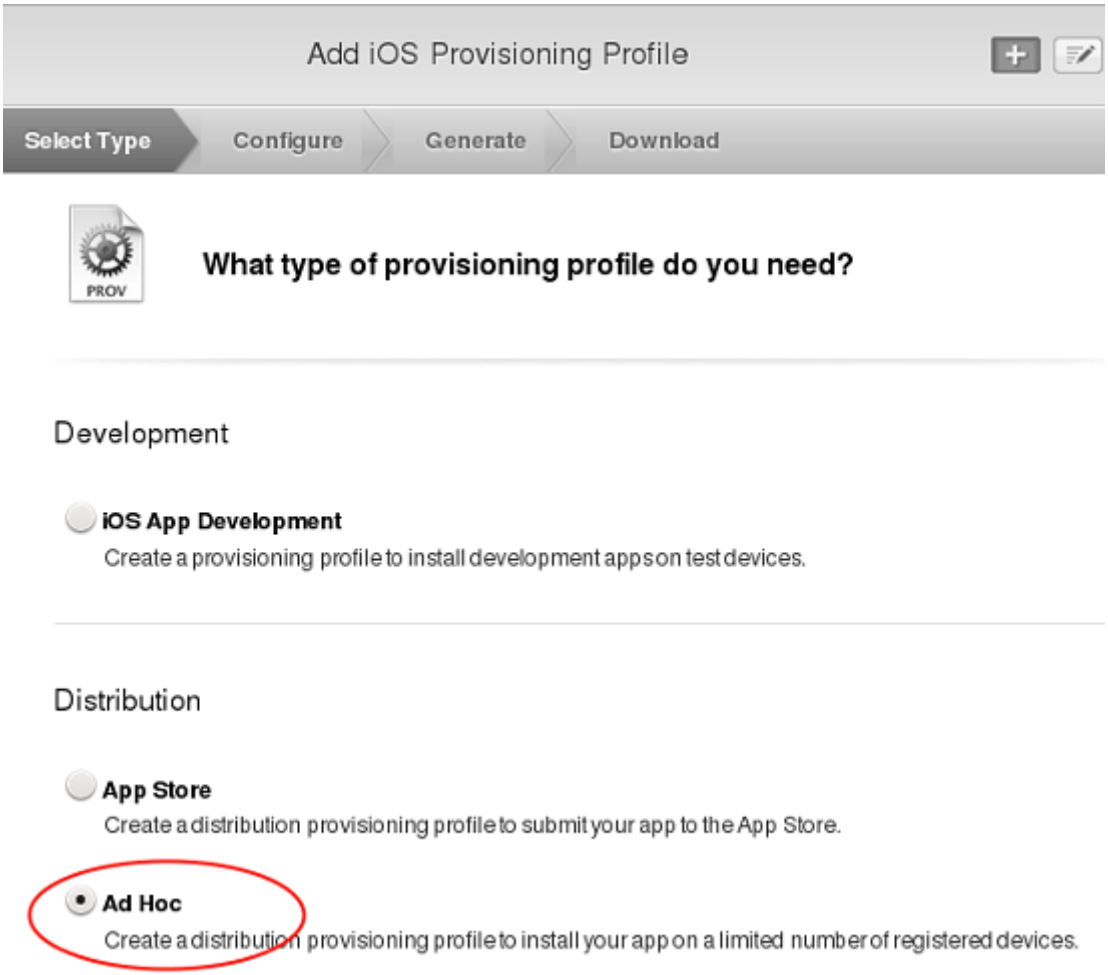
Give this provisioning profile the same name as your first profile plus **.DocumentProviderFileProvider** and click **[Generate]**. There is a 50-character limit, but don't worry about counting characters because it will be automatically truncated if you go over.



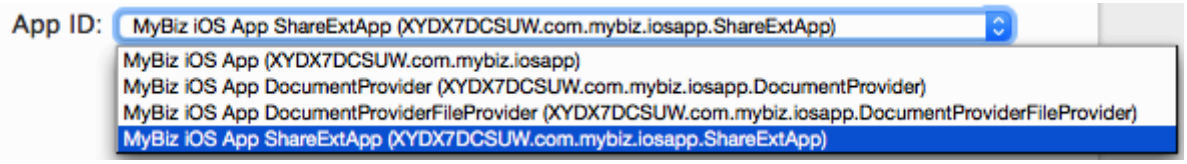
Download it to your Mac computer, and then double-click to install it in Xcode.

Fourth Ad Hoc Provisioning Profile

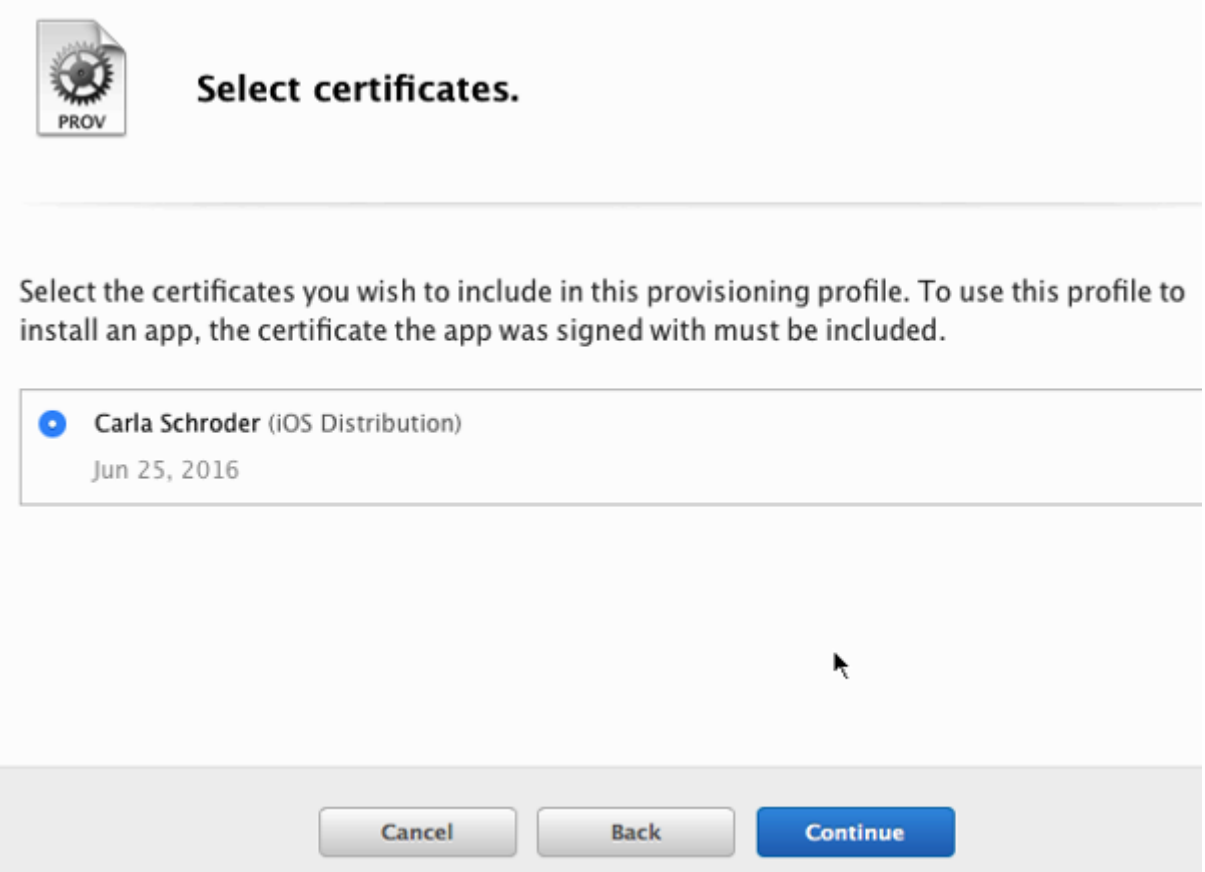
Return to the "Your provision profile is ready" screen, scroll to the bottom and click **[Add Another]**. On the following screen select **[Ad Hoc]** and click **[Continue]**.



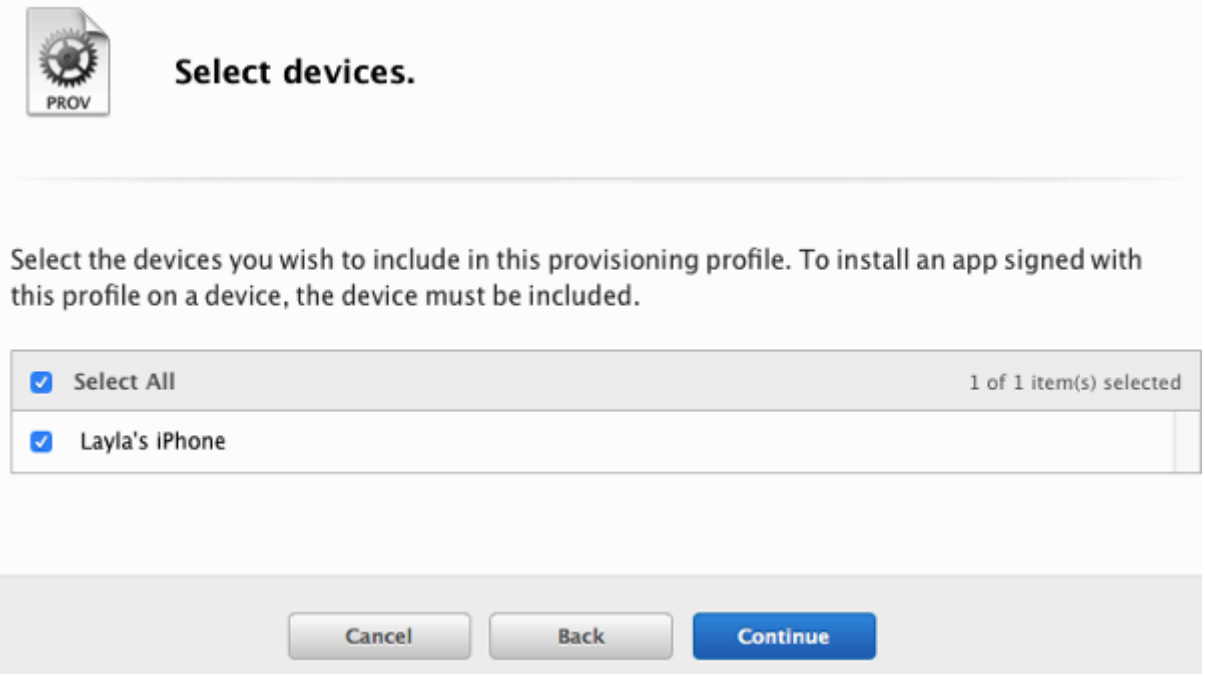
This time select the **.ShareExtApp** app ID and click **[Continue]**.



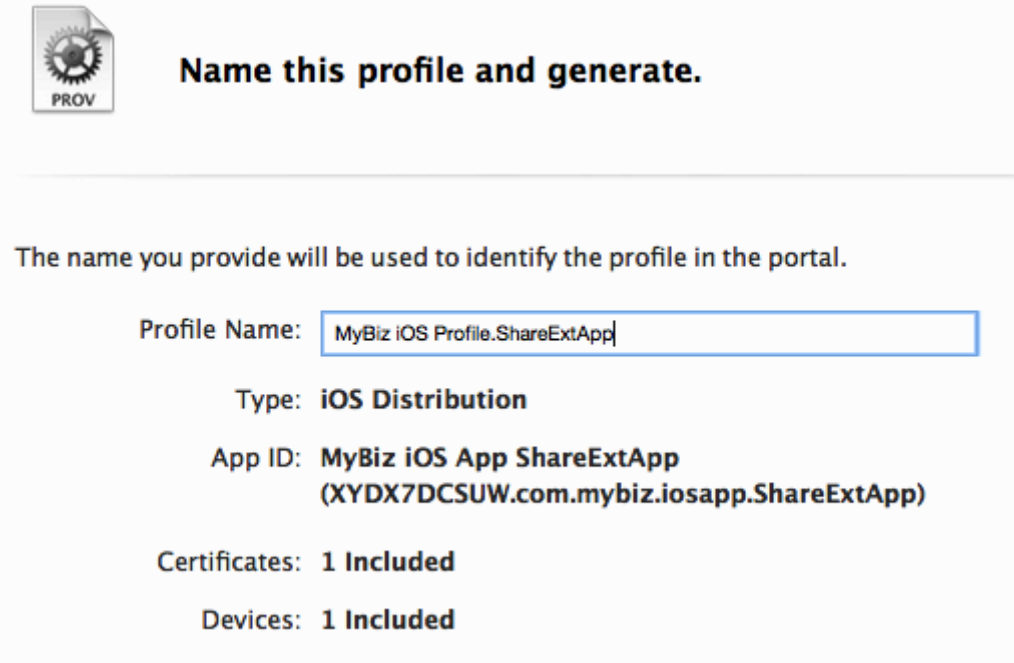
Select the certificate that you created at the beginning of this process and click **[Continue]**.



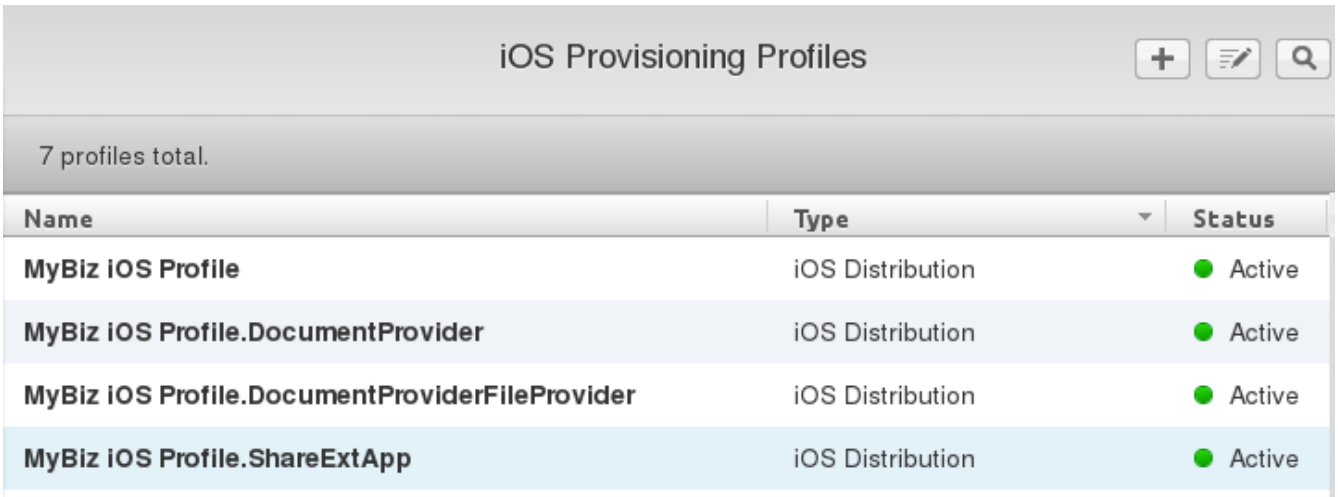
Select the devices that you want to install and test your app on, then click **[Continue]**. These must be the same devices you selected for the first provisioning profile.



Give this provisioning profile the same name as your first profile plus **.ShareExtApp** and click **[Generate]**. There is a 50-character limit, but don't worry about counting characters because it will be automatically truncated if you go over.

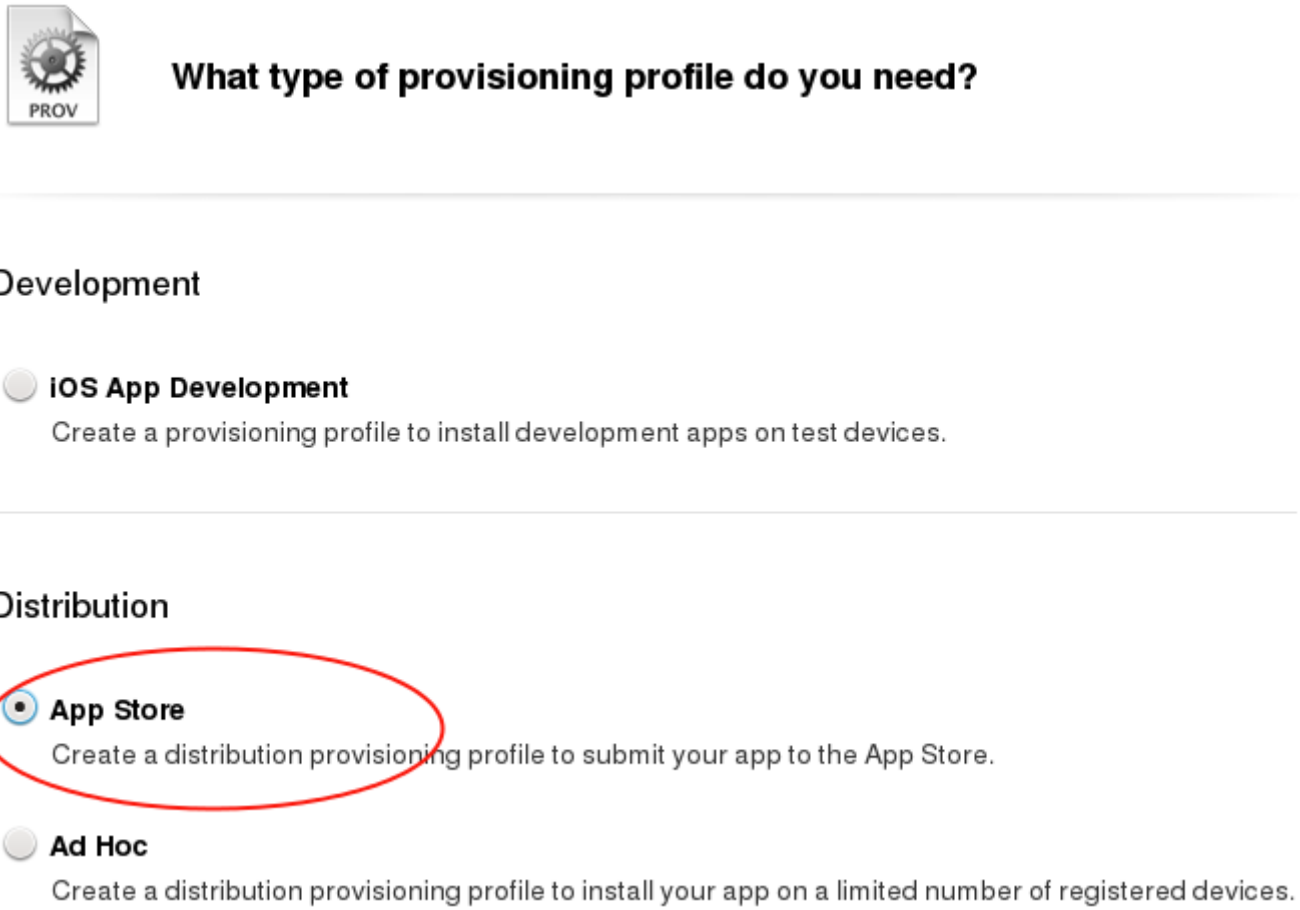


Download it to your Mac computer, and then double-click to install it in Xcode. You should now see all of your Ad Hoc provisioning profiles listed in your "iOS Provisioning Profiles".



Create Four App Store Profiles

Creating your four App Store profiles is the same as creating your Ad Hoc profiles, except that when you start you check the App Store checkbox, and you won't select testing devices.

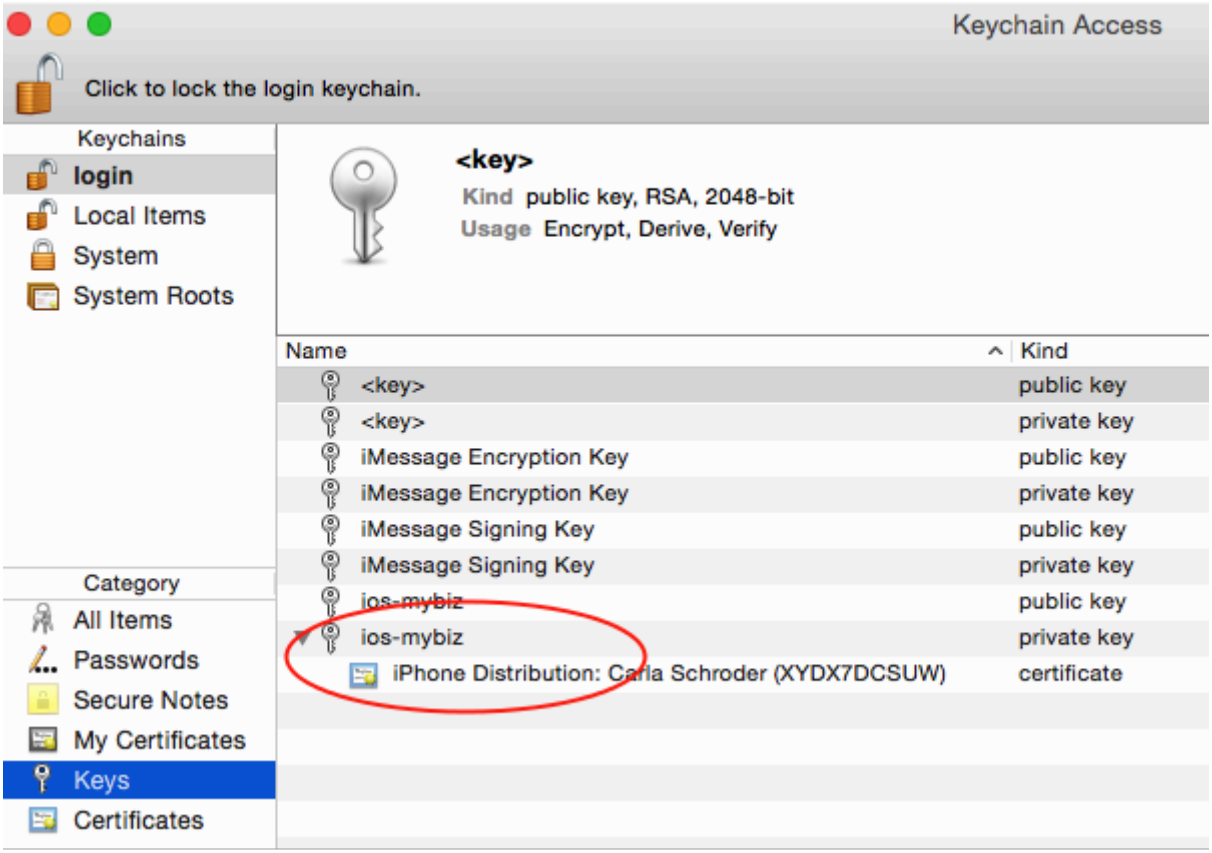


When you're finished, you'll have eight new provisioning profiles. Remember, when you build your app on ownBuilder you only send in the four Ad Hoc profiles, plus your P12 certificate.

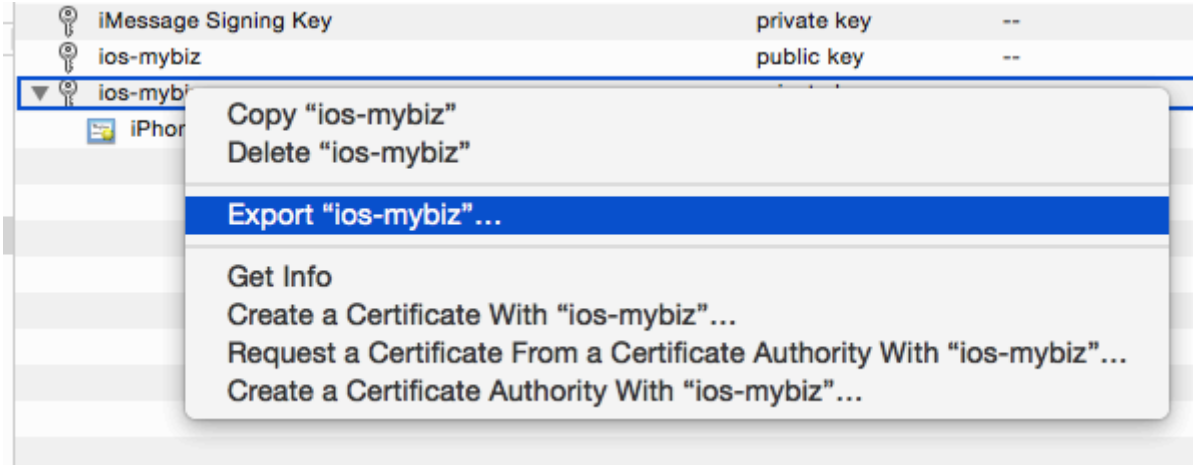
Go to the next page to learn how to create your P12 certificate <publishing_ios_app_6>.

Creating a P12 Certificate

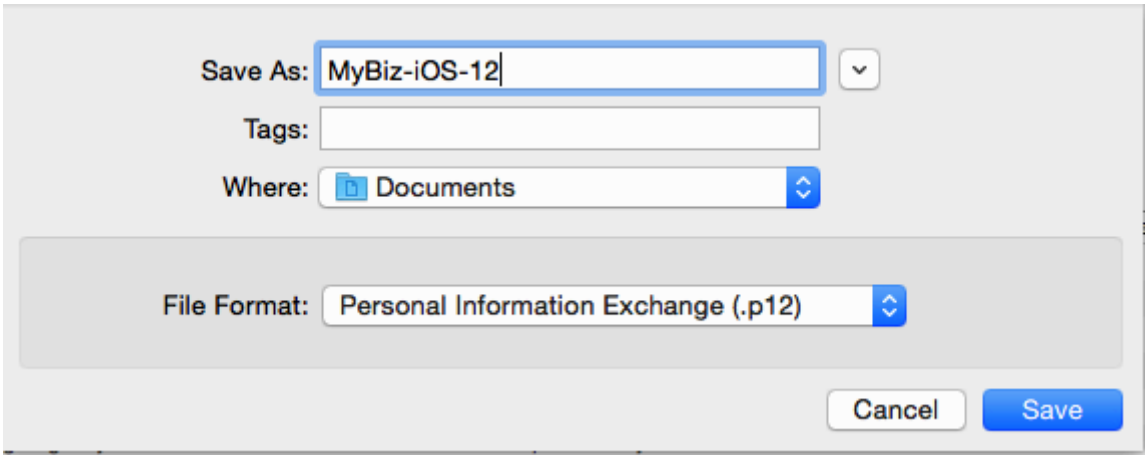
In addition to emailing your four Ad Hoc provisioning profiles to support@owncloud.com, you must also include your P12 certificate. To create this, return to Keychain Access on your Mac computer and find your private key that you created at the beginning (see Create Certificate Signing Request).



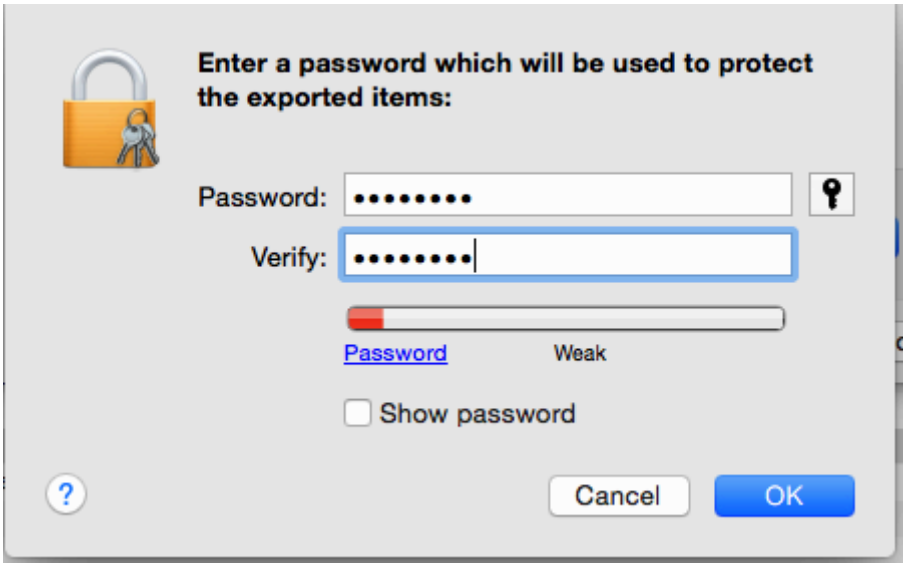
Right-click on your private key and left-click **Export [your key name]**.



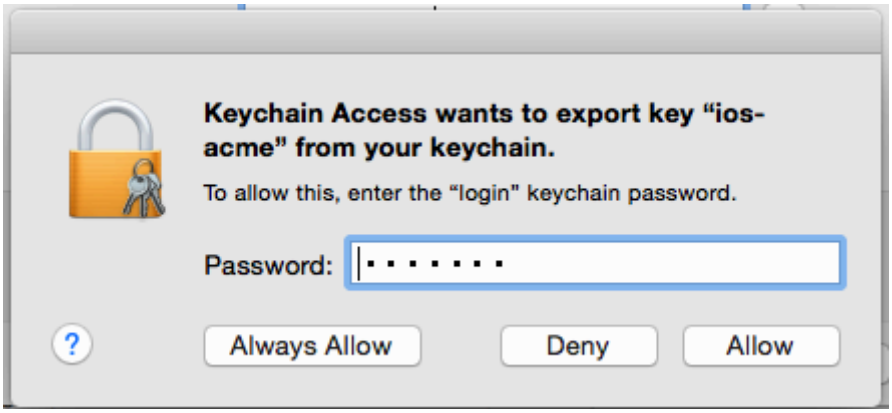
Enter any name you want, the location you want to save it to, and click **[Save]**.



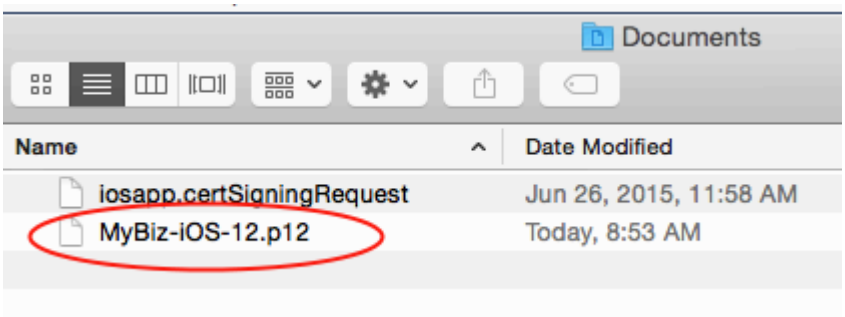
In the next screen you have the option to enter a password. If you put a password on your P12 certificate you will have to include it when you send your certificate and provisioning profiles to support@owncloud.com. Click **[OK]**.



On the next screen you must enter your login keychain password, which is your Mac login password, and click **[Allow]**.



Now your new P12 certificate should be in the directory you saved it in.

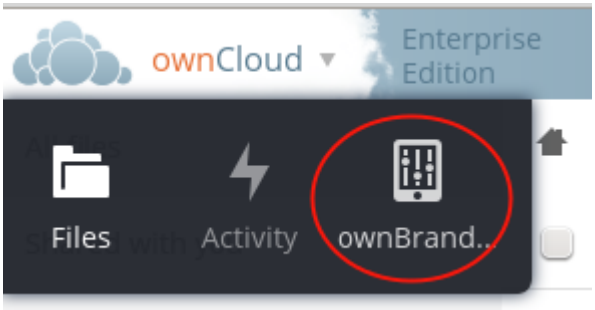


You have now completed all the necessary steps for signing your branded iOS app. The next step is to build your app with the ownBrander app on https://customer.owncloud.com.

Building Your iOS App With ownBrander

Build Your Branded iOS App

At long last you have arrived at the point where you can actually build your branded iOS app. Log into your account on customer.owncloud.com/owncloud and open the ownBrander app.



If you don't see the ownBrander app, open a support request with the **[Open Case]** button.

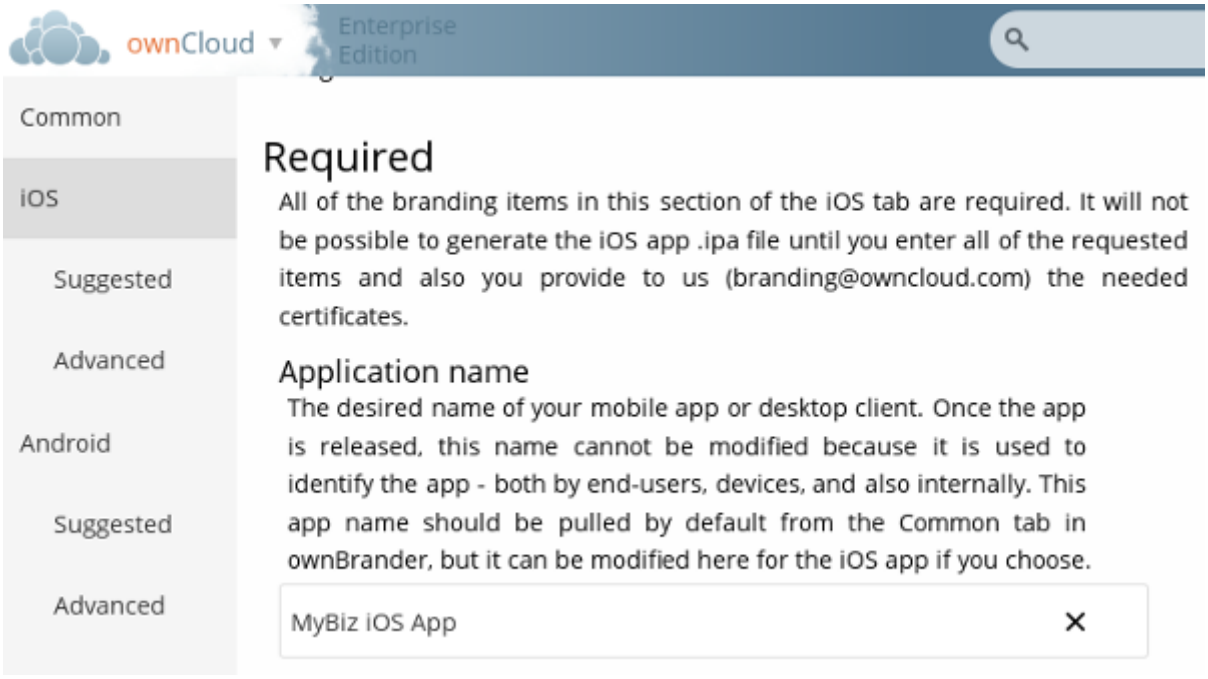
Your first ownBrander task is to review the iOS page on ownBrander for your image requirements. You will need a lot of them, in specific sizes and formats, and they're all listed on the ownBrander page.

There are three sections: Required, Suggested, and Advanced. The Required sections contains all of the required elements that you must configure. Suggested and Advanced allow additional customizations.

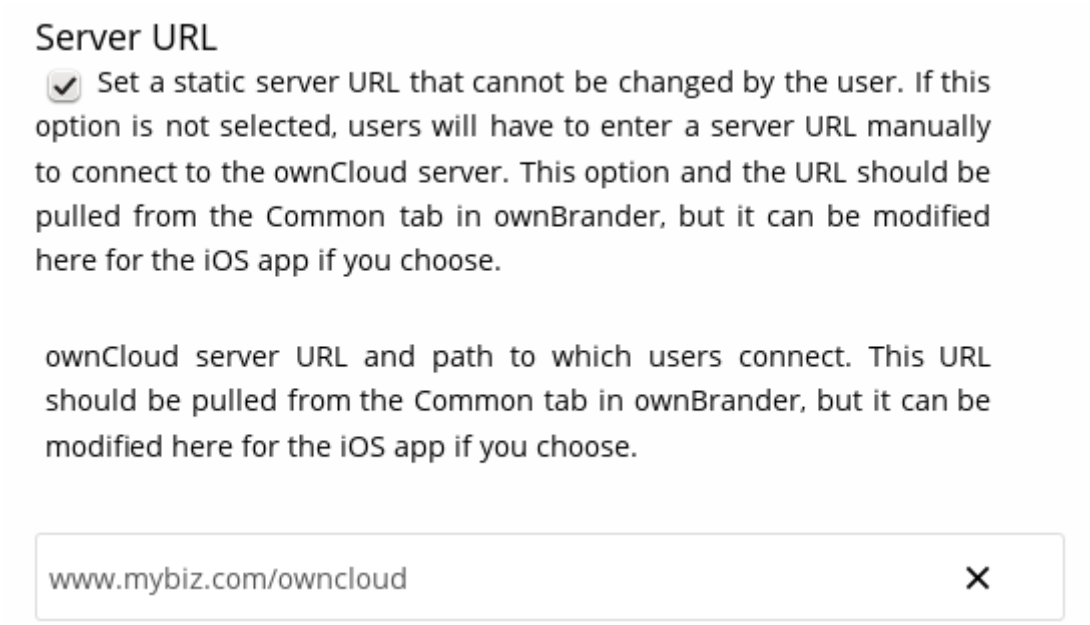
When you have completed and submitted your app, email your three provisioning profiles and P12 certificate to support@owncloud.com.

Required Section

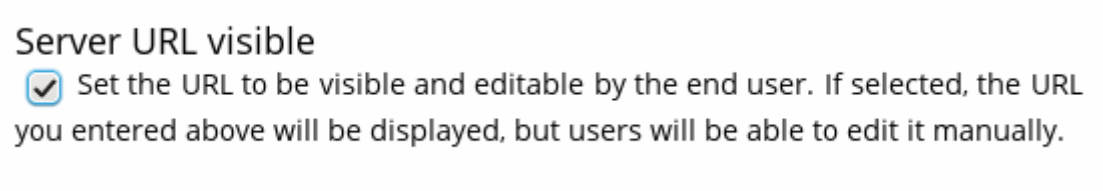
Enter your application name. This can be anything; in this example it is the same name used in our signing certificate examples.



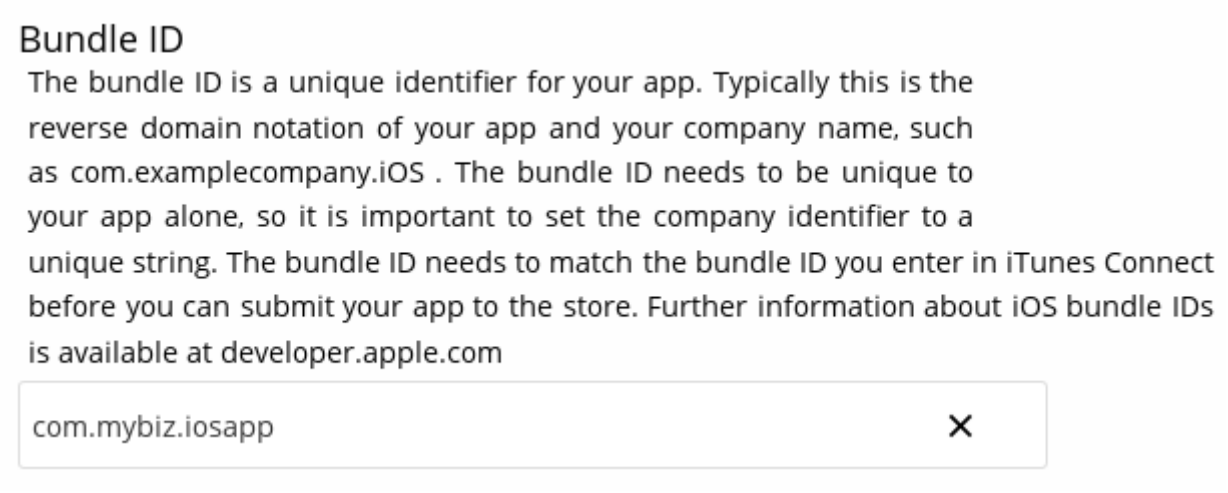
Next, enter your ownCloud server URL. This hard-codes it into your app. If you leave this blank then your users will have to enter it every time they use the app.



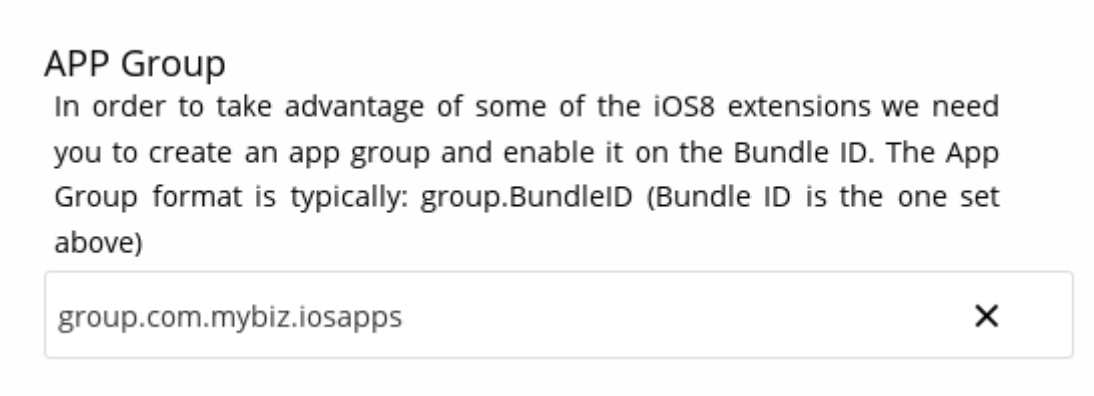
Check **Server URL Visible** to make your ownCloud server URL the default, and to allow your users to enter a different URL.



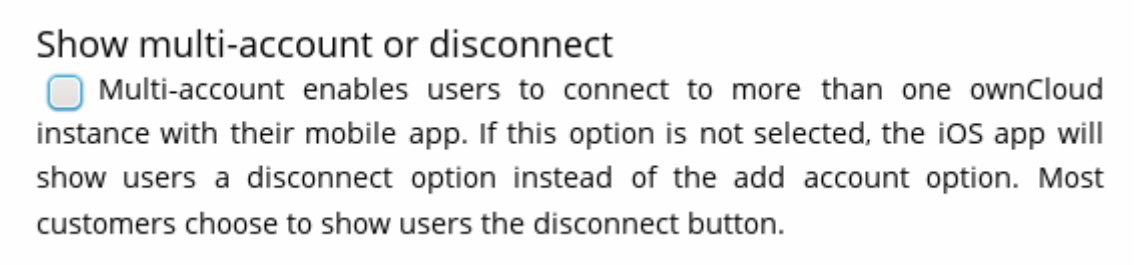
And now, the all-important **Bundle ID**. Make sure that this is exactly the same as the **Bundle ID** you created on developer.apple.com (see Bundle ID).



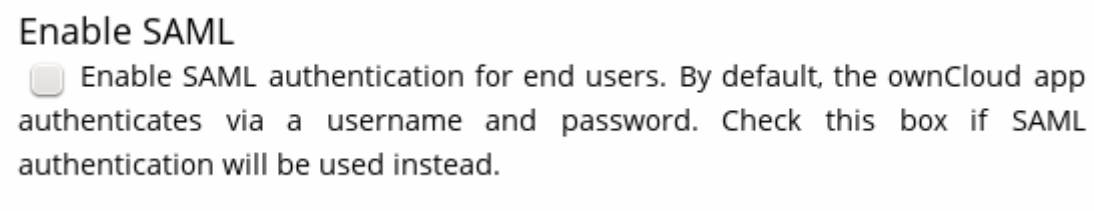
You must also enter the **App Group** you created.



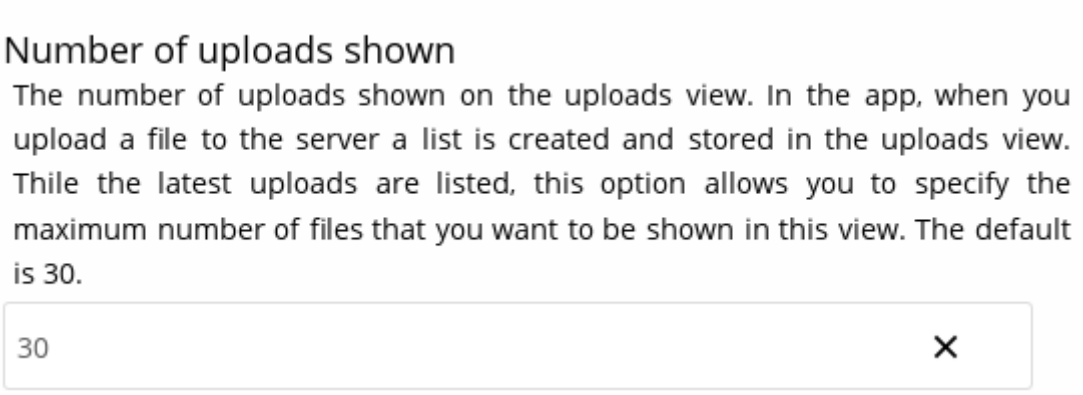
Check **Show multi-account or disconnect** if you plan to allow your users to have more than one ownCloud account.



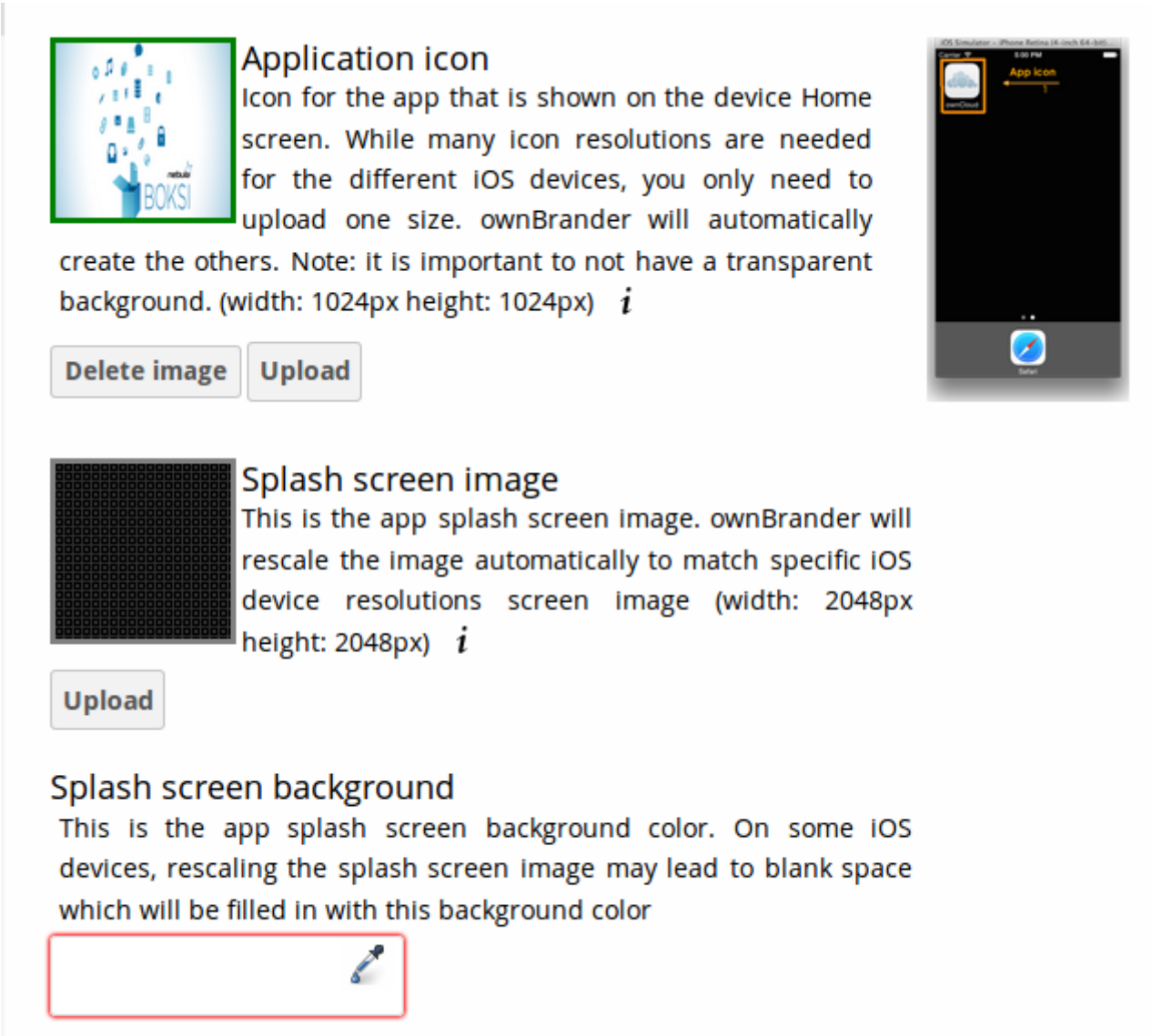
Check **Enable SAML** authentication if that is what you use on your ownCloud server. Otherwise leave it blank.



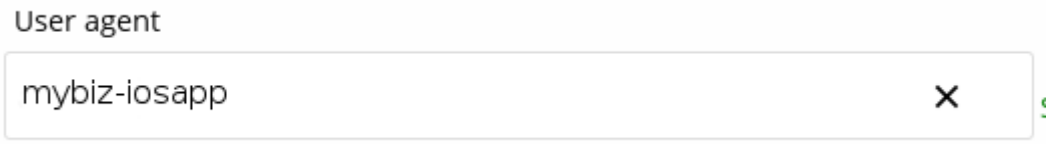
Number of uploads shown controls the length of the most recent uploads list on the app. The default is 30.



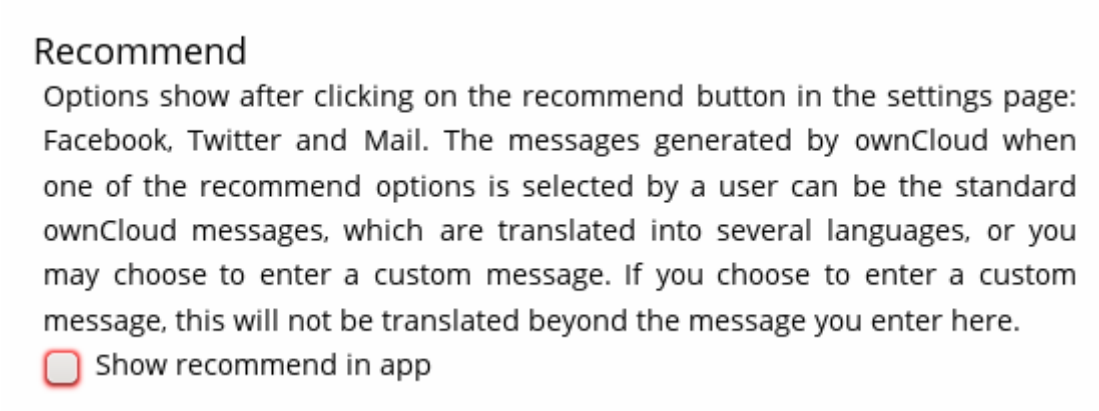
The next section is for uploading your custom artwork to be built into the app. The ownBuilder app tells you exactly which images you need, and their required size. You only need one Splash Screen image, and ownBrander will automatically resize and crop it for different-sized screens. You must also select a background color, which ensures that the splash screen image is always at the correct size ratio. (Click the example images on the right to enlarge them.)



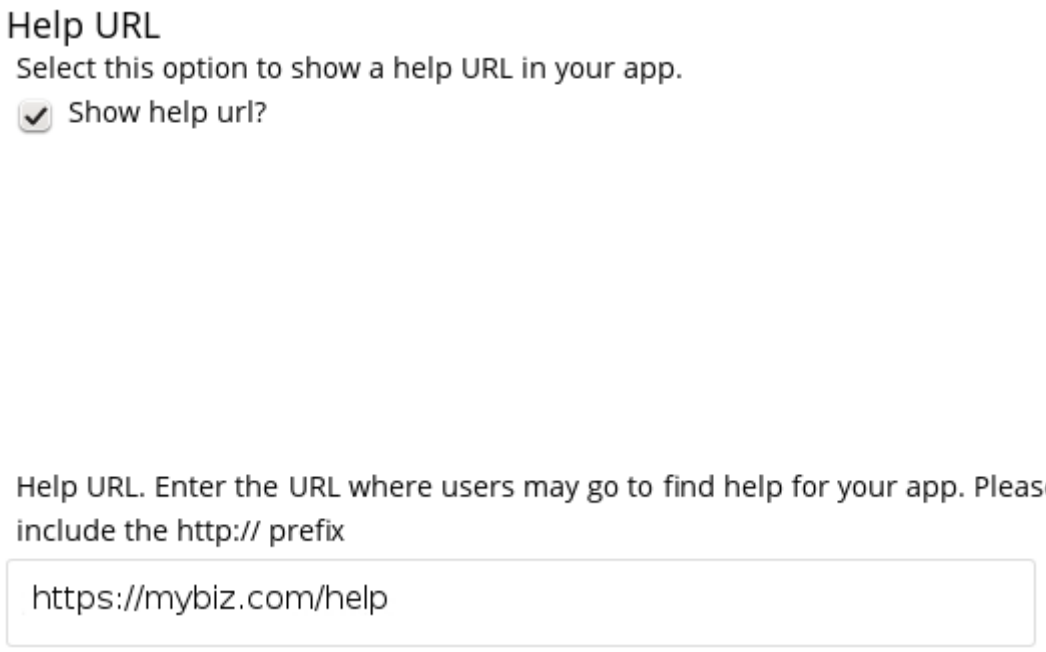
You may enter a custom **User agent**, which is useful for traffic analysis and whitelisting your app.



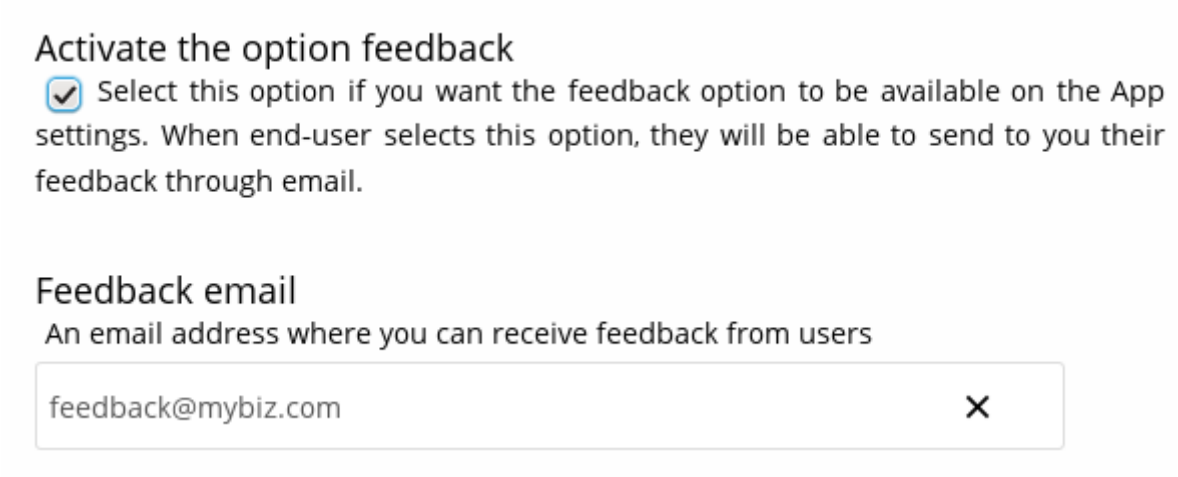
Check **Recommend** to open a Twitter, Facebook, and Email recommendation configurator.



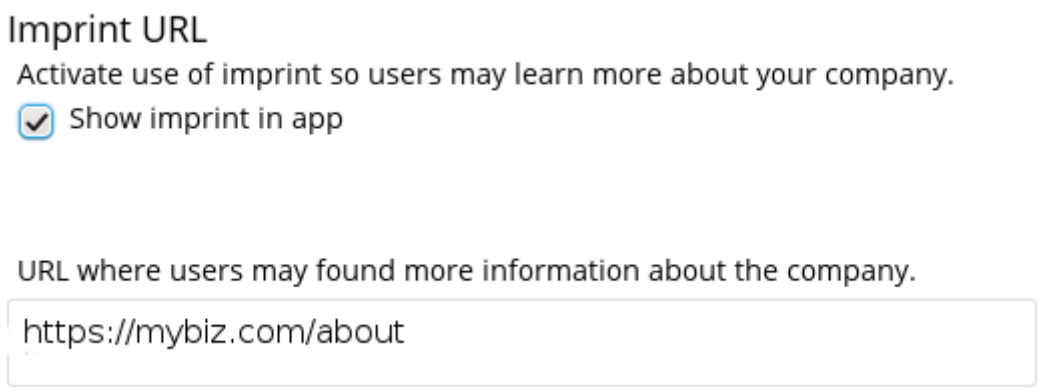
If you have online help, enter the URL here.



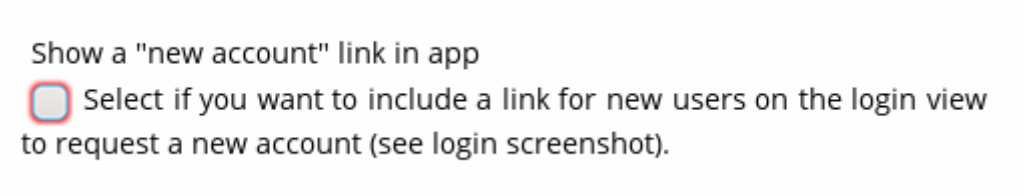
Activate the option feedback creates an option for your users to either enable or not enable the feedback option on their devices. If you enable this, enter your **Feedback email** address.



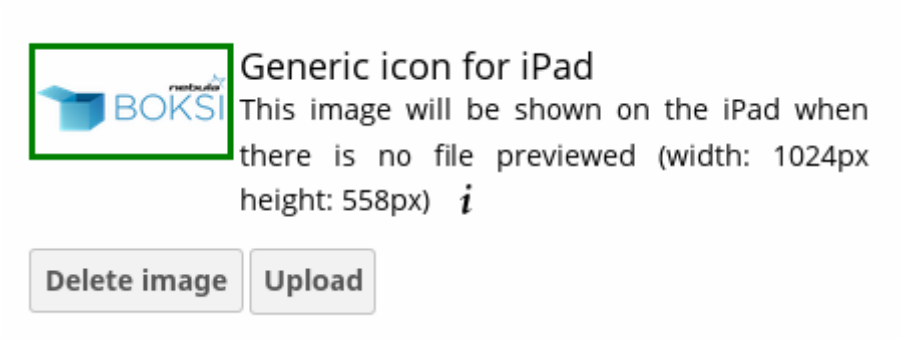
Enter your **Imprint URL** (your "about" page)



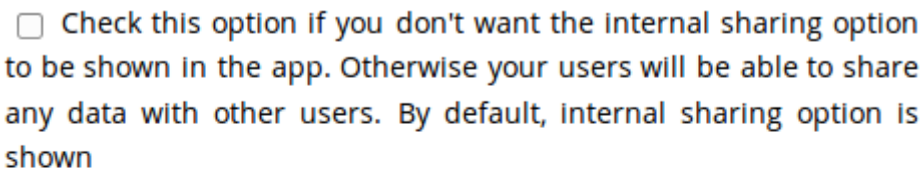
Check **Show a "new account" link in app** to allow new users to request a new account.

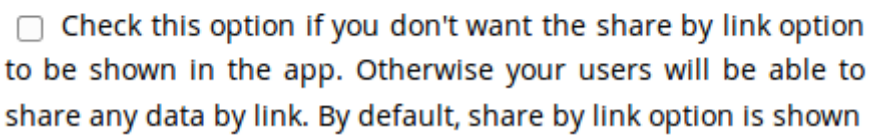


Upload an icon that will be displayed by default when there is no file preview to display.

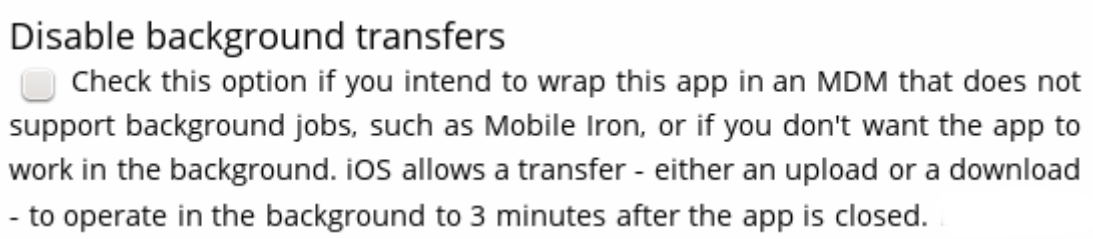


By default, both internal sharing and sharing by link are enabled. You have the options to disable one or both of these.

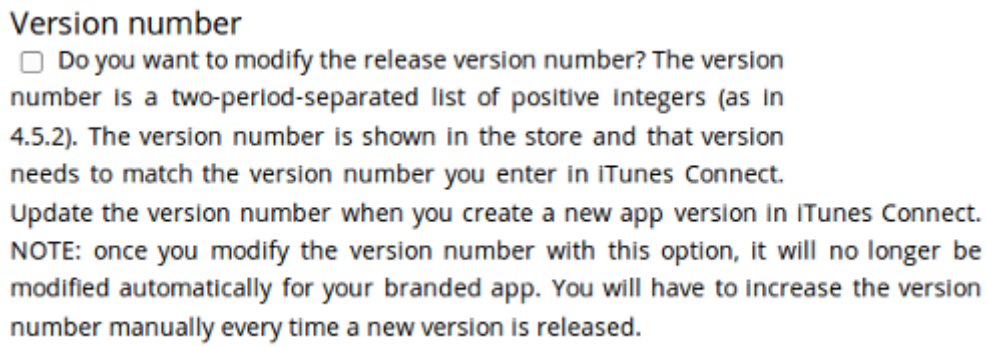




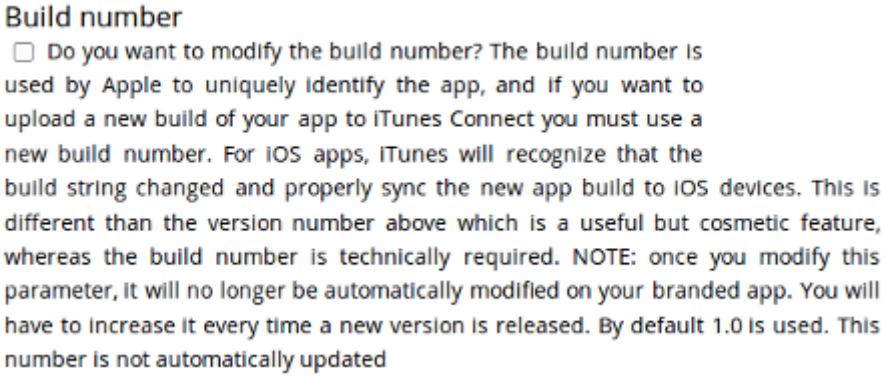
You may disable background transfers if you are using mobile device management (MDM), such as Mobile Iron, that does not support background jobs, or if you simply do not want to allow the app to work in the background. By default, the ownCloud iOS app supports background file transfers by taking advantage of Background Execution.



The default version number of your branded app is the same as the official ownCloud app. You have the option to customize your version number. Once you do this, you will have to update it manually for new releases. This must be the same as the version number that you enter in iTunes. Your version number is visible to your users.



You may also customize the build number, which defaults to 1.0.0. This must also be manually updated when you customize it. Your build number is used by iTunes to uniquely identify your app. When the build number changes, iTunes automatically syncs the updates for your users. The build number is not visible to your users.



That completes the required elements of your branded iOS app.

Suggested Section

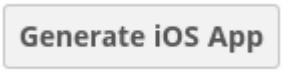
The Suggested section allows you to customize additional elements such as text and background colors, and icons. The Suggested items are all optional.

Advanced Section

The Advanced section allows you to optionally customize the color of messages such as connection status, error messages, letter separators, buttons, and additional icons.

Generate iOS App

When you have uploaded all of your images and completed your customizations, click the **Generate iOS App** button and take a well-deserved break. Remember to email your four Ad Hoc provisioning profiles and P12 certificate to support@owncloud.com.



You may go back and make changes, and when you click the **Generate iOS App** button the build system will use your latest changes.

Check your account on customer.owncloud.com in 48 hours to see your new branded ownCloud app.

Testing Your New Branded iOS App

Distribute the File

You'll distribute the file with the .ipa extension, like our example MyBiz iOS App-3.4.211.ipa, from your https://customer.owncloud.com/owncloud account to your beta testers. To do this you'll need a Mac computer, an iPhone or iPad registered in your Apple developer account, and the iTunes account associated with your Apple developer account.

1. Connect your registered iPhone or iPad to a Mac running iTunes.

2. Double-click your iOS .ipa file.
3. You should see your device in the upper left corner of your iTunes windows. Click on it.
4. Click the **[Apps]** button. Now you should see your app in the iTune apps list, with an Install button. Click it.
5. The Install button changes to Will Install.
6. Click the **[sync]** button in the lower-right corner to sync your device. This installs your app on your device.

Your other testers can now install and test your app on their registered iPhones and iPads just like any other app from iTunes.

If you have the Enterprise Apple developer account, there is no limit on the number of testing devices, and they do not have to be registered.

Getting Crash Reports From Testers

iOS automatically records crash logs when apps crash. Your testers can retrieve and send these logs to you. They must follow these steps:

1. Connect the testing device to a Mac computer running iTunes.
2. The crash logs are automatically downloaded to `~/Library/Logs/CrashReporter/MobileDevice`
3. Attach the relevant log files to email and send them to you.

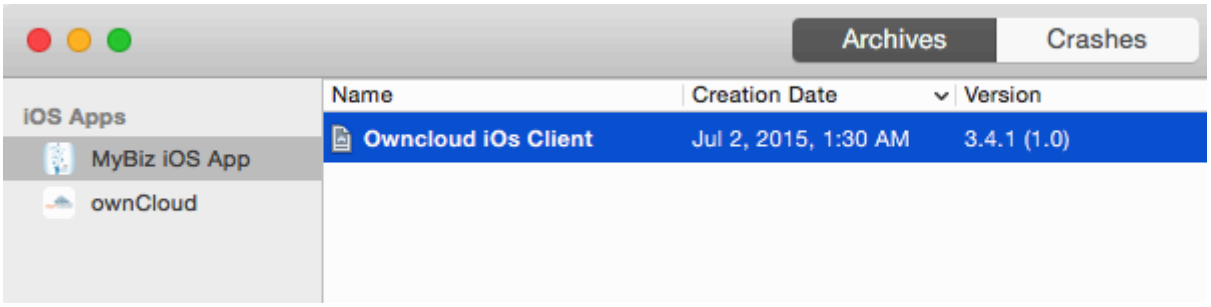
Publishing Your New Branded iOS App

Publish for General Distribution on iTunes

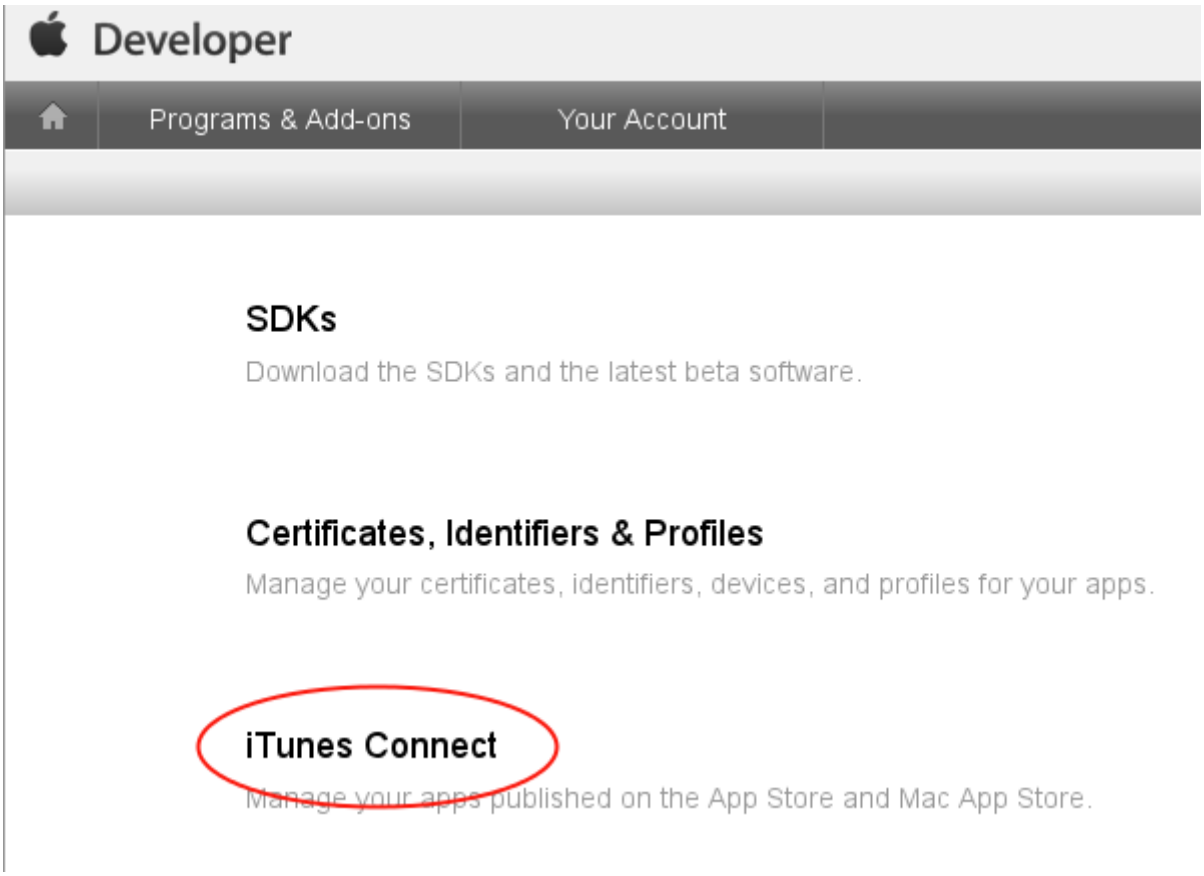
At last, after following all the previous steps and passing beta testing, your branded iOS app is ready to publish for general distribution on iTunes. You need a Mac computer with Xcode installed (Xcode is a free download), and you need the eight provisioning profiles (4 Ad Hoc and 4 Apple Store) and p12 file that you created copied to the same computer that you are using to upload your app to iTunes. You will also need a number of screenshots of your app in specific sizes and resolutions, which are detailed in your iTunes Connect setup screen.

Apple must review and approve your app, and the approval process can take several days to several weeks.

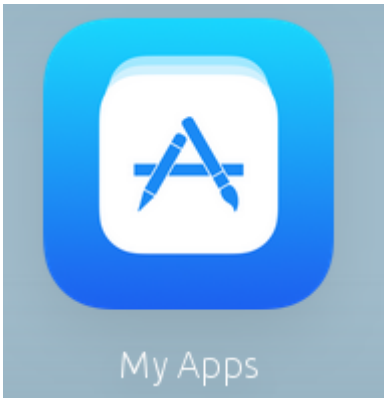
Download the `xcarchive.zip` file from your account. Your friendly macOS computer will automatically unpack it and change the name to something like `ownCloud iOS Client 02-07-15 10.30.xcarchive`. Double-click on this file to automatically install it into Xcode. Go to Xcode and you will see it in the Archives listing under **Window › Organizer**.



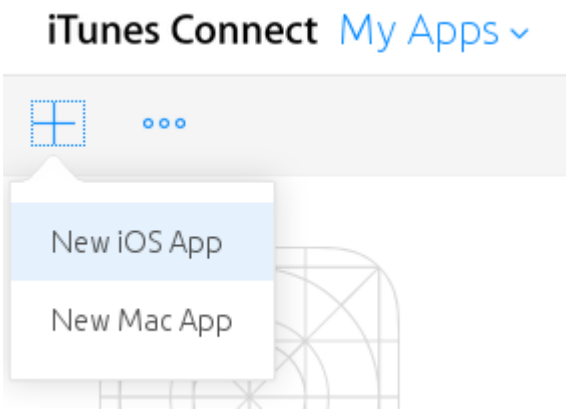
Next, go back to the Apple Developer Member Center to log into iTunes Connect to set up your app storefront.



After logging in click the blue **[My Apps]** button. This takes you to the main screen for managing your apps on iTunes.



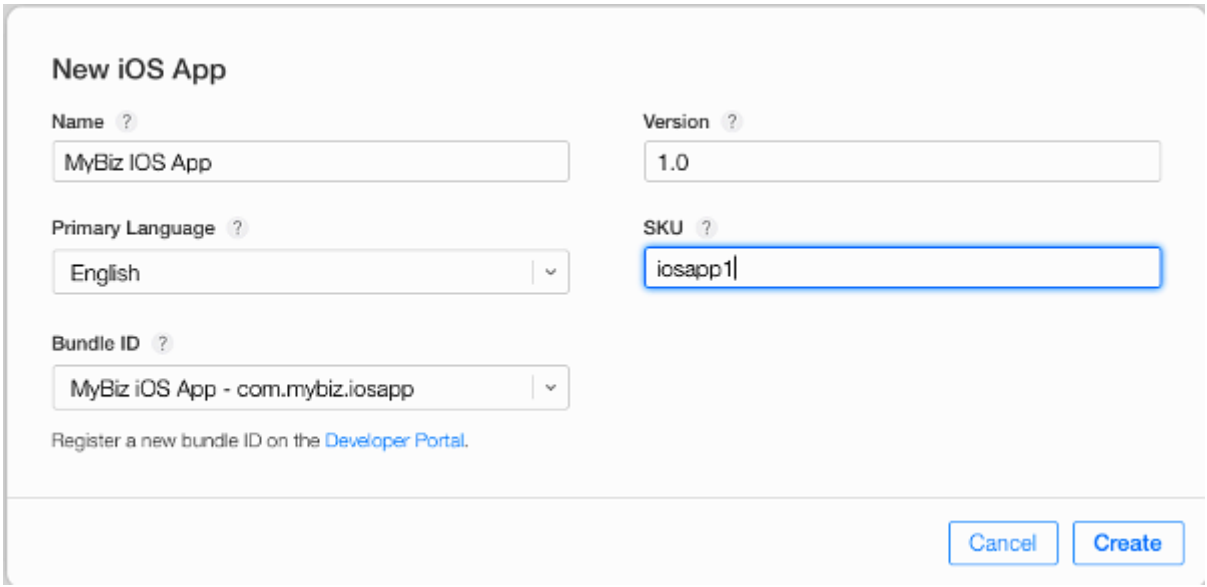
Click the plus button on the top left to setup your new branded iOS app.



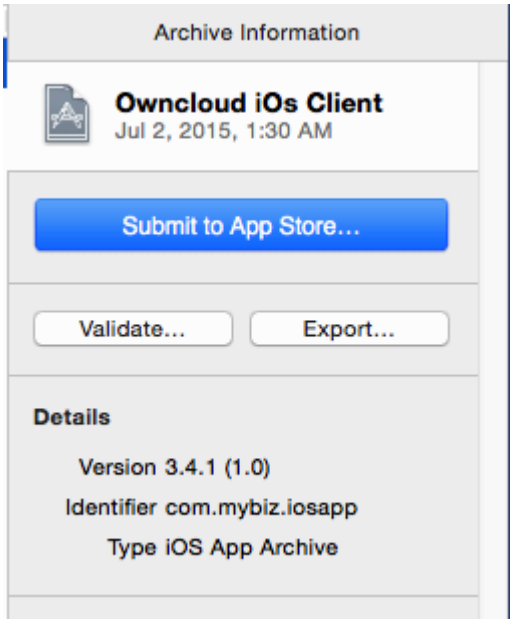
This opens a screen where you will enter your app information. Make sure you get it right the first time, because it is difficult to delete apps, and Apple will not let you re-use your app name or SKU.

- Enter any name you want for your app. This is the name that will appear in your App Store listing.
- Choose your primary language.
- Select the bundle ID from the drop-down selector.
- Enter your app version number, which should match the version number as it appears in your Xcode organizer.
- The SKU is a unique ID for your app, and is anything you want.

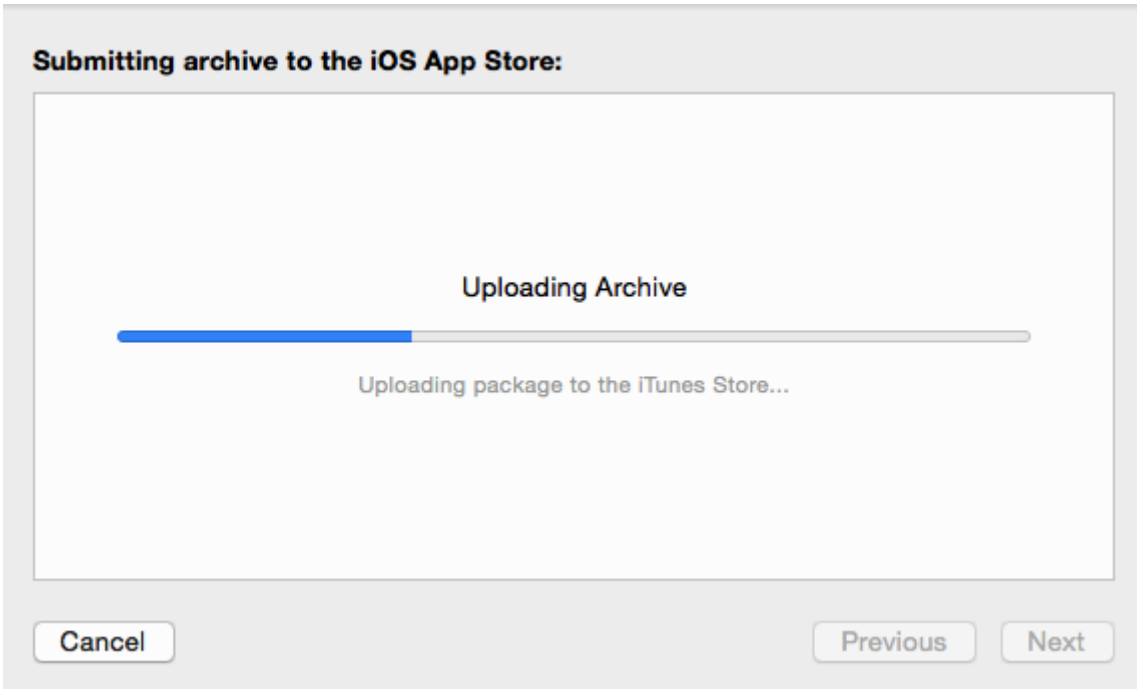
Then click the **[Create]** button.



Now go back to your Xcode organizer to upload your app; click the blue **[Submit to App Store]** button.



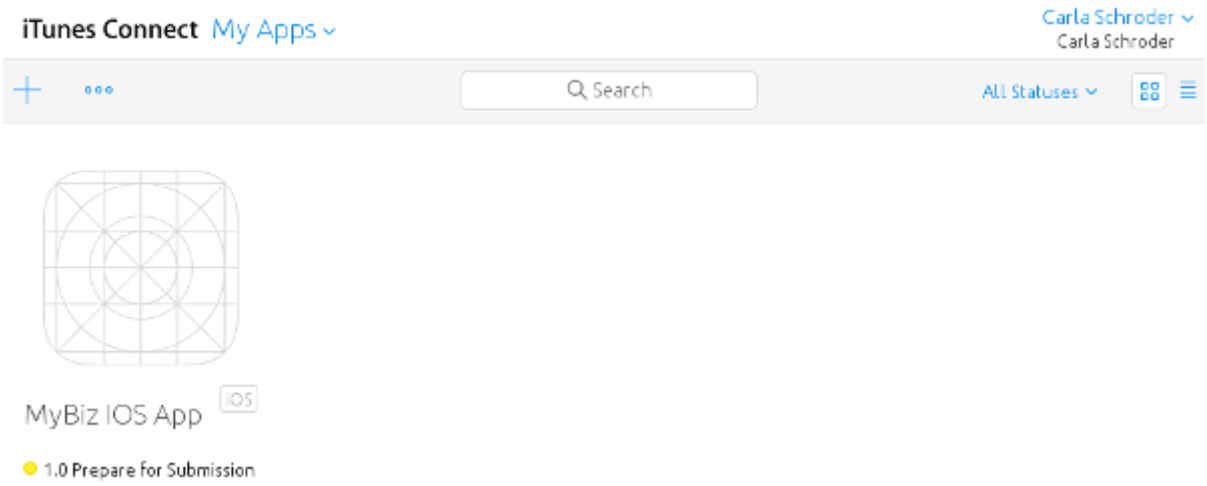
This takes a few minutes as it verifies your bundle ID and certificates, and then you will see an upload status.



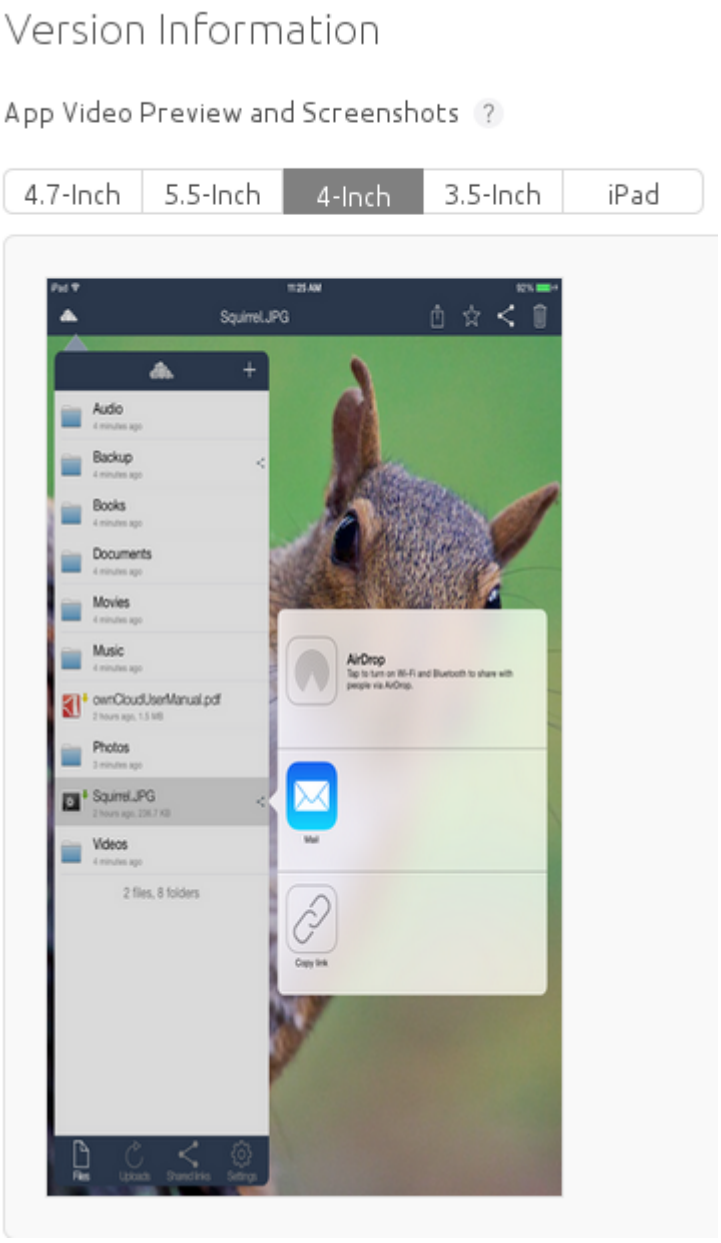
At long last, after working through this long complex process, you are almost ready to publish your app on iTunes.

Setting up Your iTunes Storefront

There are just a few steps remaining. Now that you have uploaded your branded iOS app, you need to upload some screenshots, an optional demo video, and fill in some information for your app listing on your iTunes storefront. You should see something like this on your main screen (figure 8). You should click the **[Save]** button at the top right periodically to preserve your changes.



This screen displays all of your apps and their submission status. Click **[Prepare for Submission]** to get started on the submission process. The first screen is for entering screenshots of your app for various devices, and optionally a demonstration video. Click the little question marks to learn the required image specifications.



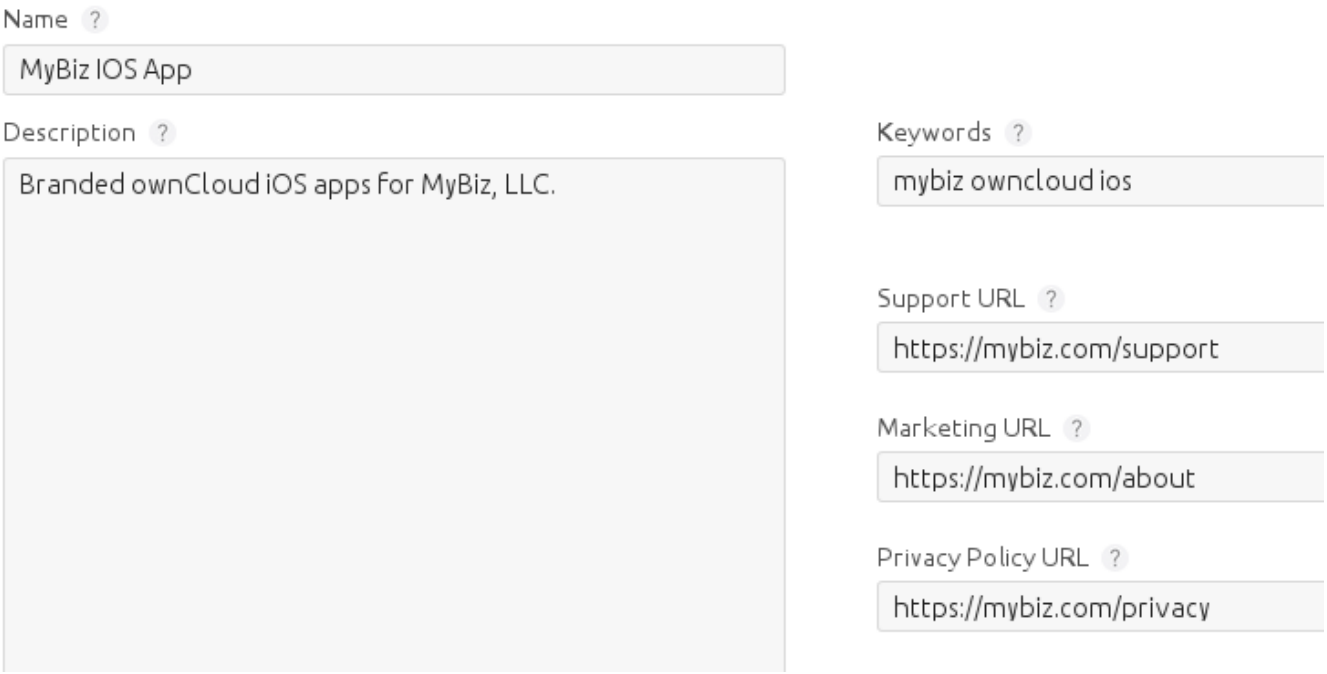
Apple simplified the screenshot submission process. Please check this Video (in Safari) for details.

For the ownCloud client, we also don't use real screenshots, we use frames in different sizes instead. You can find templates to generate those assets. Here are examples for the Sketch app:

- https://github.com/LaunchKit/SketchToAppStore
- https://github.com/MengTo/AppStoreSketch

Then you must enter:

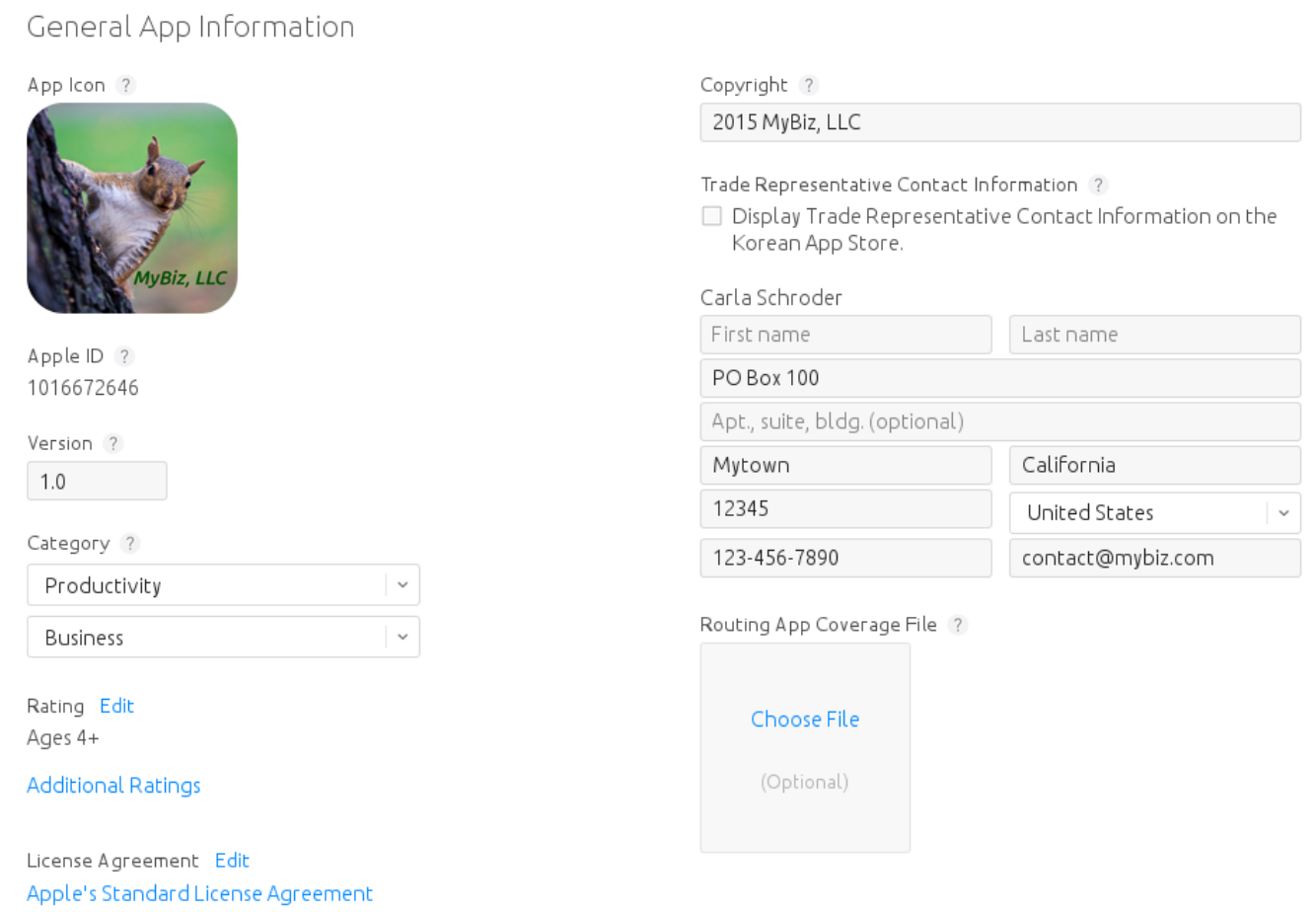
- Your app name
- A description
- Some keywords for iTunes searches; and
- Some optional URLs



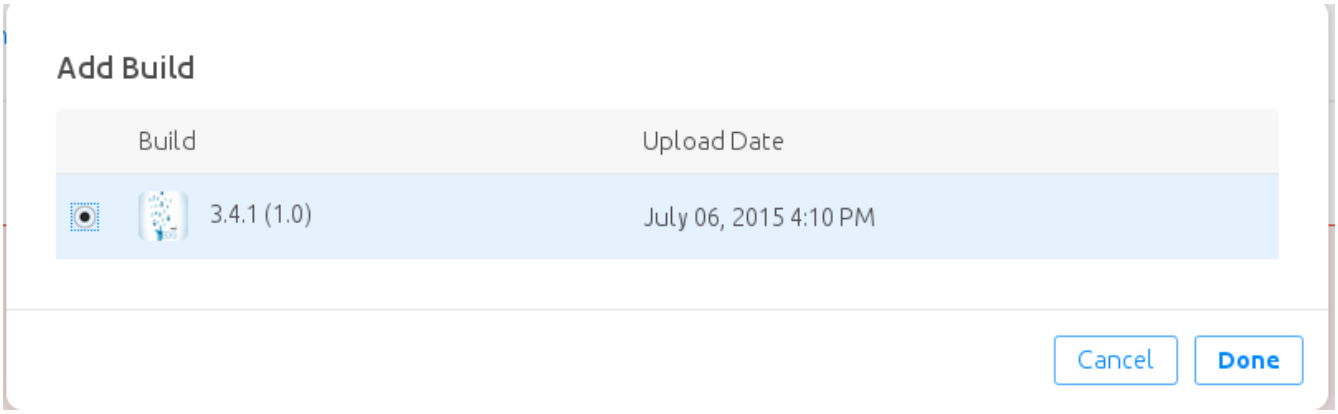
The next section is for Apple Watch. If you don't support Apple Watch you can skip this.

The **General App Information** section requires a:

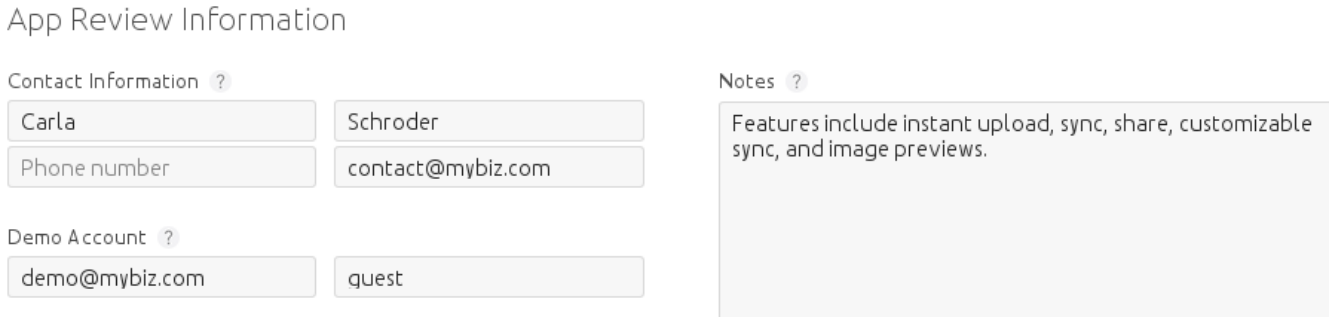
- 1024 x 1024 logo
- Version
- Rating
- Category
- License
- Copyright, and
- Contact information



In the **Build** section, click the plus button and select your app.



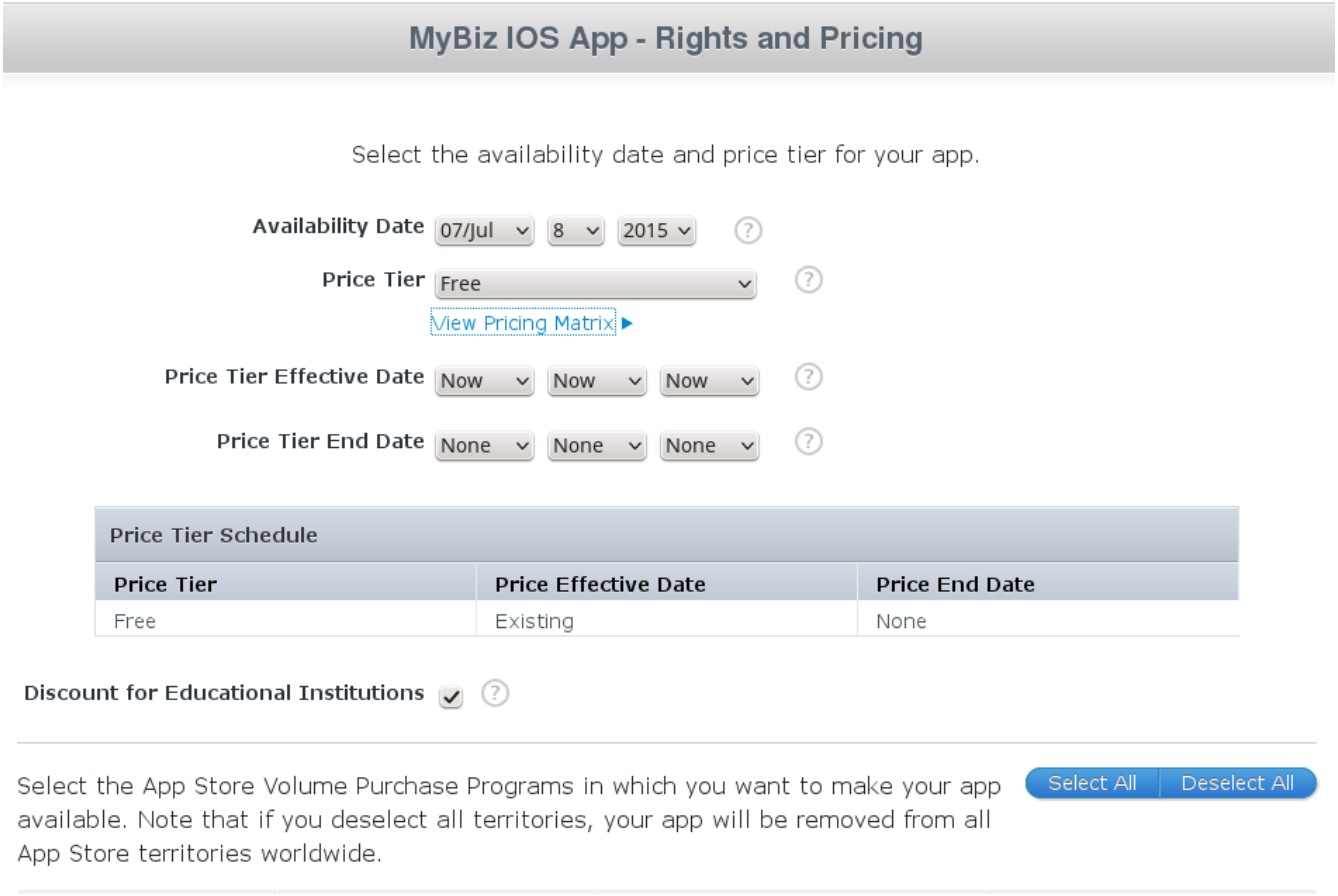
The **App Review Information** requires contact information, and some information about your app to guide reviewers. Remember, everyone on iTunes can review your app, so it's in your best interest to be helpful. You may optionally provide a login for a demo account.



The **Version Release** section allows you to choose between automatic release, which means your app will be published upon approval, or manual release, where you must release your app after it is approved.

Pricing

Next, you must go to the **Pricing** page to set your price, and to select the territories you want your app to be available in.



Submit For Review

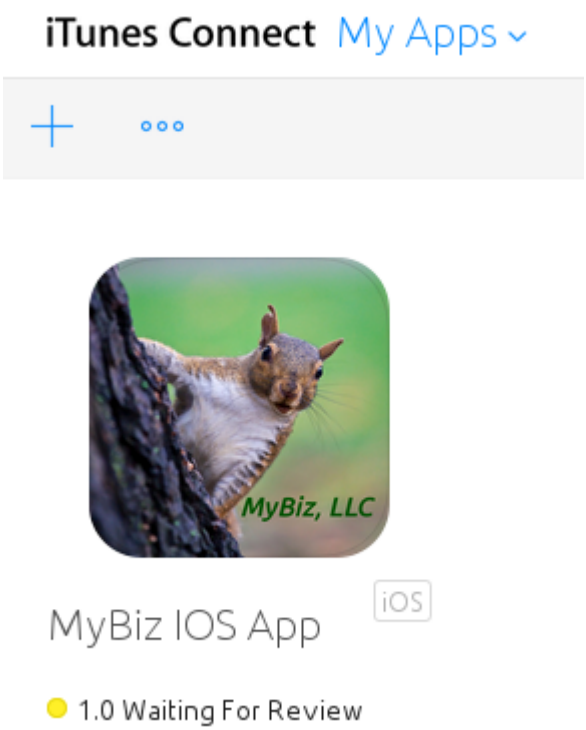
When you have filled in all the required forms and provided the required screenshots, click **Save** and then **Submit for Review**. If anything needs to be corrected you will see messages telling you exactly what must be fixed.

The next screen is legalese; click the appropriate Yes or No boxes, and then click the **Submit** button.

You are now finished. No really, you are. When you return to your **My Apps** page you'll see that the status of your app has changed to "Waiting for review". In a few days, or perhaps many days, your app will either be approved or rejected. If it is rejected Apple will tell you what you need to do to get it approved.

FAQ

Here are the most common answers to questions from the iOS App Review Team.



When, at last, it is published on iTunes you may distribute the URL so that your users may install and use your app.

FAQ iOS App Review Team

Information from Apple: https://developer.apple.com/support/app-review/

The product contains cryptography, and whether it classifies for export exemptions.

No, the product does not contain cryptography. Although the app is ready to connect via SSL, this does not imply that the app includes any cryptography

How does the app utilize Document Picker and File Provider extensions?

The ownCloud app takes advantage of the Document Provider extensions so that those apps that act as Document Picker may access to the ownCloud data, edit it and then changes are automatically uploaded back to the ownCloud server.

Background Audio

Questions:

- What is the purpose of declaring Audio background mode? Please explain the need for this background mode and where the usage can be found in your binary.
- Your app declares support for audio in the UIBackgroundModes key in your Info.plist but did not include features that require persistent audio. The audio key is intended for use by applications that provide audible content to the user while in the background, such as music player or streaming audio applications. Please revise your app to provide audible content to the

user while the app is in the background or remove the "audio" setting from the UIBackgroundModes key.

Answer:

Sometimes, usually, the first time the ownCloud app is submitted, it is rejected because it is included the background mode, Apple rejected it because in the past some apps used this trick to avoid the app to be fully closed. Howerver, the ownCloud app used it only when music is played. This may be checked by Apple reviewers, what we suggest is to be proactive, instead of waiting for the app to be rejected because of that, adding an explanation line, something such as: You may notice that the app is ready to play music not only in foreground but also in background, for you to test it we have uploaded to the test account the file XXX

Content Rights - Does your app contain, display, or access third-party content?

If the branded app has the help option enable, the answer is yes. Within the help, we are having access to an external web Otherwise, no

Does this app use the Advertising Identifier (IDFA)?

No, no ads at all

IPv6 Connectivity

Question:

> We discovered one or more bugs in your app when reviewed on the iPad and the iPhone running iOS 10.2 on Wi-Fi connected to an IPv6 network - Specifically, the app does not connect to the server.

Information from Apple: https://developer.apple.com/library/content/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/UnderstandingandPreparingfortheIPv6Transition/UnderstandingandPreparingfortheIPv6Transition.html

Here you can check your server for IPv6 connectivity: http://ipv6-test.com/validate.php

Business questions from Apple

- Does your app access any paid content or services?
- What are the paid content or services, and what are the costs?
- Who pays for the content or services?
- Where do they pay, and what's the payment method?
- If users create an account to use your app, are there fees involved?
- How do users obtain an account?

This is a standard question Apple has to avoid iTunes circumvention as for some stuff they want the 30% revenue share. (see In-App Purchase: https://developer.apple.com/in-app-purchase/)

Building Branded Android Apps

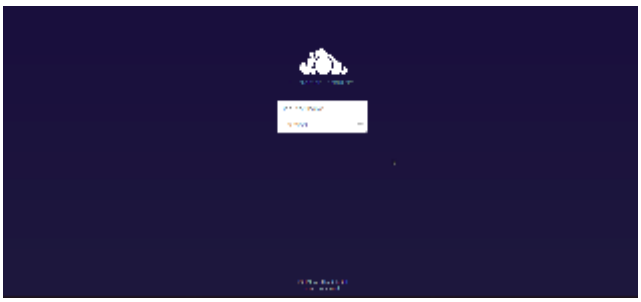
Building Your App With ownBrander

Introduction

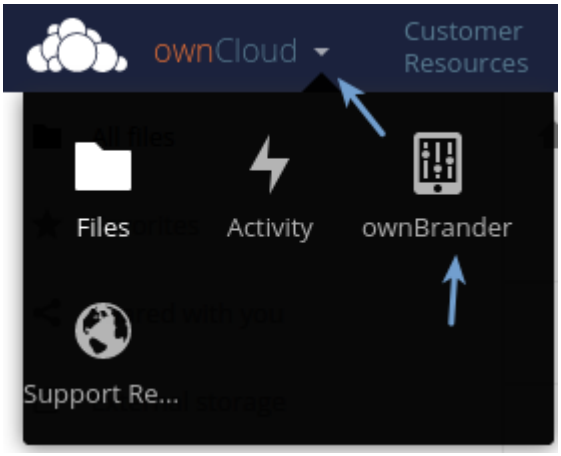
Follow along and begin creating your ownBrander Android app. Within the ownBrander you will find that most of the fields that require your input are self-explanatory. If you still have questions, or perhaps suggestions, feel free contact us.

Use your credentials to log in to customer.owncloud.com.

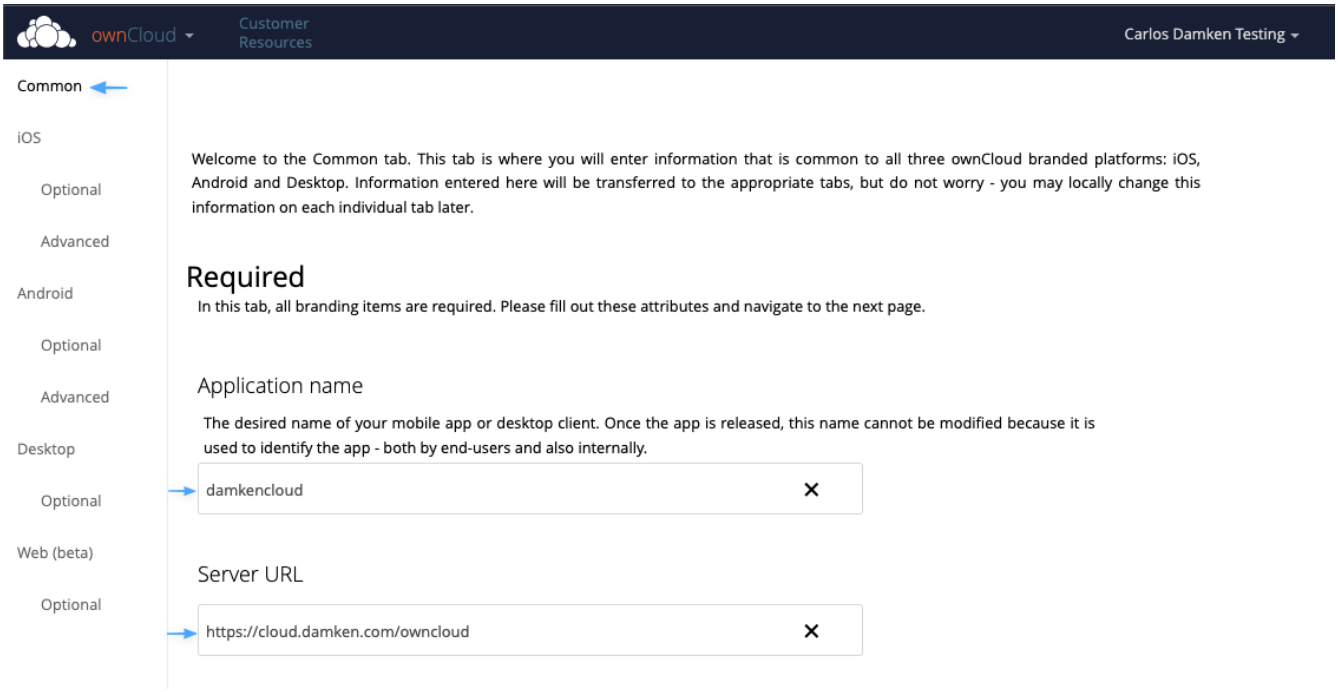
If you need credentials, do not hesitate to contact your account sales representative.



After successfully logging in to your account, navigate to the left side of your ownCloud instance and click on the inverted arrow to open the menu. Then click the ownBrander icon to open it.



Begin on the **Common Tab** and navigate to the **Required** area. Next, enter your application name and the URL of your ownCloud server in the corresponding fields.

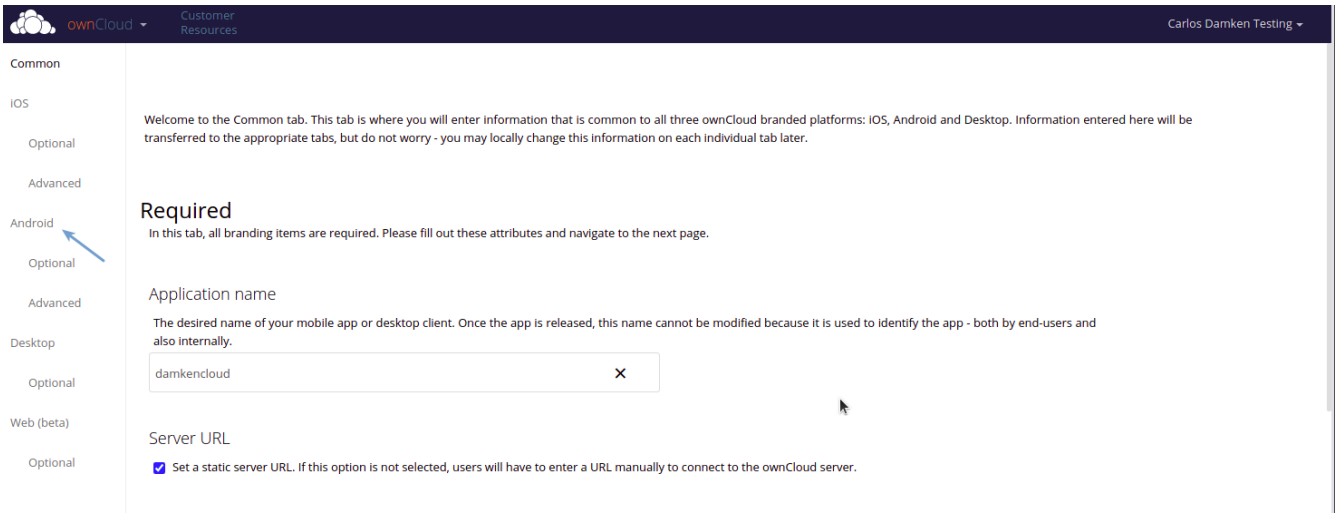


These entries define your global defaults for all of the platforms in ownBrander. You can change them when you create your apps.

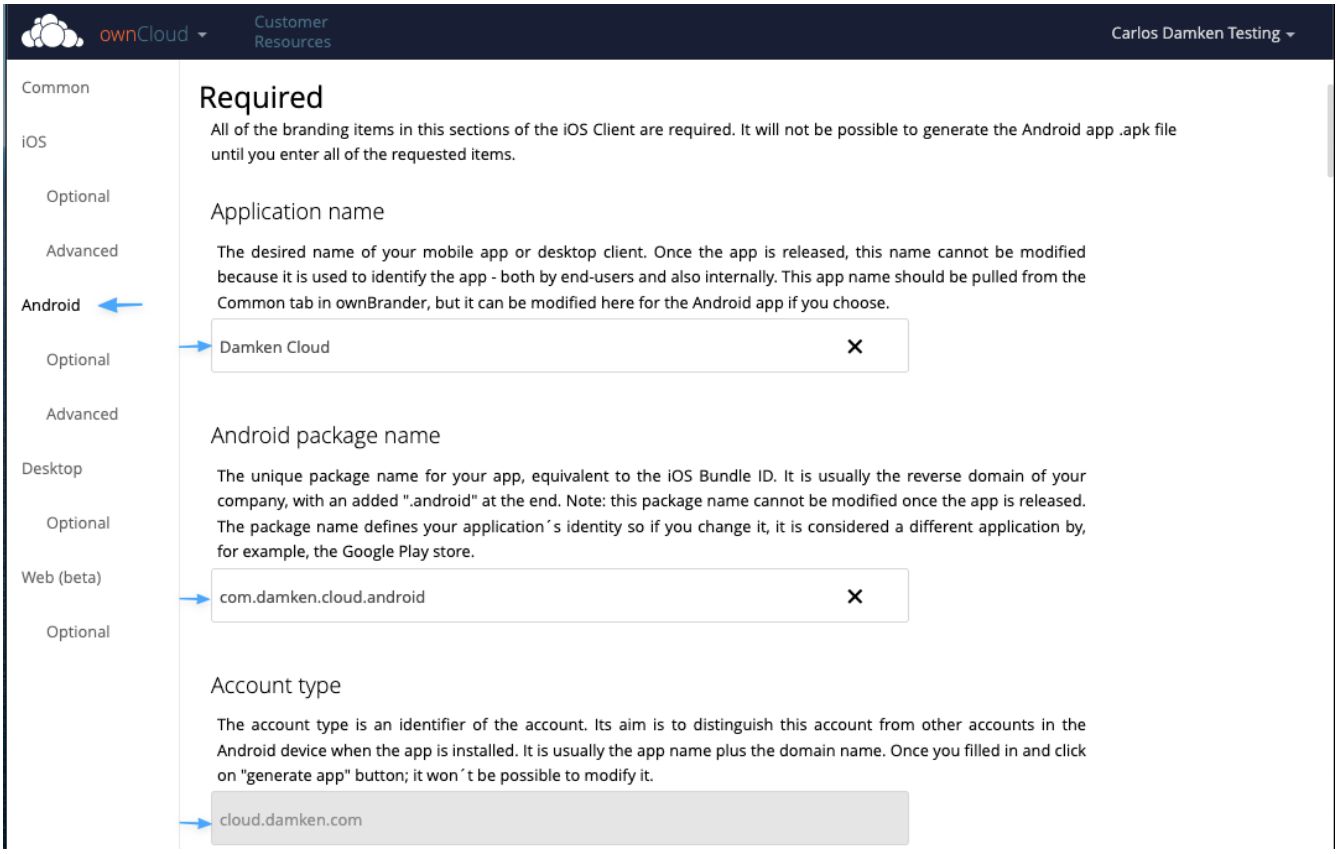
When you create production apps, you must then use your real app name, and the URL must point to your real ownCloud server. (However, for testing purposes, these values can be anything.)

Configuring ownBrander Parameters

Next, click on the **Android Tab**. Here you will find another **Required** area to fill out. You will also see two additional tabs: **Optional**, and **Advanced**. These are not mandatory for the processing of your branding experience.



Start in the **Required** section and type in your application name, your Android package name and your account typ.

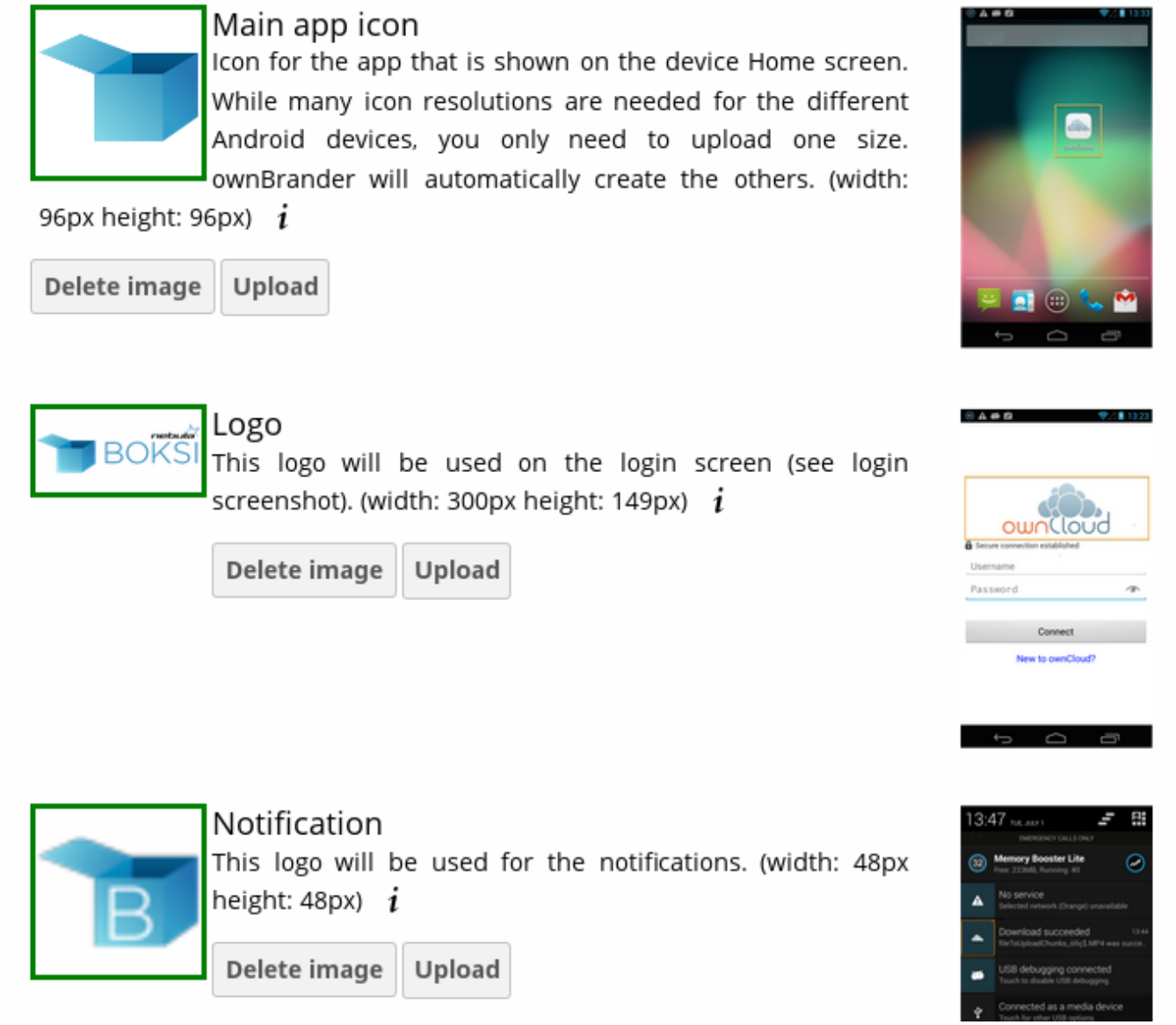


Move on to styling your app.

Wizard for your Branding Images

Building an Android app requires just a few images (preferably PNG format). The wizard also provides you with the exact dimension requirements. Please make certain that your images fulfill the specifications.

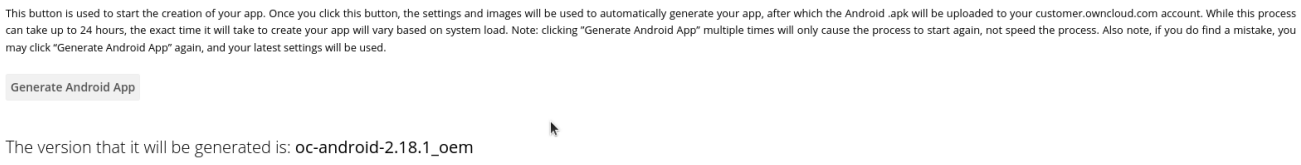
Now you can upload your images into the pre-defined boxes.



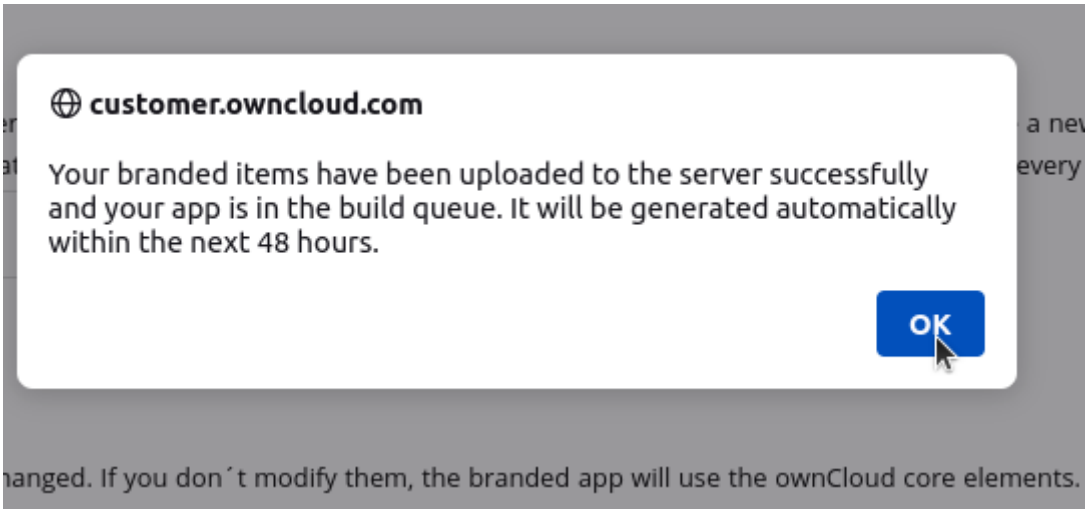
Proceed to the section regarding the signing of your Android client and click on the checkbox if you want this option. Provide the required information for the three fields (Key Store Password, Key Alias and Key Alias Password.) You will need these credentials in the Google Play Store Console.

Generating Your App

After a first-time ownCloud branding or after performing any modifications (updating your app), click the **Generate Android App** button.



The following information will pop up - click ok.

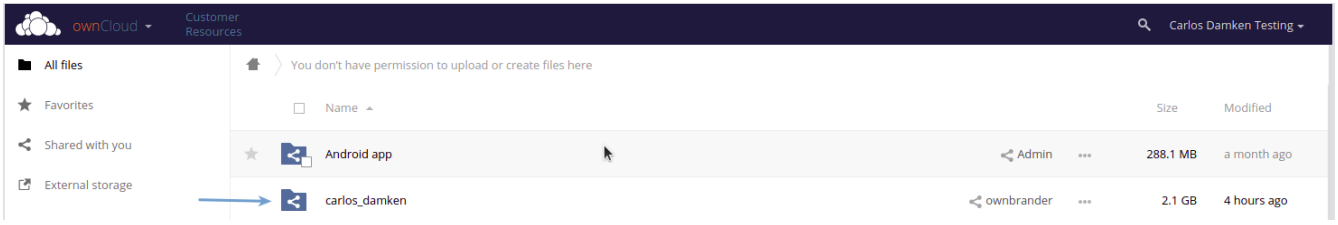


Download Your Branded App

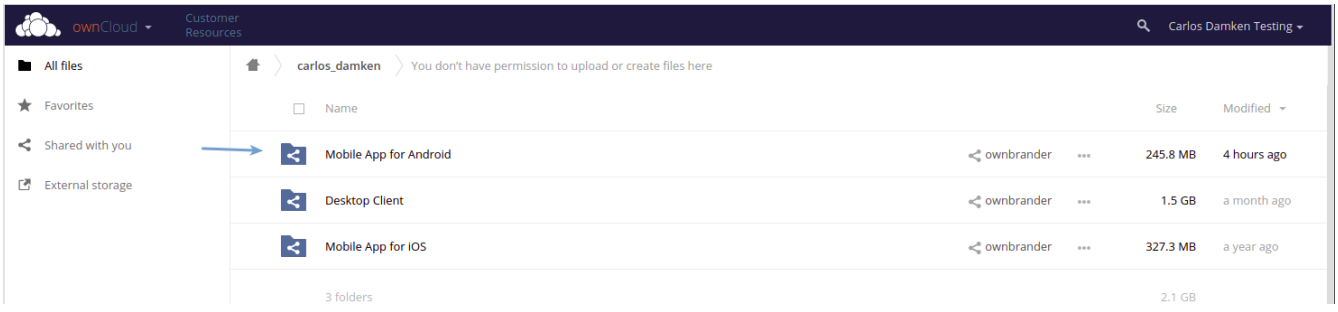
After the above mentioned time frame, open your account at: customer.owncloud.com

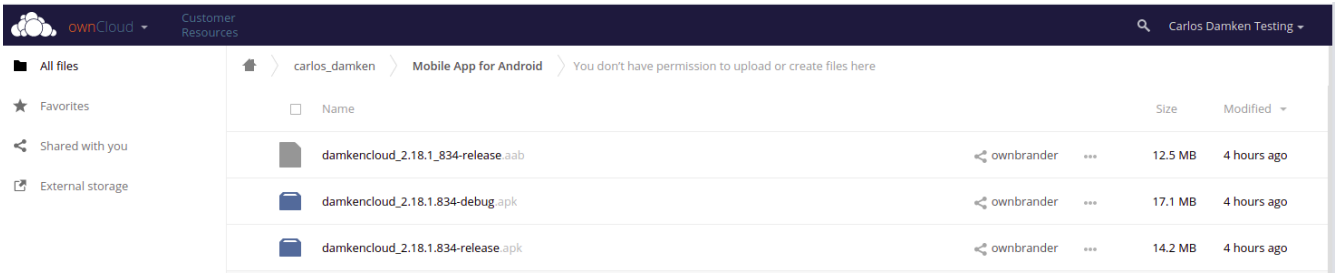
The ownBrander produces three files required to build your app. These are automatically uploaded into your personal folder.

Choose the folder with your name to find the folders for your various branded clients.



Choose the **Mobile App for Android** folder. This folder should contain the three files seen in the image below. The .aab file is of relevance for your app in the Google Play Console.





Now you must sign your app in the Google Play Store in order to distribute it to your users.

Distributing Your Branded Android App

Introduction

Now that you have created your branded Android app with ownCloud's ownBuilder service (building_branded_android_client) how do you distribute it to your users? There are multiple ways: email, publish_server, or publish_google_play. However you distribute it, the first step is to digitally sign your new app. Signing your app verifies authorship and authenticity.

When you create your branded Android app we supply you with two `.apk` files: one for debugging and testing, and one for deployment, like these examples:

```
acmecloud_2.0.0-debug.apk
acmecloud_2.0.0-release-unsigned.apk
```

The second `.apk` file, `acmecloud_2.0.0-release-unsigned.apk`, is the one you will sign and distribute.

Digitally Signing Android Apps

Signing your app is required. You can do this in the ownBrander wizard <sign_android_app>, or after it is built and delivered to you. The most time-consuming part of signing the built app is installing the commands you need to sign it. You need three commands to sign your app: `keytool`, `jarsigner`, and `zipalign`. Follow these steps:

1. Install the signing commands
2. Create a self-signed certificate with `keytool`
3. Use `jarsigner` to sign the app, and to verify signing
4. Use `zipalign` to optimize your app

You only need to create a certificate once, and then use it to sign all of your branded ownCloud apps. If you publish your apps on Google Play they must all be signed with the same certificate.

Installing the App Signing Tools

`keytool` and `jarsigner` are in Java runtimes. Linux users can get these in OpenJDK. For example, on current versions of Debian, Mint, and Ubuntu Linux you need to install two packages. The first one supplies `keytool` and the second one supplies `jarsigner`:

```
sudo apt-get install openjdk-8-jre-headless
sudo apt-get install openjdk-8-jdk
```

Plus some additional 32-bit packages:

```
sudo apt-get install libc6-i386 lib32stdc++6 \
    lib32gcc1 lib32ncurses5-dev zlib1g:i386
```

On SUSE systems, install this package:

```
sudo zypper install java-1_7_0-openjdk-devel
```

It is simpler to get these on CentOS and Red Hat Enterprise Linux, as they have created some nice wrapper scripts around `keytool` and `jarsigner` that you can install standalone:

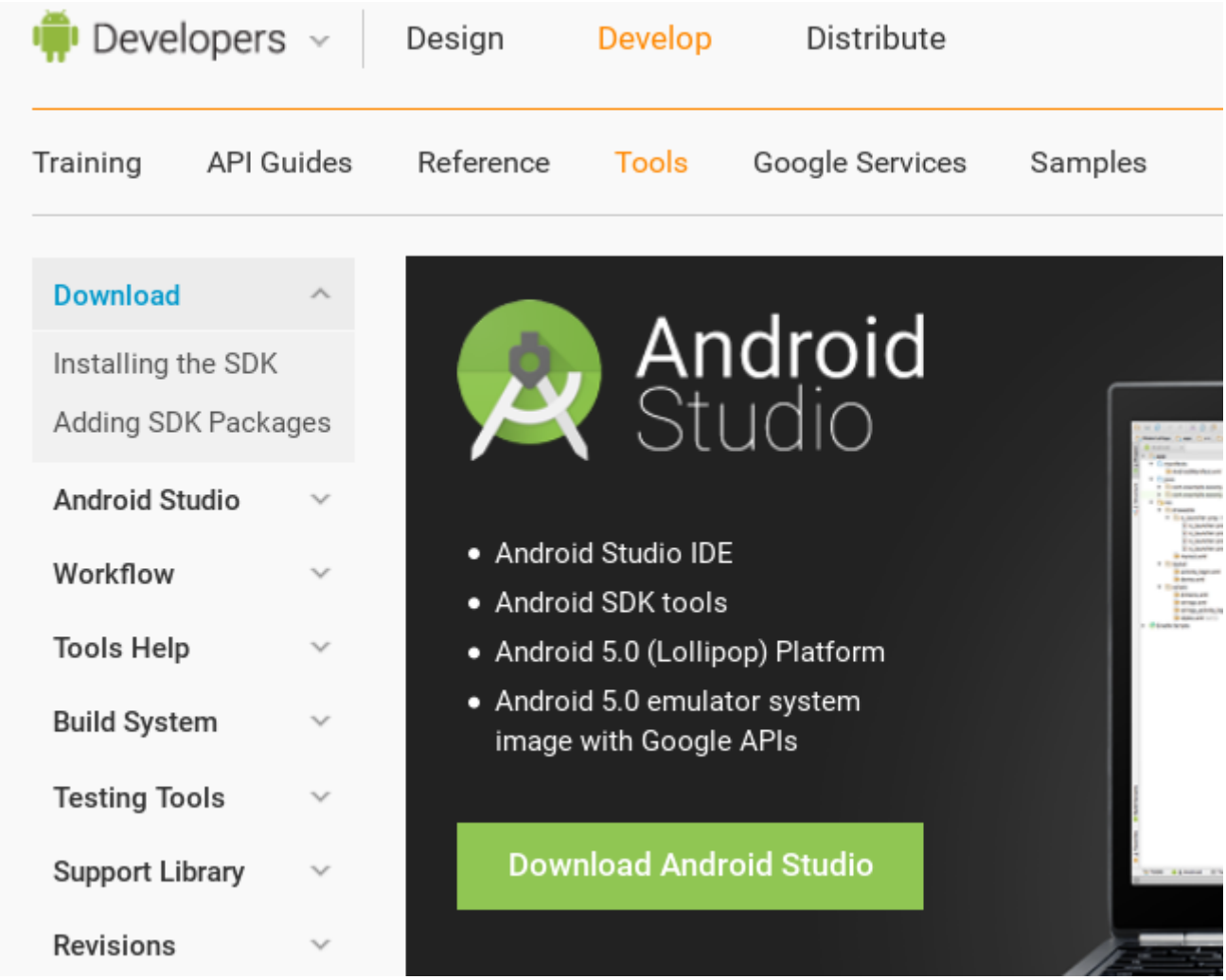
```
sudo yum install keytool-maven-plugin.noarch
sudo yum install maven-jarsigner-plugin.noarch
```

Mac OS X and Windows users can download the Oracle JDK from Oracle's Java Download page.

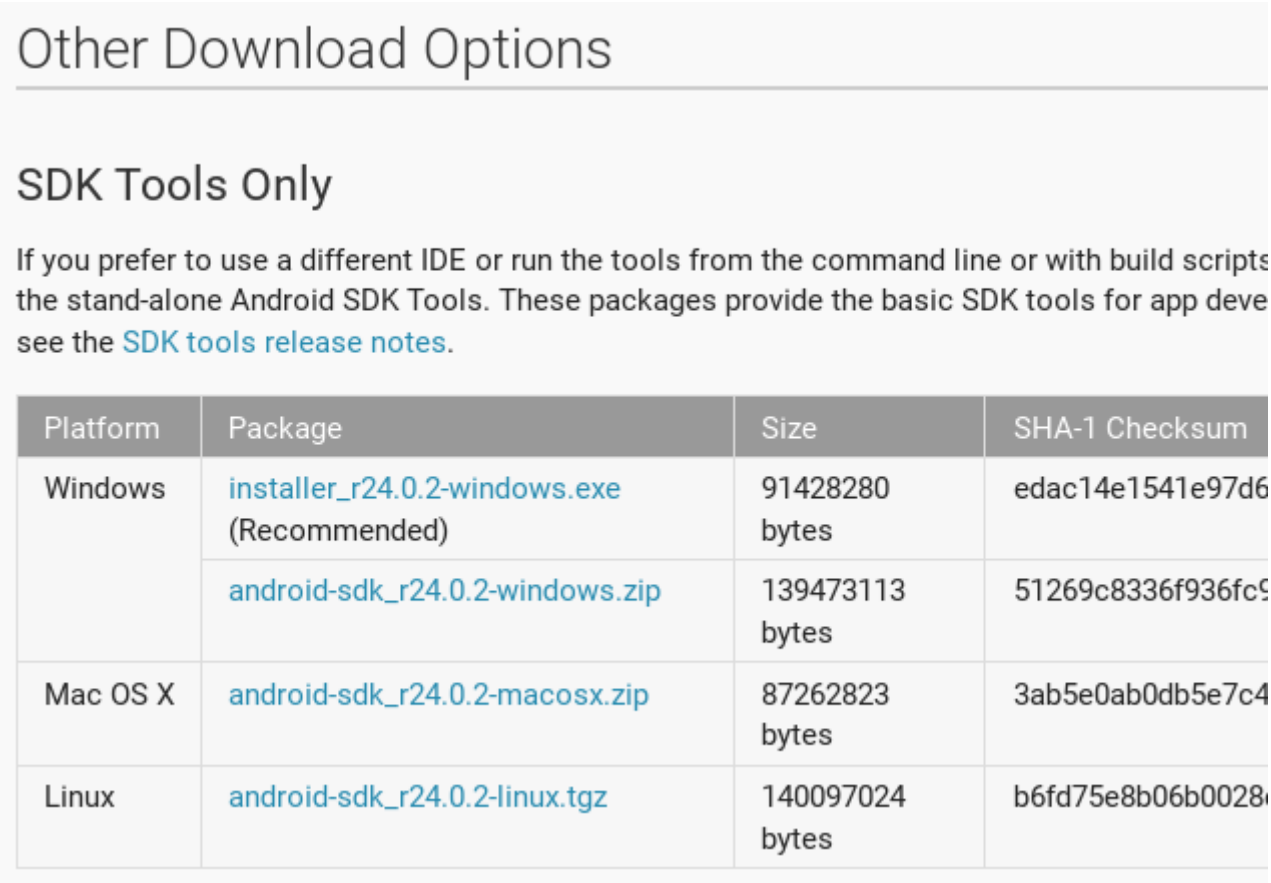
If your operating system provides the `zipalign` package, you can install it with:

```
sudo apt install zipalign
```

In case `zipalign` is not provided as installable package for your OS, you can download it from source via the Android Software Development Kit. It is a large download, but once you have downloaded it you can copy the `zipalign` binary to any computer and use it. Go to Android Software Development Kit and click the "Download Android Studio" button.



Download the appropriate **SDK Tools Only** package for your operating system.



Other Download Options

SDK Tools Only

If you prefer to use a different IDE or run the tools from the command line or with build scripts the stand-alone Android SDK Tools. These packages provide the basic SDK tools for app deve see the SDK tools release notes.

Platform	Package	Size	SHA-1 Checksum
Windows	installer_r24.0.2-windows.exe (Recommended)	91428280 bytes	edac14e1541e97d6
	android-sdk_r24.0.2-windows.zip	139473113 bytes	51269c8336f936fc9
Mac OS X	android-sdk_r24.0.2-macosx.zip	87262823 bytes	3ab5e0ab0db5e7c4
Linux	android-sdk_r24.0.2-linux.tgz	140097024 bytes	b6fd75e8b06b0028

Unpack it and change to the unpacked directory, which is `android-sdk-linux` on Linux systems, `android-sdk-macosx` on Mac systems, and `android-sdk-windows` on Windows systems. There is one more step, and that is to install additional tools. Run this command from the unpacked directory:

```
tools/android update sdk --no-ui
```

This will take some time, as it is a large download. When it's finished you'll find `zipalign` in the `build-tools` directory. For convenience, you could copy `zipalign` to your home folder or other location of your choice, and to any other computer without installing the whole Android SDK.

Digitally Signing Your App

After installing your signing tools, signing your app takes just a few steps. In these examples the name of the app, as supplied by ownBuilder, is `acmecloud_1.7.0-release-unsigned.apk`.

To create your certificate copy the following command, replacing `acme-release-key.keystore` and `acme_key` with your own keystore name and alias, which can be anything you want. The keystore name and alias must both have a password, which can be same for both. Then enter your company information as you are prompted:

```
keytool -genkey -v \
   -keystore acme-release-key.keystore \
   -alias acme_key \
   -keyalg RSA -keysize 2048 \
```

```
    -validity 10000
```

```
Enter keystore password:
Re-enter new password:
What is your first and last name?
 [Unknown]:  Acme Boss
What is the name of your organizational unit?
 [Unknown]:  Acme Headquarters
What is the name of your organization?
 [Unknown]:  Acme, Inc.
What is the name of your City or Locality?
 [Unknown]:  Anytown
What is the name of your State or Province?
 [Unknown]:  CA
What is the two-letter country code for this unit?
 [Unknown]:  US
Is CN=Acme Boss, OU=Acme Headquarters, O="Acme, Inc.", L=Anytown, ST=CA, C=US correct?
 [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA)
with a validity of 10,000 days
       for: CN=Acme Boss, OU=Acme Headquarters, O="Acme, Inc.", L=Anytown, ST=CA, C=US
Enter key password for <acme_key>
       (RETURN if same as keystore password):
[Storing acme-release-key.keystore]
```

Now use `jarsigner` to sign your app. Replace `acme-release-key.keystore` and `acme_key` with your own keystore name and alias:

```
jarsigner -verbose \
    -sigalg SHA1withRSA \
    -digestalg SHA1 \
    -keystore acme-release-key.keystore \
    acmecloud_1.7.0-release-unsigned.apk acme_key
```

```
Enter Passphrase for keystore:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/ACME_KEY.SF
  adding: META-INF/ACME_KEY.RSA
 signing: res/anim/disappear.xml
 signing: res/anim/grow_from_bottom.xml
 [...]
 jar signed.

 Warning:
 No -tsa or -tsacert is provided and this jar is not timestamped.
 Without a timestamp, users may not be able to validate this jar after the signer
```

```
 certificate's expiration date (2042-07-28) or after any future revocation date.
```

You can ignore the warning at the end; you should see a `jar signed` message when it is finished.

Now you can verify that your app is signed:

```
jarsigner -verify -verbose -certs acmecloud_1.7.0-release-unsigned.apk
```

```
sm       943 Thu Mar 12 12:47:56 PDT 2015
res/drawable-mdpi/abs__dialog_full_holo_light.9.png

X.509, CN=Acme Boss, OU=Acme Headquarters, O="Acme, Inc.", L=Anytown, ST=CA, C=US
```

This will spit out hundreds of lines of output. If it ends with the following it's good:

```
...
s = signature was verified
m = entry is listed in manifest
k = at least one certificate was found in keystore
i = at least one certificate was found in identity scope

jar verified.
```

The last step for preparing your `.apk` for release is to run `zipalign` on it. `zipalign` optimizes your file to use less memory. You must specify both an input and an output file, so this is good time to give your app a shorter name, and it should not say "unsigned". Our example file will be renamed to `acmecloud_1.7.0.apk`:

```
zipalign -v 4 acmecloud_1.7.0-release-unsigned.apk acmecloud_1.7.0.apk
```

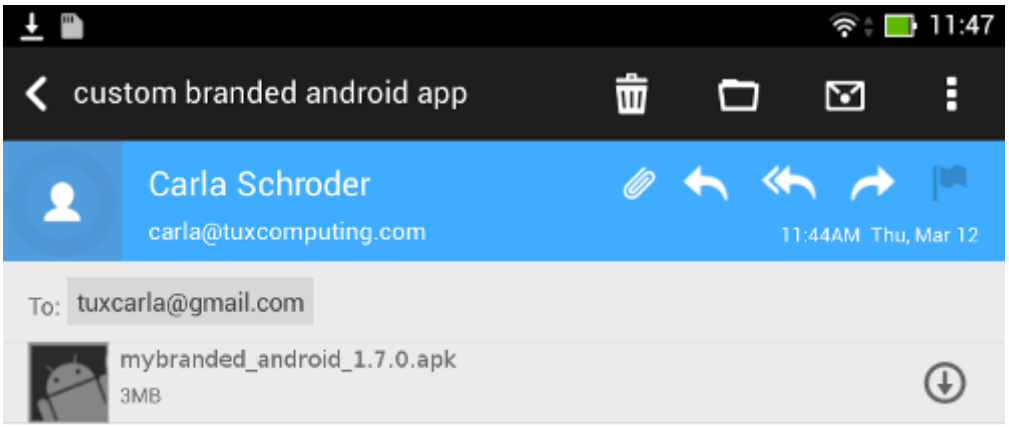
```
Verifying alignment of acmecloud_1.7.0.apk (4)...
     50 META-INF/MANIFEST.MF (OK - compressed)
  13277 META-INF/ACME_KEY.SF (OK - compressed)
  27035 META-INF/ACME_KEY.RSA (OK - compressed)
  28206 res/anim/disappear.xml (OK - compressed)
  [..]
  Verification succesful
```

Again, this emits a lot of output, and when you see **Verification succesful** at the end you know it succeeded, and it is ready to distribute.

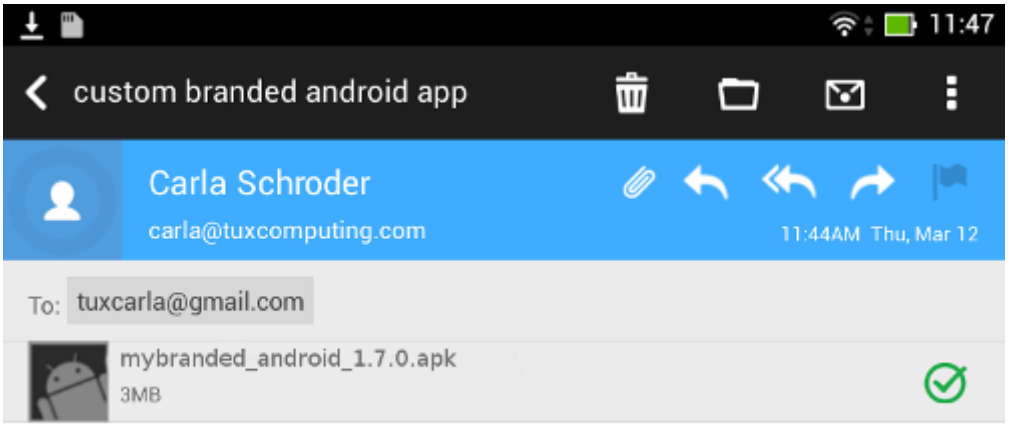
Distribution via Email

You can download your branded Android app from your account on customer.owncloud.com, and

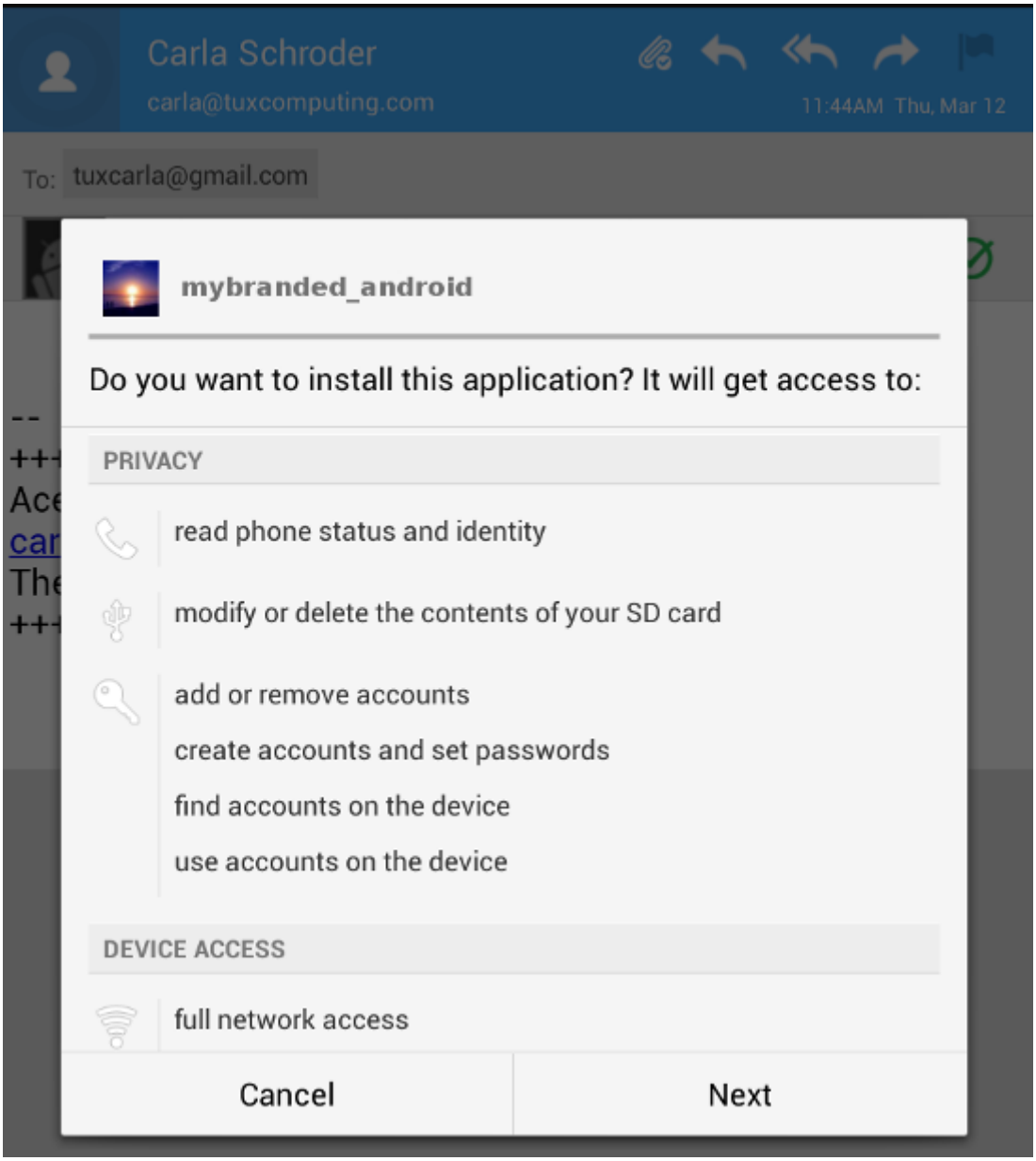
send it as an email attachment to your users. (This is not the optimal way to distribute it as it is over 2 megabytes in size.) When they open your email on their Android phone or tablet, they must first click the the download arrow (bottom right of the screenshot) to download your app.



When the arrow changes to a green checkbox, it has been downloaded.



Now your user must click on the green checkbox, and this launches the app installer, and all they have to do is follow the installation wizard to install your branded app.



When the installation is complete, the ownCloud Android App Manual contains instructions for using the app.

Publish On Your ownCloud Server

You can distribute your branded app from your ownCloud server. Simply upload it to your ownCloud server and share it like any other file: you can create normal ownCloud shares with ownCloud users and groups, and you may create a link share to share it with anyone. (See the **Sharing Files** section of the ownCloud Web Manual to learn more about sharing files.)

Publish to the Google Play Store

You may elect to publish your app in the Google Play store, either as a free or paid app. There are several steps to publishing a free app:

1. Create a Google Play Publisher account.
2. Sign your branded app with your own signing certificate.

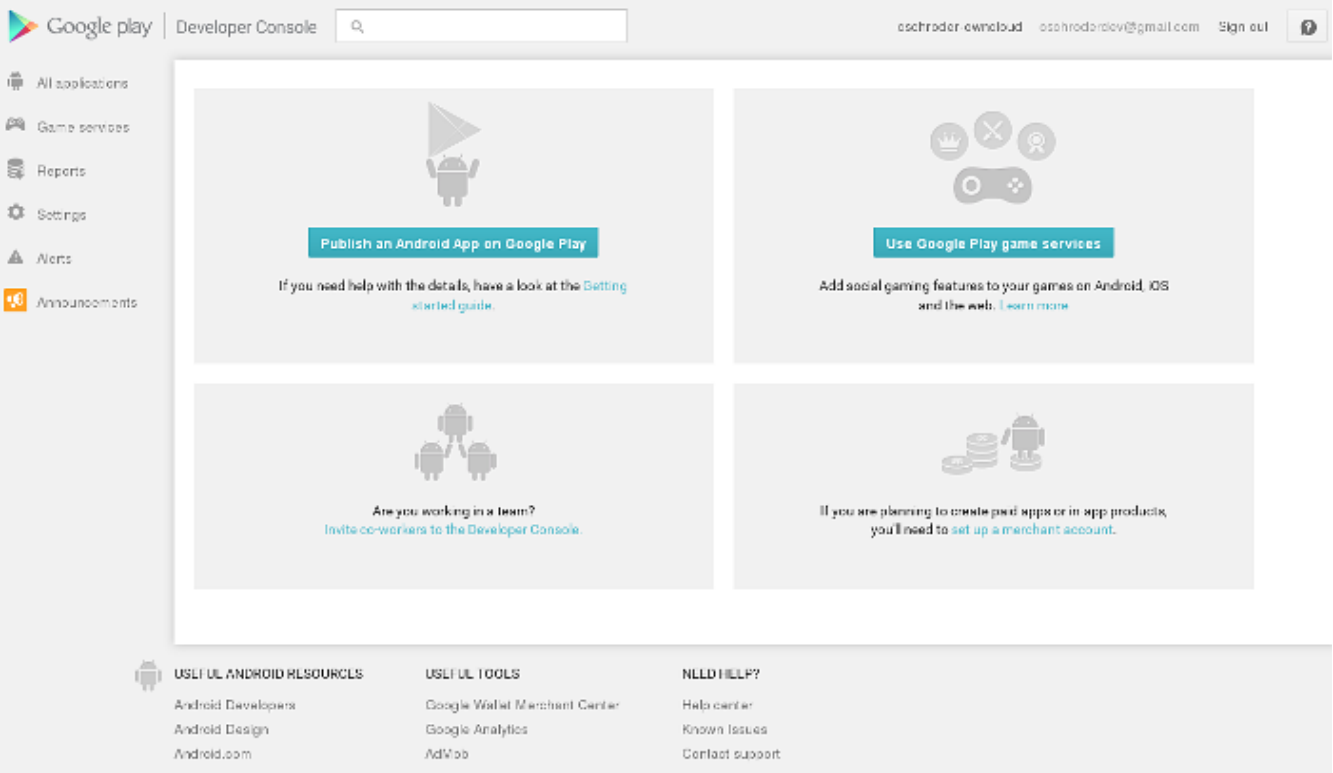
3. Upload your signed branded app to your Google Play Publisher account.

As part of creating your Google Play Publisher account you will have to create some screenshots of your app in specific sizes, and create a store description.

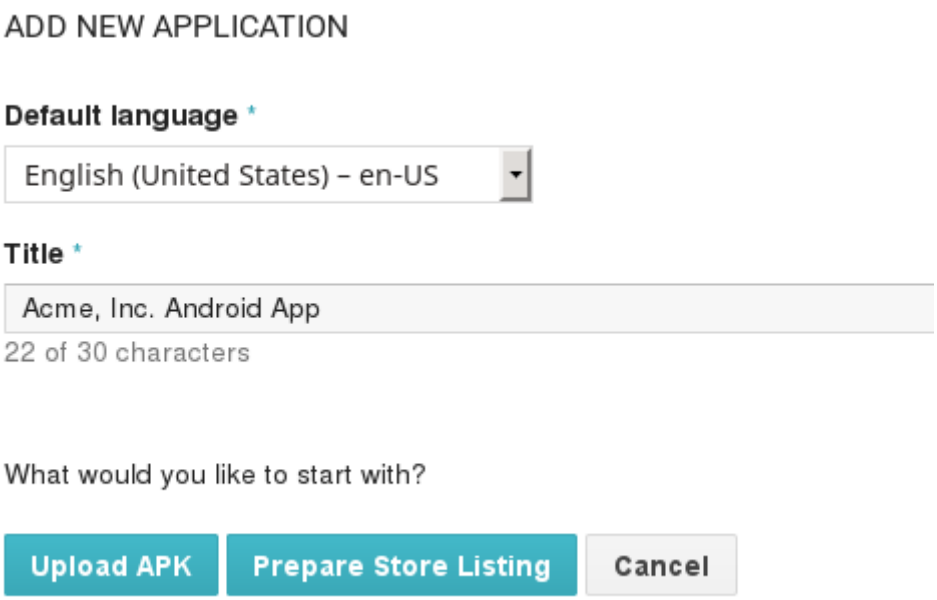
Create a Google Play Publisher Account

Start at Google's Get Started With Publishing page. Have a credit card ready, because it costs $25. If you already have a Google account, it is usually better to create a separate new account just for publishing apps to the Google Play Store.

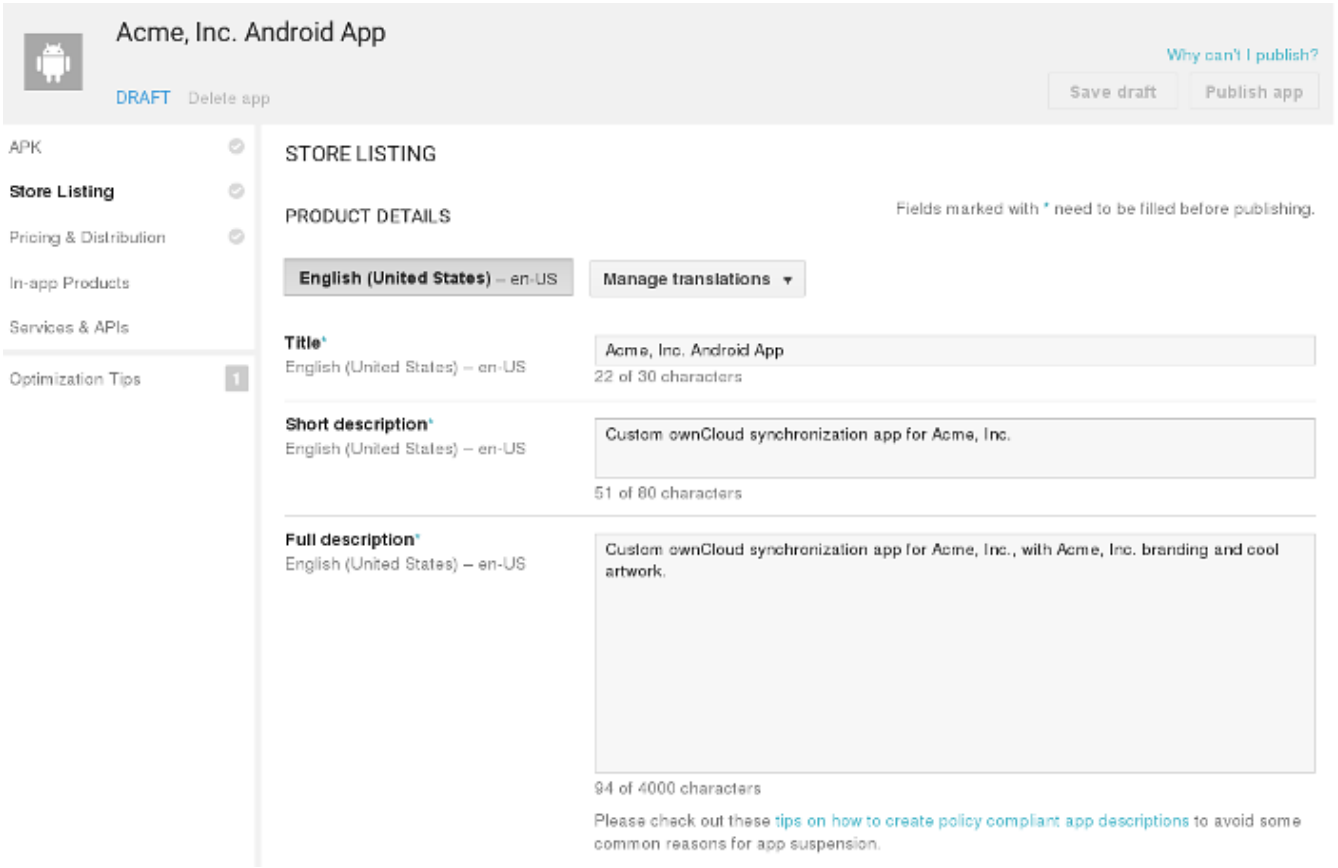
Google's process for uploading apps is fairly streamlined, and the most time-consuming task is creating all the required graphics. After registering, you'll see the welcome screen for the Google Dev Console. Click **Publish an Android app on Google Play**.



This opens the **Add New Application** screen. Click the **Prepare Store Listing** button. (Note that as you navigate the various screens, you can click the Save Draft button to preserve your changes.)



On the next screen, enter your product description.



Then you'll have to upload a batch of graphics in various sizes for the **Graphic Assets** section, like these images for a smartphone and seven-inch tablet. You are required to upload at least two images.

Screenshots *

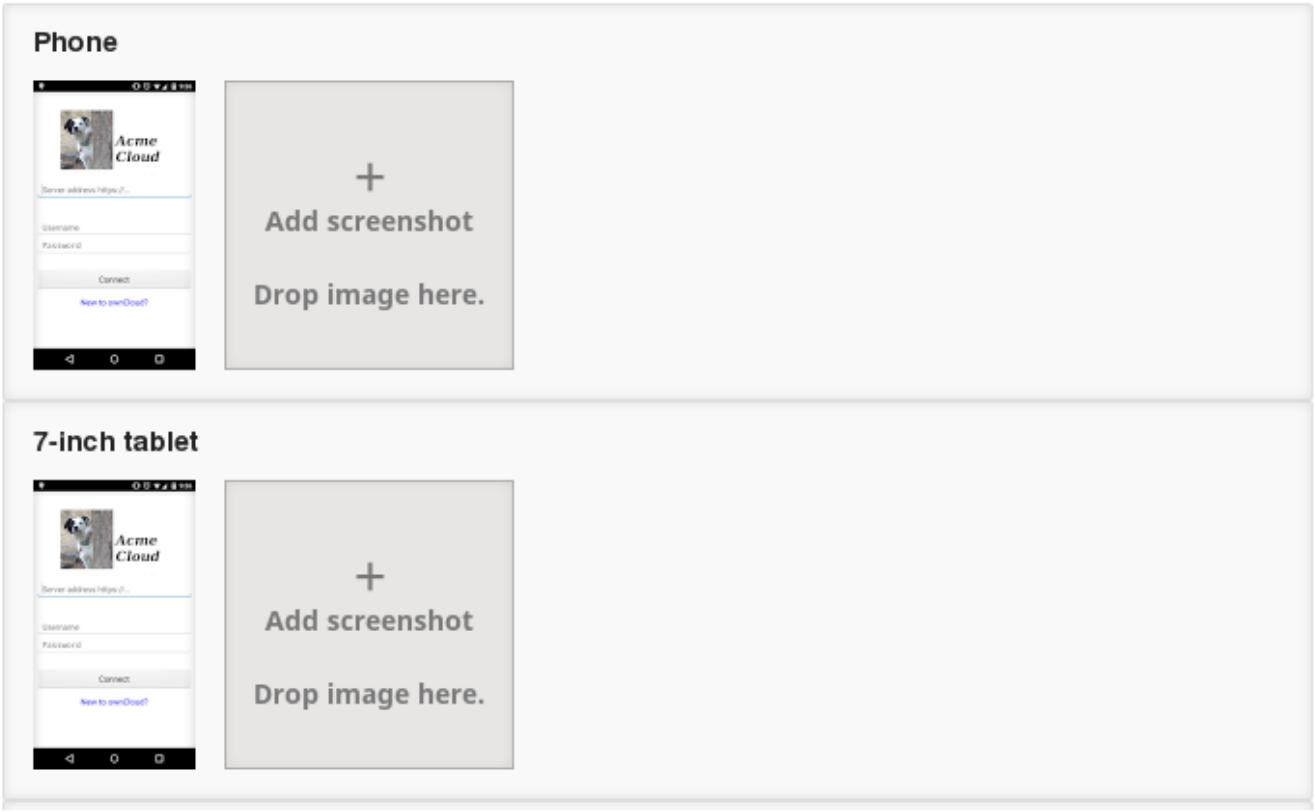
Default – English (United States) – en-US

JPEG or 24-bit PNG (no alpha). Min length for any side: 320px. Max length for any side: 3840px.

At least **2 screenshots are required** overall. **Max 8 screenshots per type**. Drag to reorder or to move between types.

For your app to be showcased in the 'Designed for tablets' list in the Play Store, you need to upload at least one 7-inch and one 10-inch screenshot. If you previously uploaded screenshots, make sure to move them into the right area below.

Learn how tablet screenshots will be displayed in the store listing.



You must also upload a 512x512-pixel logo, and a 1024x500 banner.

Hi-res icon *

Default – English (United States) – en-US

512 x 512

32-bit PNG (with alpha)

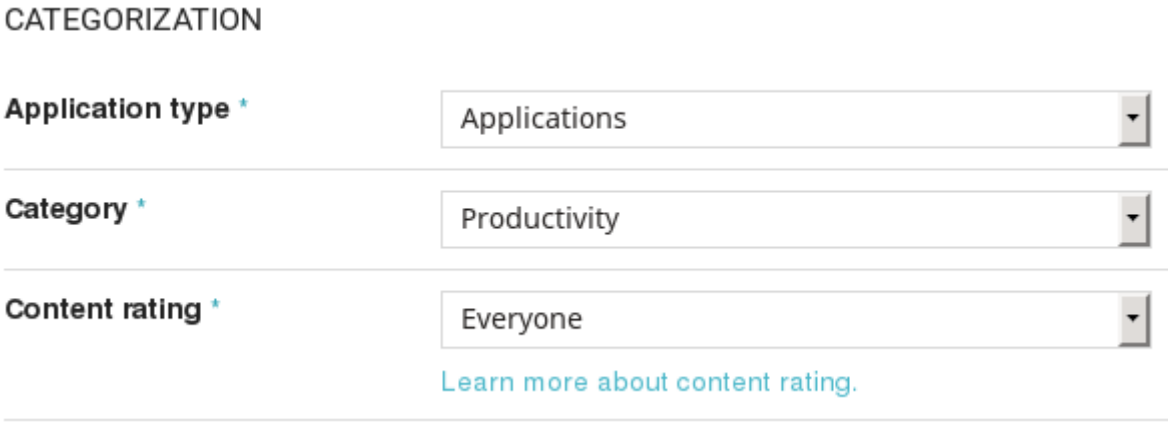
Feature Graphic *

Default – English (United States) – en-US

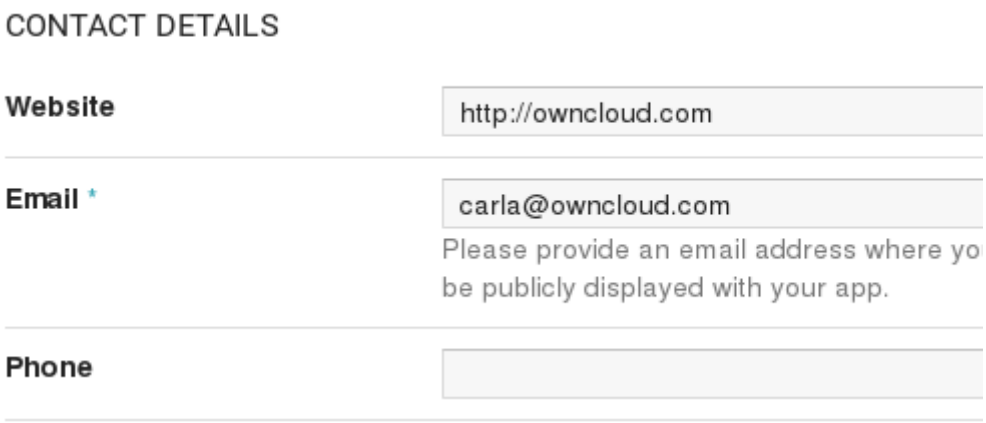
1024 w x 500 h

JPG or **24-bit PNG** (no alpha)

Now choose the store categories for your app.



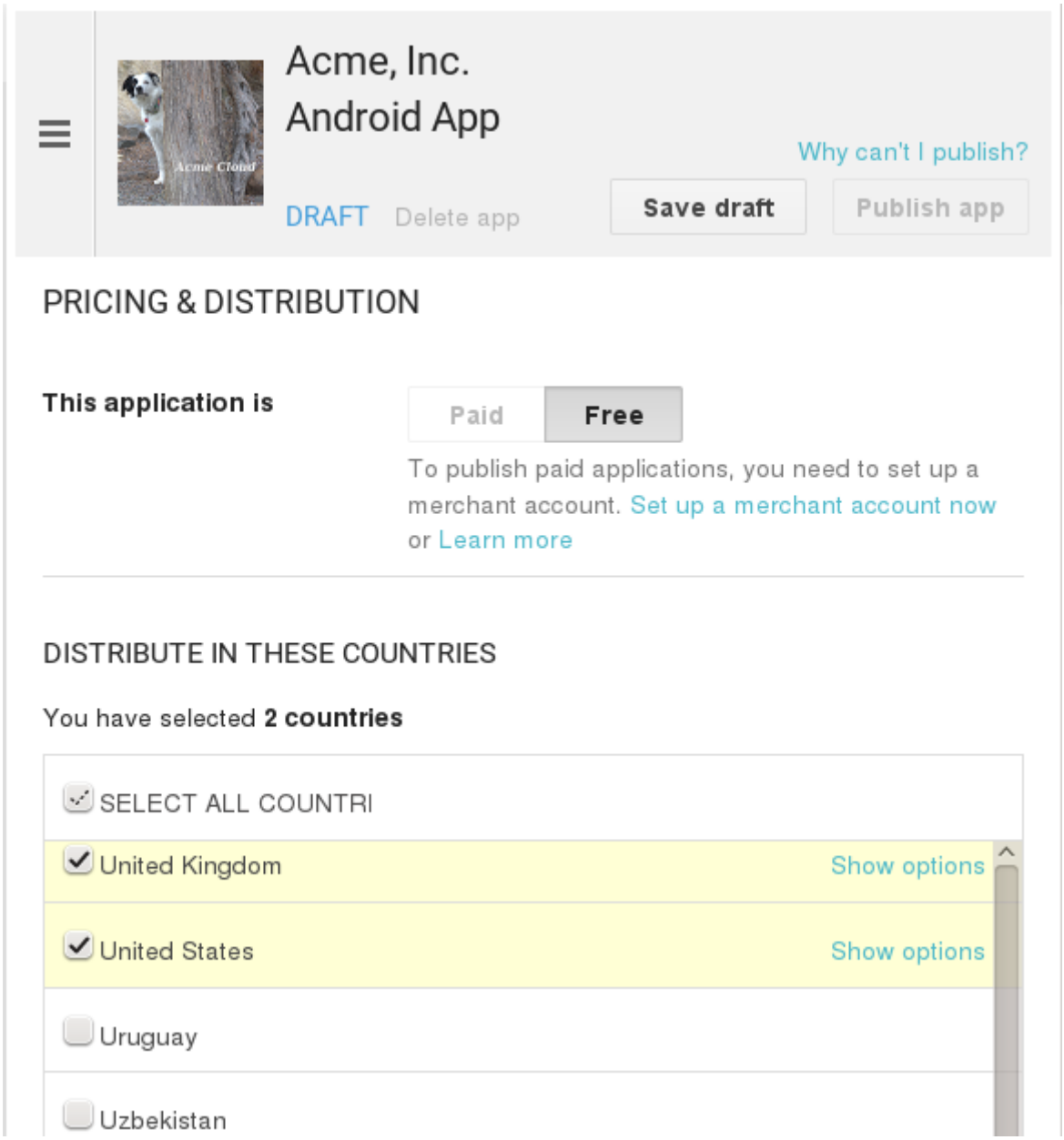
Then enter your contact information, which will be visible on your store listing.



On the next line you may optionally link to your privacy policy. It is recommended to have a privacy policy.

When you're finished with the **Store Listing** page, go to the **Pricing and Distribution** page. You may make this a paid or free app. You cannot convert a free app to paid. You may convert a paid app to free, but then you can't convert it back to paid. You'll have numerous options for paid apps, such as Android Wear, Android TV, and various Google marketing tie-ins, and many more.

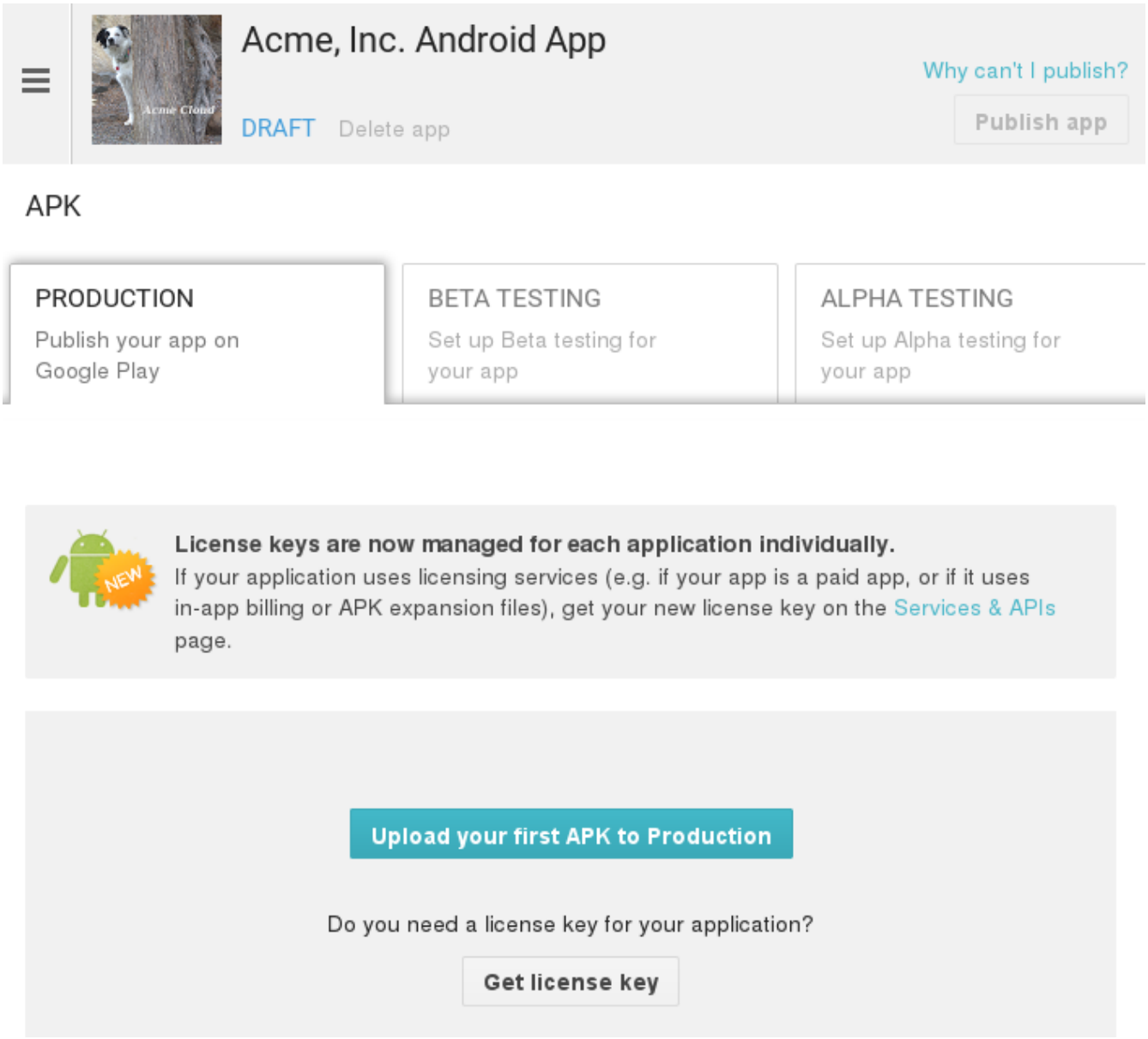
For now let's make this a free app, so click the Free button and select the countries you want to distribute it in.



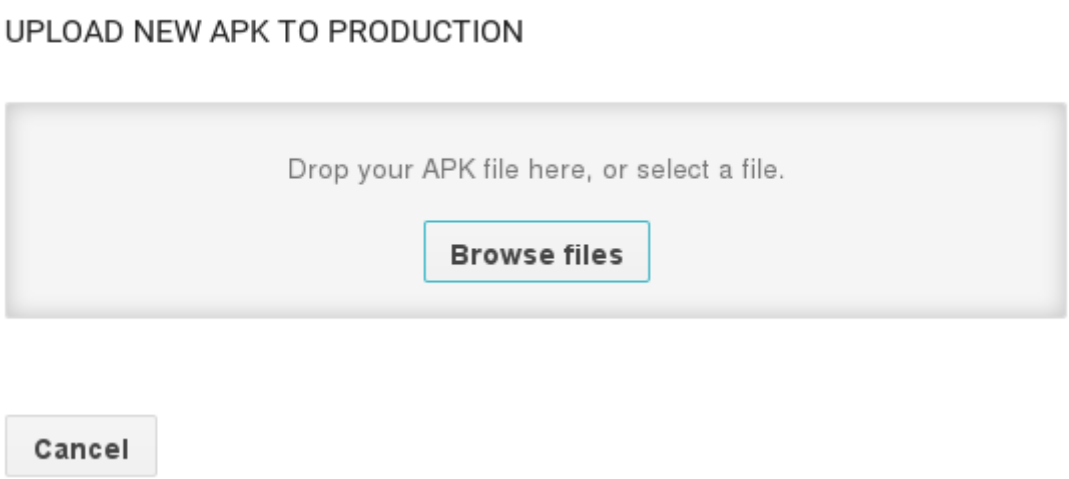
Now you may upload your app.

Uploading to Google Play Store

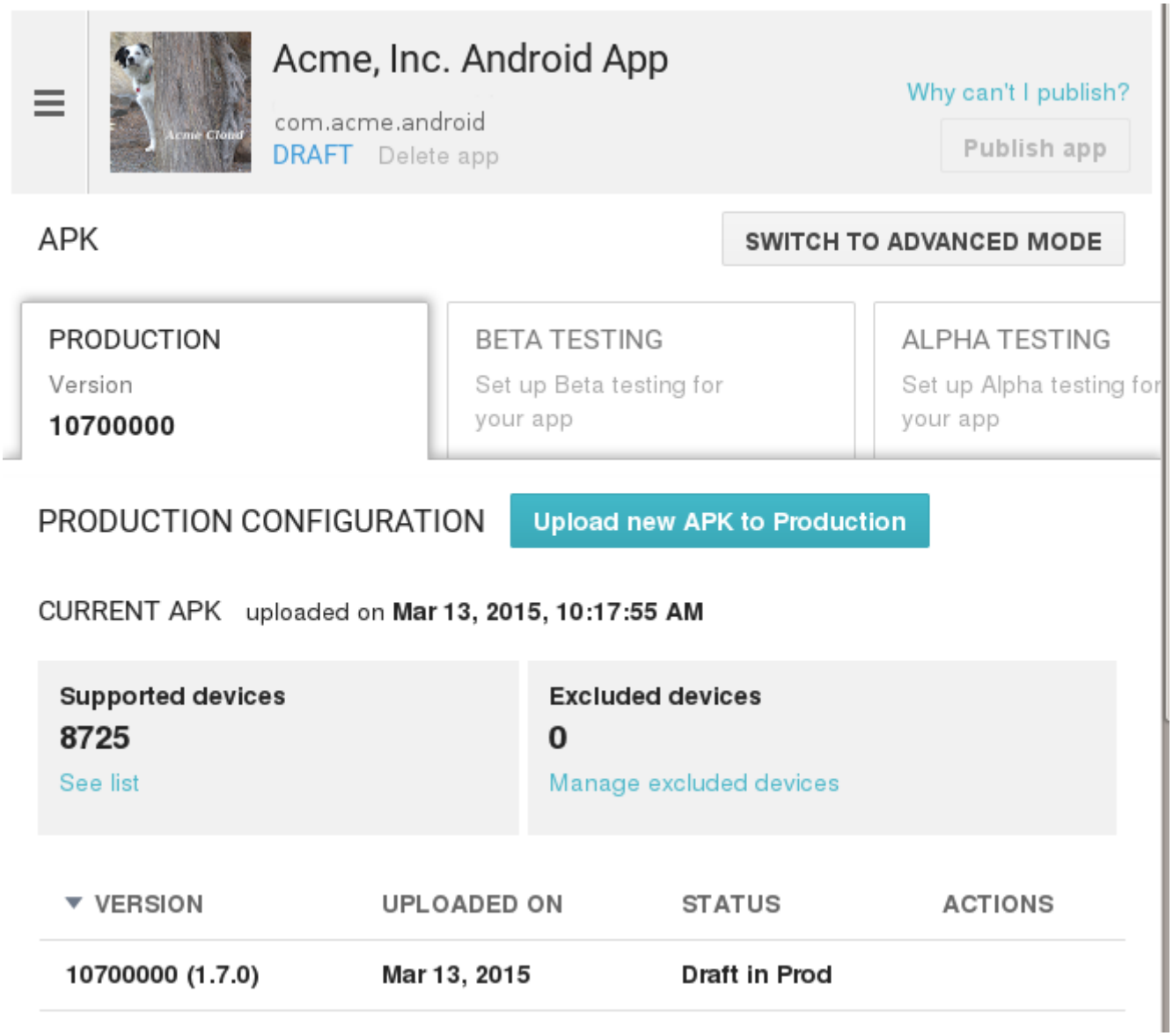
Now you can upload your app to your Google Play Store page. Go to the **APK** page and click **Upload your first APK to Production**. You don't need a license key for a free app.



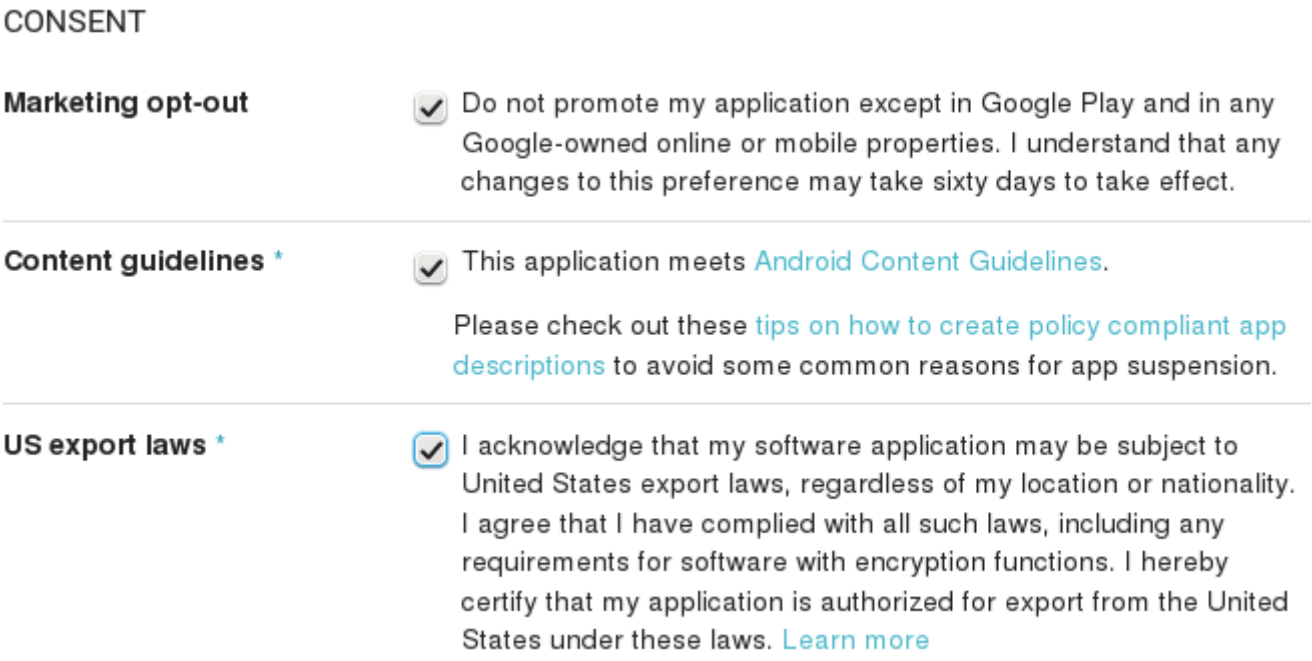
Drag-and-drop, or browse to select your app.



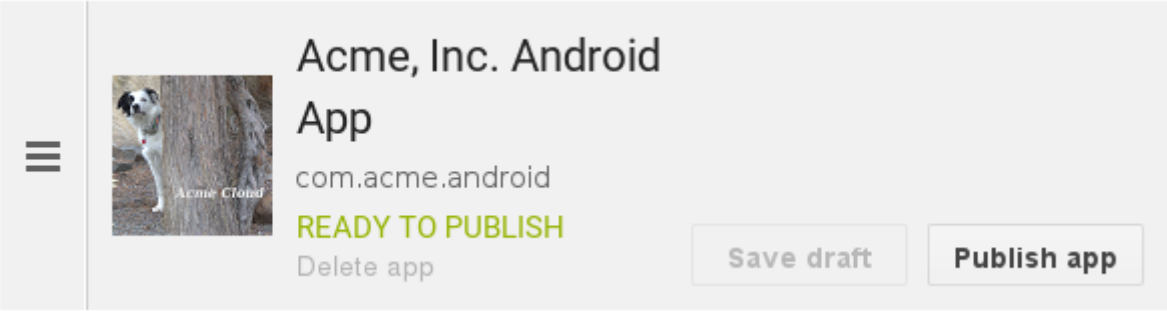
A successful upload looks like this:



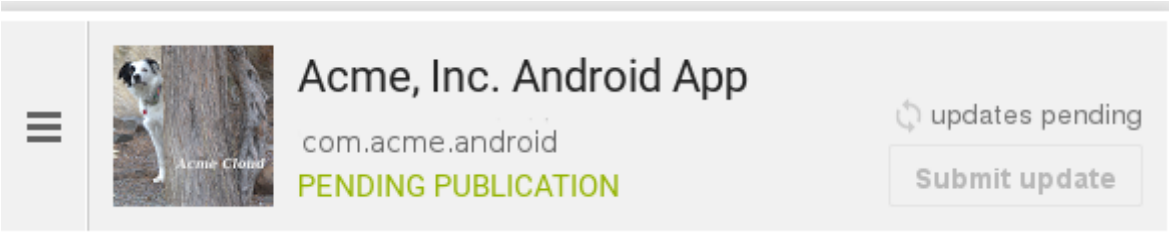
Your app is not yet published, but only uploaded to your account. There is one more step to take before you can publish, and that is to go back to the **Pricing & Distribution** page and fill out the **Consent** section.



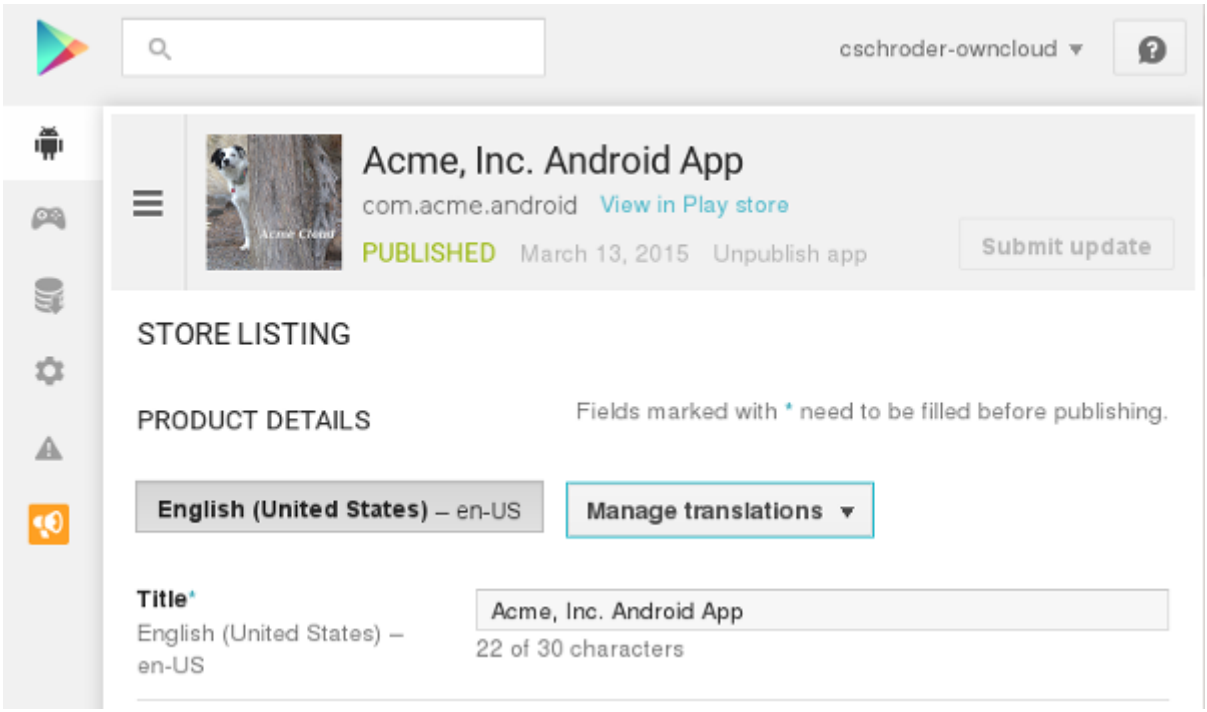
Click the Save Draft button, and if you followed all the required steps you should now see a **Publish App** button.



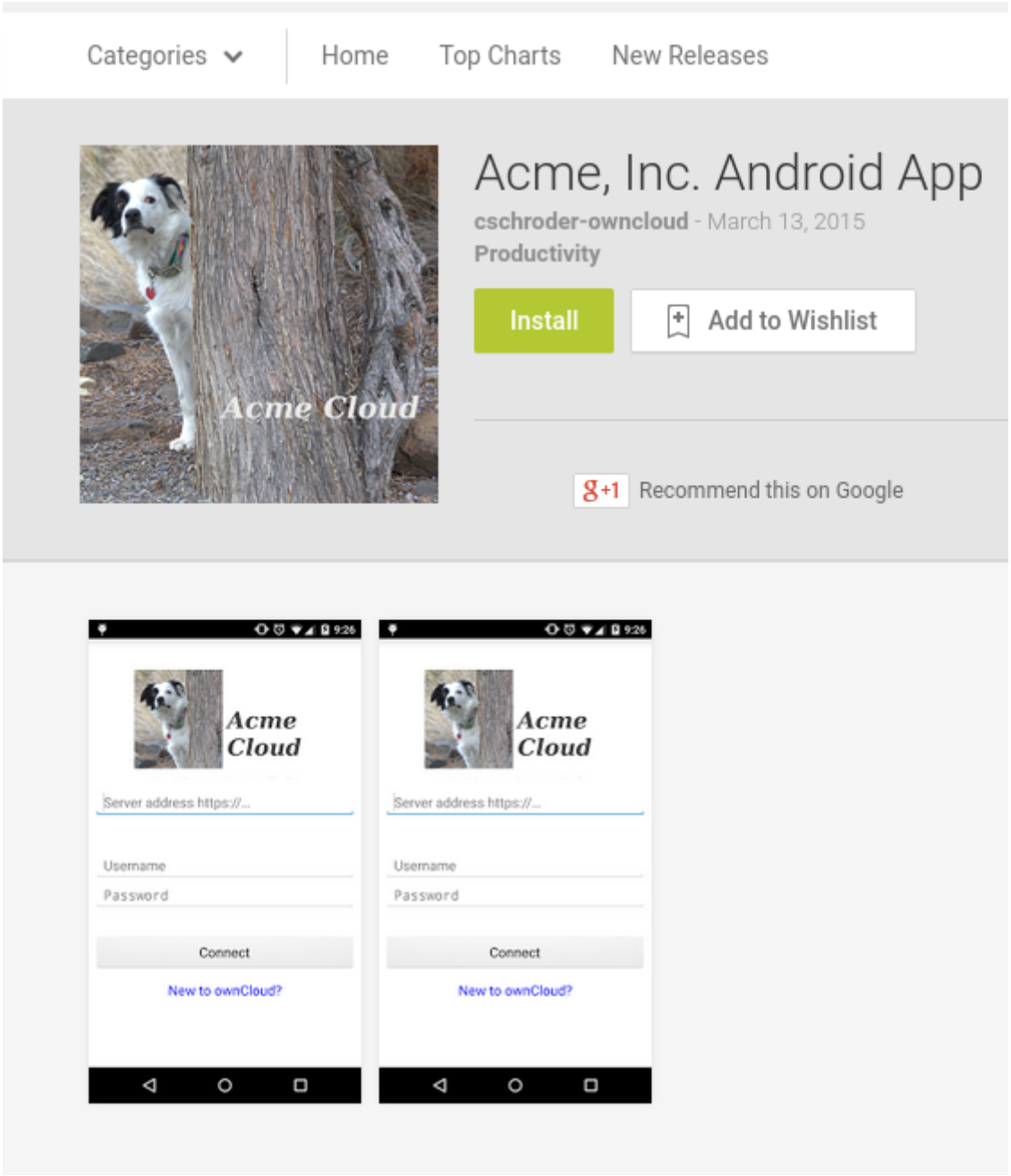
It will not be published immediately, but after review by Google, which usually takes just a few hours.



After it has been published, your store listing is updated as PUBLISHED, and it includes a link to your Play Store listing.



Now all you need to do is distribute the URL to your users, and they can install it either from their Web browsers, or from their Google Play Store apps. This is how it looks to your users.



Customize Download Link

You may configure the URLs to your own download repositories for your ownCloud desktop clients and mobile apps in config/config.php. This example shows the default download locations:

```
<?php

  "customclient_desktop" => "https://owncloud.com/desktop-app/",
  "customclient_android" =>
  "https://play.google.com/store/apps/details?id=com.owncloud.android",
  "customclient_ios"     =>
  "https://itunes.apple.com/us/app/owncloud/id543672169?mt=8",
```

Simply replace the URLs with the links to your own preferred download repos.

You may test alternate URLs without editing config/config.php by setting a test URL as an environment variable:

```
export OCC_UPDATE_URL=https://test.example.com
```

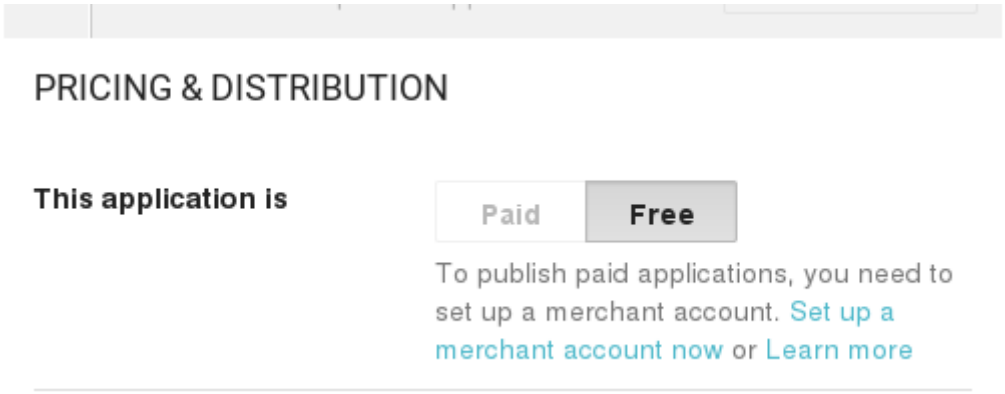
When you're finished testing you can disable the environment variable:

```
unset OCC_UPDATE_URL
```

Publishing a Paid App in Google Play

If you would rather not give your branded app away you can sell it on Google Play. You may convert a paid app to free, but you may not convert a free app to paid.

You must establish a Google Wallet Merchant Account. On your Google Dev Console click the **Learn more** link under the Free/Paid button for a nice thorough review of the process and tools. It requires verifying your business information and bank account, and you should expect it to take 3-4 days.



When you're ready to set it up, click the **Set up a merchant account now** link under the Free/Paid button.

Resources

- Get Started With Publishing
- Signing Your App Manually
- Developer Console

Branded Clients

Building a Branded Desktop Sync Client

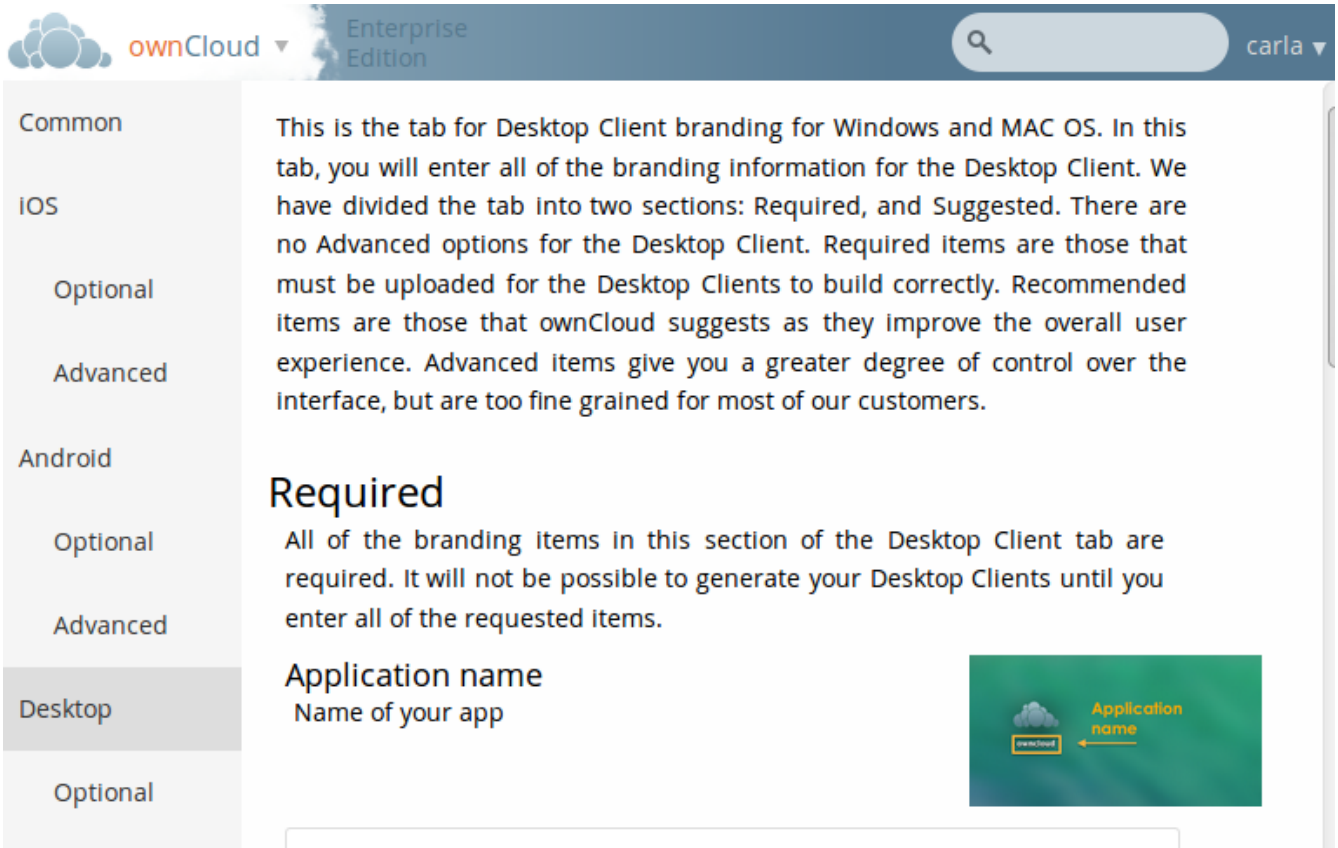
Introduction

To build a branded Desktop sync client, you need to supply your own artwork and use the ownBrander wizard in your account on customer.owncloud.com. The ownBrander wizard details the required image specifications.

Build Process

In the ownBrander wizard at your account, start with the Common section at the top, and enter information common to all clients that you can build with ownBrander. You may override any settings inside the *Common* section of the *Client* sections.

Then go to the *Desktop client section* of ownBrander, which has two sections, *Required* and *Optional*.



Work your way through the wizard, enter required elements and any optional elements you wish. When you have completed the wizard, press the **[Generate Desktop Client]** button. You will either get messages warning of any items that need to be corrected, or a success message.

It takes 24-48 hours to build your client. When finalized you will see it in your account on customer.owncloud.com.

Updating Your Branded Desktop Clients

Introduction

The Client Updater Server provides a Web service that will tell an ownCloud Desktop sync client whether or not an update is available. If an update is available, it will also provide metadata for the update, such as the Download URL, signatures or a fallback URL that the client can resort to in case the update goes wrong.

Clients for Mac OS X and Windows will update themselves automatically. Linux clients will not. You have two options for your Linux users:

- Set up your own download repository so your Linux users can update your branded clients with their package managers when they receive an update notification.
- Upload new versions of your branded client to your Web server. Your Linux users receive update notifications, then download and install the client manually.

There are times when you may want to disable update notifications. See the examples below to learn how to do this.

Prerequisites

1. Configure *Update URL* in the *Desktop* section of your ownBrander account (available for *advanced* users only).
    - Example:
      `https://mycloud.example.com/updates/`
      (note the forward slash at the end)
2. Generate branded clients.
3. Upload branded clients to your Web server.
    - Windows example:
      `https://mycloud.example.com/install/mycloud-2.1.1.240-setup.exe`
    - Mac OS X examples:
      `https://mycloud.example.com/install/mycloud-2.1.1.787.pkg`
      `https://mycloud.example.com/install/mycloud-2.1.1.787.pkg.tbz`
      `https://mycloud.example.com/install/mycloud-2.1.1.787.pkg.tbz.sig`
    - You should have a Web page with links to your branded clients, so your users can find and download them. For example, `https://mycloud.example.com/install/` with `Options +Indexes` in your ownCloud `.htaccess` file.

Install client-updater-server

1. Download `client-updater-server-0.4.tar.xz` from https://customer.owncloud.com/

2. Extract `client-updater-server-0.4.tar.xz` to your Web server. The `index.php` must be accessible at `https://mycloud.example.com/updates/index.php`.
3. Copy your ownCloud `config/ownCloud.yml` file, and name it according your **Application short name** as configured in ownBrander.

   Example: `config/mycloud.yml`

Configure client-updater-server

All configuration is done in your `config/mycloud.yml`:

```
throttle: 1 # 100% of the requests get served with the new version

platforms:
  win32msi:
    currentVersion: 2.5.0.10598
    currentVersionString: ownCloud Client 2.5.0 (build 10598)
    updateUrl: https://owncloud.com/desktop-app
    downloadUrl: http://download.owncloud.com/desktop/stable/ownCloud-2.5.0.10598.msi

  win32:
    currentVersion: 2.4.3.10188
    currentVersionString: ownCloud Client 2.4.3 (build 10188)
    updateUrl: https://owncloud.com/desktop-app
    downloadUrl: http://download.owncloud.com/desktop/stable/ownCloud-2.4.3.10188-
setup.exe

  linux:
    currentVersion: 1.8.0
    currentVersionString: ownCloud Client 1.7.1
    updateUrl: https://owncloud.com/desktop-app

  macos:
    currentVersion: 1.8.0.2139
    currentVersionString: ownCloud Client 1.8.0 (build 2139)
    downloadUrl: https://download.owncloud.com/desktop/stable/ownCloud-
1.8.0.2139.pkg.tbz
    pubDate: 2015-03-26
    signature: MCwCFFedScUKeRXYMS6vKVLw821B+/+lAhRbiCxHNzVVZFNXHSvB9GNHOuI5cw==
    minimumSystemVersion: 10.7.0
```

In earlier versions this configuration was written in PHP, which is still supported but no longer the default. The structure slightly changed and would look like this analoguely to the yml config `config/mycloud.php`:

```
<?php

$updateInfo = [
```

```
    'throttle' => 0.7, // 70% of the requests get served with the new version
    'platforms' => [
        'win32msi' => [
            'currentVersion' => '2.5.0.10598',
            'currentVersionString' => 'ownCloud Client 2.5.0 (build 10598)',
            'updateUrl' => 'https://owncloud.com/desktop-app',
            'downloadUrl' => 'http://download.owncloud.com/desktop/stable/ownCloud-
2.5.0.10598.msi',
        ],
        'win32' => [
            'currentVersion' => '2.4.3.10188',
            'currentVersionString' => 'ownCloud Client 2.4.3 (build 10188)',
            'updateUrl' => 'https://owncloud.com/desktop-app',
            'downloadUrl' => 'http://download.owncloud.com/desktop/stable/ownCloud-
2.4.3.10188-setup.exe',
        ],
        'linux' => array(
            'currentVersion' => '1.8.0',
            'currentVersionString' => 'ownCloud Client 1.7.1',
            'updateUrl' => 'https://owncloud.com/desktop-app',
        ),
        'macos' => array(
            'currentVersion' => '1.8.0.2139',
            'currentVersionString' => 'ownCloud Client 1.8.0 (build 2139)',
            'downloadUrl' => 'https://download.owncloud.com/desktop/stable/ownCloud-
1.8.0.2139.pkg.tbz',
            'pubDate' => '2015-03-26',
            'signature' =>
'MCwCFFedScUKeRXYMS6vKVLw821B+/+lAhRbiCxHNzVVZFNXHSvB9GNHOuI5cw==',
            'minimumSystemVersion' => '10.7.0',
        ),
    ]
];
```

(The former top-level config options were moved under a `platforms` key.)

Disabling Notifications

There may be times when you wish to disable update notifications. To do this, make the `'currentVersion'` and `'currentVersionString'` older than the currently installed version. To re-enable notifications, change these to release versions that are newer than the currently installed clients.

Windows

- `'currentVersion'`
  Exact version of the new client, including the build number
- `'currentVersionString'`
  Name of the new client, same as "Application name" configured in ownBrander.

- `'updateUrl'`
  Human-readable Web site with links to your new client files.
- `'downloadUrl'`
  Full URL to download the *.exe file. http**s** needed.

Mac OS X

- `currentVersion'`
  Exact version of the new client, including the build number.
- `'currentVersionString'`
  Name of the new client, same as `Application name` configured in ownBrander.
- `'downloadUrl'`
  Full URL to download the *.pkg**.tbz** file. http**s** needed.
- `'pubDate'`
  Currently not used.
- `'signature'`
  Content of `mycloud-2.1.1.787.pkg.tbz.sig`, adds some extra security to the Mac OS X updater.
- `'minimumSystemVersion'`
  Minimum required Mac OS X version according to https://owncloud.com/desktop-app/

Linux

- `'currentVersion'`
  Exact version of the new client, including the build number
- `'currentVersionString'`
  Name of the new client, same as `Application name` configured in ownBrander.
- `'updateUrl'`
  Human-readable Web site with links to your new client files to manually install new client versions.

Debugging client-updater-server

Windows

This a example URL of a 2.5.0 client for Microsoft Windows:
`https://mycloud.example.com/updates/?version=2.5.0.10598&platform=win32&msi=true&oem=mycloud`

You should see something like the following in your Web server logs:

```
[19/Feb/2016:14:33:35 +0100] "GET
/updates/?version=2.5.0.10598&platform=win32&msi=true&oem=mycloud HTTP/1.1" 200 185 "-
"
"Mozilla/5.0 (Windows) mirall/2.5.0 (mycloud)" microsecs:530450
```

The output should look like this if you call the URL manually:

```
<?xml version="1.0"?>
   <owncloudclient>
      <version>2.5.0.10598</version>
       <versionstring>MyCloud Client 2.5.0 (build 10598)</versionstring>
       <web>https://mycloud.example.com/install/</web>
       <downloadurl>https://mycloud.example.com/install/
        mycloud-2.5.0.10598.msi</downloadurl>
   </owncloudclient>
```

Mac OS X

This a example URL of a 2.1.1 client for Mac OS X:

`https://mycloud.example.com/updates/?version=2.1.1.687&platform=macos&oem=mycloud&sparkle=true`

You should see something like the following in your Web server logs:

```
[19/Feb/2016:14:00:17 +0100] "GET
/updates/?version=2.1.1.687&platform=macos&oem=mycloud&sparkle=
true HTTP/1.1" 200 185 "-" "Mozilla/5.0 (Macintosh) mirall/2.1.1 (mycloud)"
microsecs:1071 response_size:2070 bytes_received:306 bytes_sent:2402
```

The output should look like this if you call the URL manually:

```
<?xml version="1.0" encoding="utf-8"?>
  <rss version="2.0"
  xmlns:sparkle="http://www.andymatuschak.org/xml-namespaces/sparkle"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel>
     <title>Download Channel</title>
     <description>Most recent changes with links to updates.</description>
     <language>en</language><item>
     <title>MyCloud Client 2.1.1 (build 787)</title>
     <pubDate>Mon, 23 Feb 16 00:00:00 -0500</pubDate>
     <enclosure url="https://mycloud.example.com/install/
       mycloud-2.1.1.787.pkg.tbz" sparkle:version="2.1.1.787"
       type="application/octet-stream"
       sparkle:dsaSignature="MCwCFFedScUKeRXYMS6vKVLw821B+/+
         lAhRbiCxHNzVVZFNXHSvB9GNHOuI5cw=="/>
      <sparkle:minimumSystemVersion>10.7.0</sparkle:minimumSystemVersion>
   </item>
   </channel>
  </rss>
```

Deploy And Update Branded Linux Desktop Clients

Introduction

As an ownBrander user, you can enable the build of branded Linux in the “Desktop” section of your account. You can download a `*.tar` file which contains everything to set up complete self-hosted Linux repositories for the selected Linux distributions.

Setup

This is the content of `mycloud-2.10.0.6752-linux.tar`:

```
mycloud-2.10.0.6752-linux
 ├──── CentOS_7
 │     └────── ...
 ├──── Debian_10
 │     └────── ...
 ├──── Debian_11
 │     └────── ...
 ├──── Fedora_33
 │     └────── ...
 ├──── Fedora_34
 │     └────── ...
 ├──── Fedora_35
 │     └────── ...
 ├──── Ubuntu_18.04
 │     └────── ...
 ├──── Ubuntu_20.04
 │     └────── ...
 ├──── Ubuntu_21.04
 │     └────── ...
 ├──── Ubuntu_21.10
 │     └────── ...
 ├──── download
 │     ├──── CentOS.html
 │     ├──── Debian.html
 │     ├──── Fedora.html
 │     ├──── Ubuntu.html
 │     ├──── allplatforms.html
 │     ├──── assets
 │     │     ├──── application.css
 │     │     ├──── application.js
 │     │     ├──── arch.png
 │     │     ├──── centos.png
 │     │     ├──── debian.png
 │     │     ├──── download.html
 │     │     ├──── favicon.png
 │     │     ├──── fedora.png
 │     │     ├──── global-navigation-data-en.js
 │     │     ├──── globalnav-im.png
 │     │     ├──── header-logo.png
 │     │     ├──── opensuse.png
```

```
│   │   ├── repo.cfg
│   │   ├── rhel.png
│   │   ├── sl.png
│   │   ├── sle.png
│   │   ├── ubuntu.png
│   │   ├── univention.png
│   │   ├── unknown.png
│   │   └── ymp-added-repos.txt
│   ├── bin
│   │   └── repo-admin.py
│   ├── download.json
│   ├── example.sh
│   ├── index.html
│   ├── openSUSE.html
│   └── ymp
│       ├── openSUSE_Leap_15.2
│       │   └── mycloud-client.ymp
│       ├── openSUSE_Leap_15.3
│       │   └── mycloud-client.ymp
│       └── openSUSE_ymp.html
├── openSUSE_Leap_15.2
│   └── ...
└── openSUSE_Leap_15.3
    ├── INSTALL.sh
    ├── PREINSTALL.sh
    ├── libmycloudsync-devel-2.10.0-6752.x86_64.rpm
    ├── ocqt51210-libQt5Concurrent5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5Core5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5DBus5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5Gui5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5Network5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5PrintSupport5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5Sql5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5Widgets5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libQt5Xml5-5.12.10-lp153.15.1.x86_64.rpm
    ├── ocqt51210-libqt5-qtsvg-5.12.10-lp153.2.1.x86_64.rpm
    ├── ocqt51210-libqt5-qttranslations-5.12.10-lp153.2.1.x86_64.rpm
    ├── ocqt51210-libqt5-qtwayland-5.12.10-lp153.18.1.x86_64.rpm
    ├── ocqt51210-qt5keychain1-0.12.0-lp153.3.1.x86_64.rpm
    ├── repodata
    │   ├── 3087af5cd89a88977dc9d04e5a3724c573b5b1da3da26bf39931c35f30f00b04-
primary.xml.gz
    │   ├── c0ee1520c439b5272490d193ceb9a31230b9cb7de16dd88565e1c8f6218f006b-
other.xml.gz
    │   ├── da70c01f9d56865f5792e19529fe3b5cdbd5dd94162fbd3e117577ead7a422cf-
filelists.xml.gz
    │   ├── repomd.xml
    │   ├── repomd.xml.asc
    │   └── repomd.xml.key
    ├── mycloud-client-2.10.0-6752.x86_64.rpm
    ├── mycloud-client-doc-2.10.0-6752.x86_64.rpm
```

```
    ├──── mycloud-client-dolphin-2.10.0-6752.x86_64.rpm
    ├──── mycloud-client-nautilus-2.10.0-6752.x86_64.rpm
    ├──── mycloud-client-nemo-2.10.0-6752.x86_64.rpm
    ├──── mycloud-client-overlays-icons-2.10.0-6752.x86_64.rpm
    ├──── mycloud-client.ymp
    ├──── mycloud.repo
    └──── src
        ├──── mycloud-client-2.10.0-6752.src.rpm
        └──── mycloud-client-overlays-2.10.0-6752.src.rpm
```

The download folder provides detailed instructions for your users at `download/index.html` about how to install the branded Linux clients on the selected Linux distributions. All location information in the HTML files is set to `download.example.com`. All metadata in the repo is set to `download.example.com` too.

The download folder contains a shell script (`/download/example.sh`). This allows you to modify the HTML and the repo itself according to your repo location on your webserver.

download/example.sh

```
#! /bin/bash
#
# This example demonstrates how to call repo-admin.py
# You will need to call repo-admin.py with your download url.
# Basic auth username and password is supported as shown below.
#
# You can customitze the main html file and re-run repo-admin.py later.
#
cd $(dirname $0)
set -x
python bin/repo-admin.py \
  --url http://download.example.com/repo \
  -d 'download' \
  -p '.*-client' \
  -i 'index.html' \
  -f ..
```

Replace `http://download.example.com/repo` in the above example with the base URL of your repository and save the file with a new name. (`download/mycloud.sh`)

Then execute the script and check `download/index.html` on your webserver.

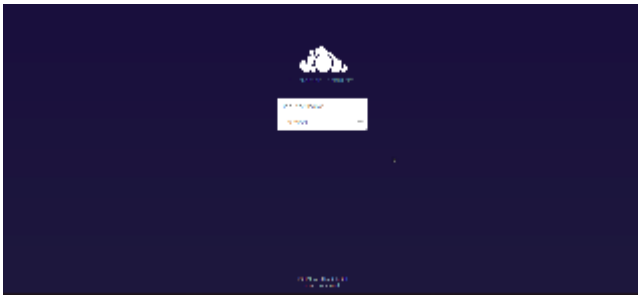
Building Your App With ownBrander

Introduction

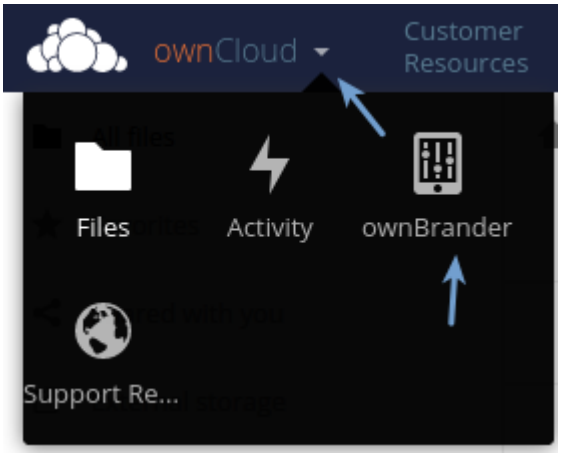
Follow along and begin creating your ownBrander Android app. Within the ownBrander you will find that most of the fields that require your input are self-explanatory. If you still have questions, or perhaps suggestions, feel free contact us.

Use your credentials to log in to customer.owncloud.com.

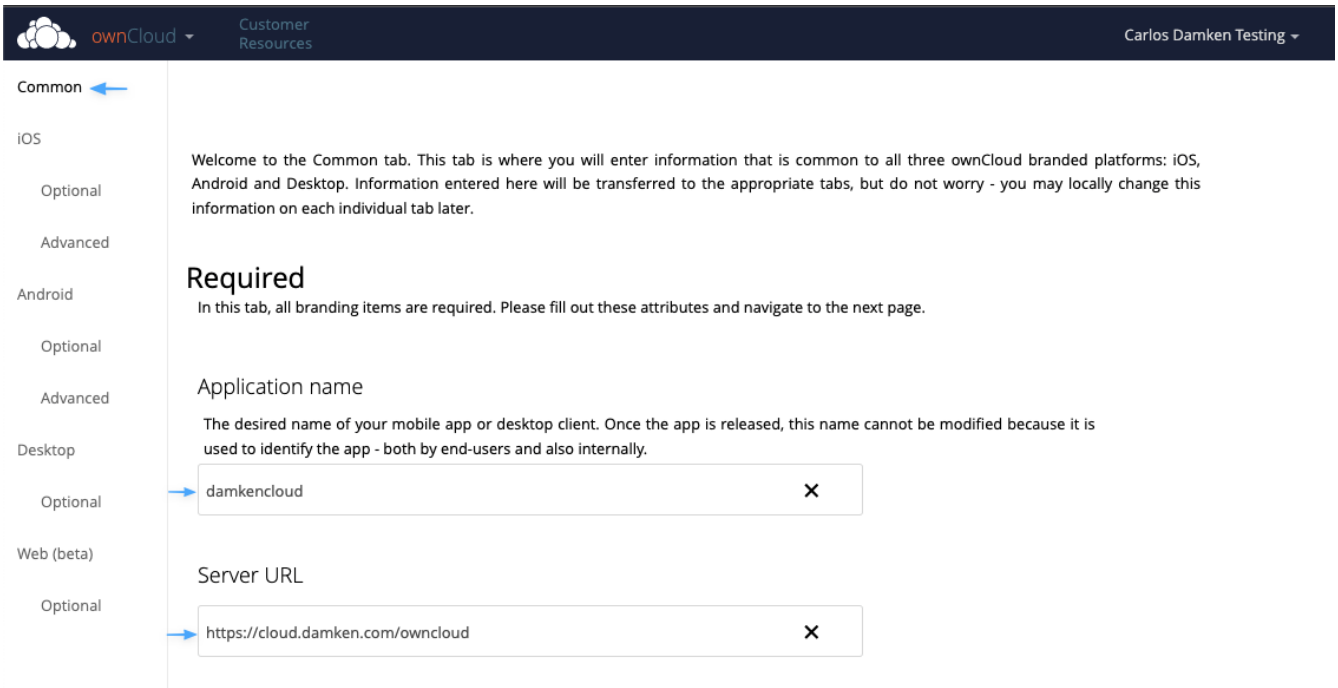
If you need credentials, do not hesitate to contact your account sales representative.



After successfully logging in to your account, navigate to the left side of your ownCloud instance and click on the inverted arrow to open the menu. Then click the ownBrander icon to open it.



Begin on the **Common Tab** and navigate to the **Required** area. Next, enter your application name and the URL of your ownCloud server in the corresponding fields.

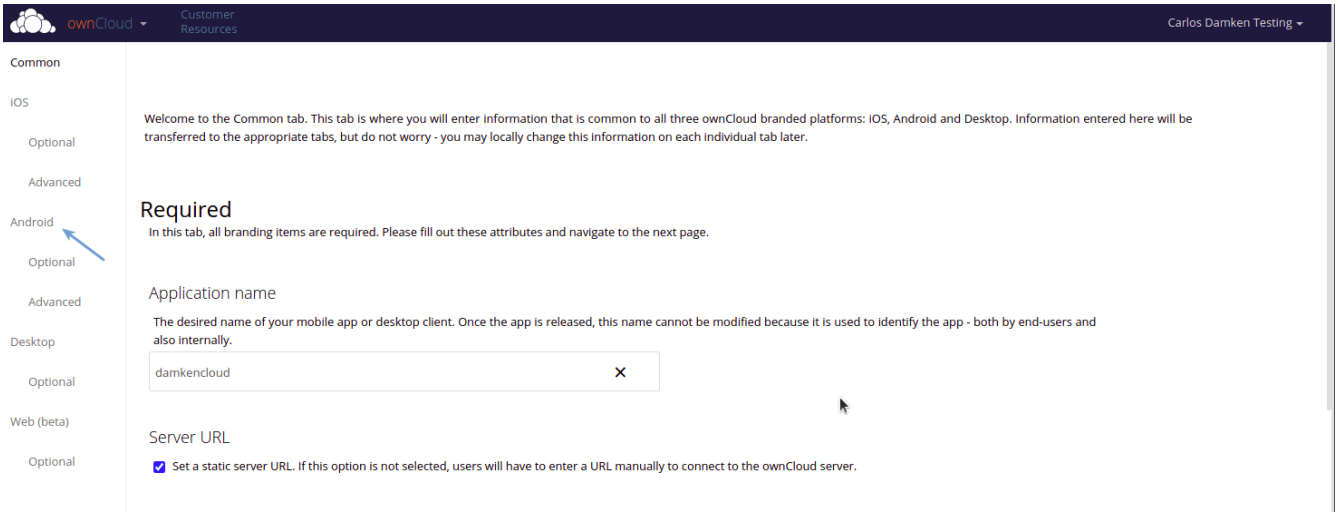


These entries define your global defaults for all of the platforms in ownBrander. You can change them when you create your apps.

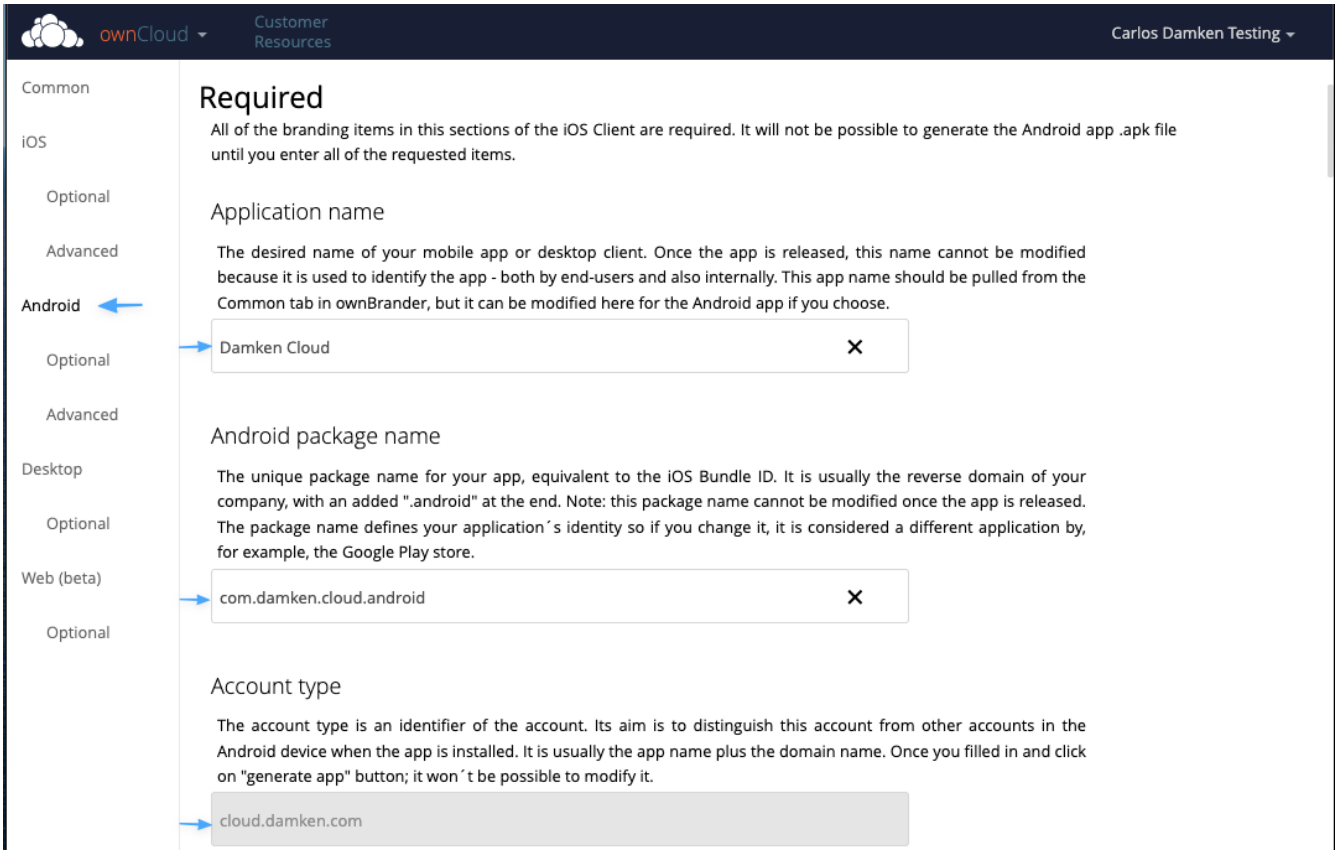
When you create production apps, you must then use your real app name, and the URL must point to your real ownCloud server. (However, for testing purposes, these values can be anything.)

Configuring ownBrander Parameters

Next, click on the **Android Tab**. Here you will find another **Required** area to fill out. You will also see two additional tabs: **Optional**, and **Advanced**. These are not mandatory for the processing of your branding experience.



Start in the **Required** section and type in your application name, your Android package name and your account typ.

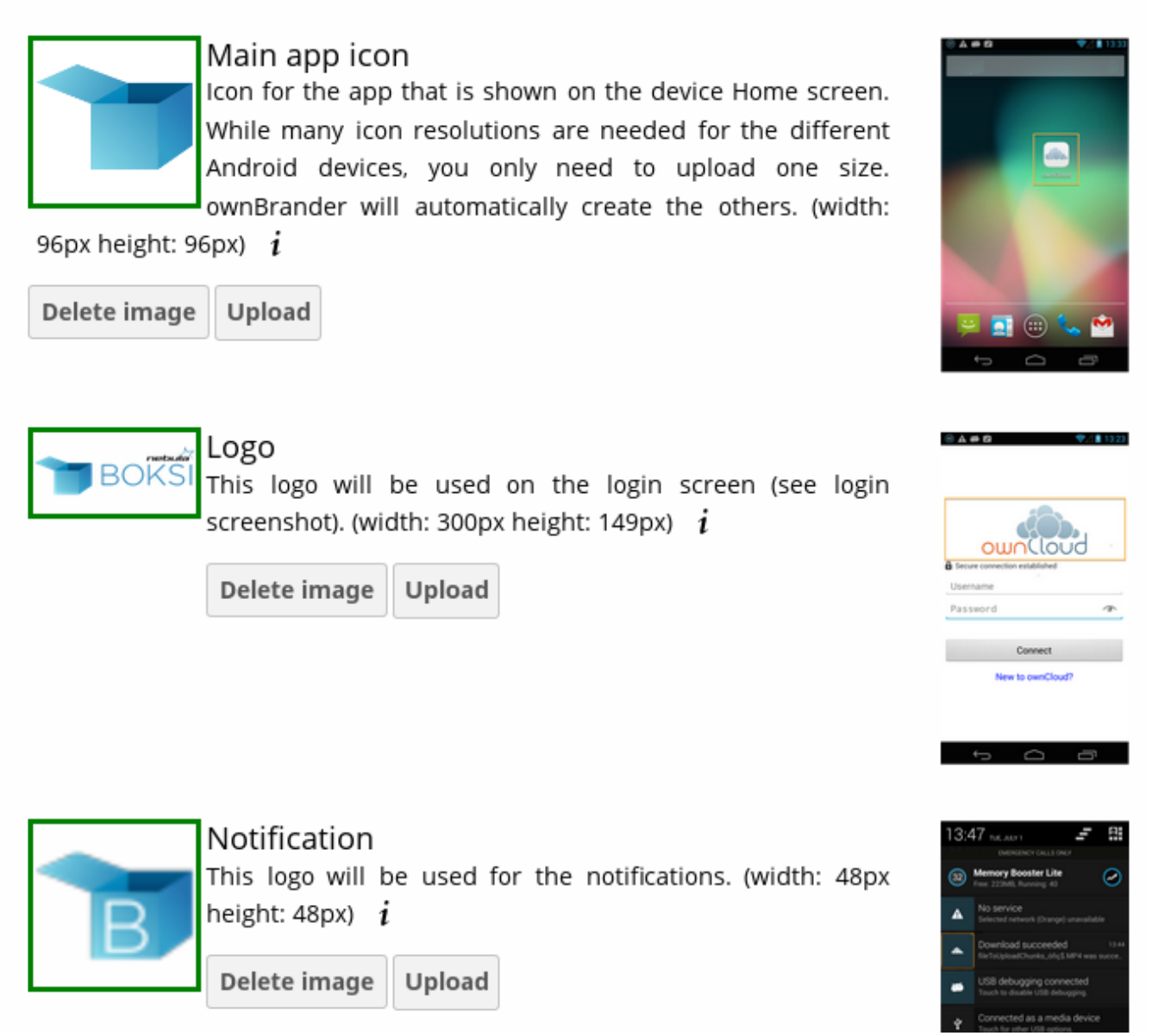


Move on to styling your app.

Wizard for your Branding Images

Building an Android app requires just a few images (preferably PNG format). The wizard also provides you with the exact dimension requirements. Please make certain that your images fulfill the specifications.

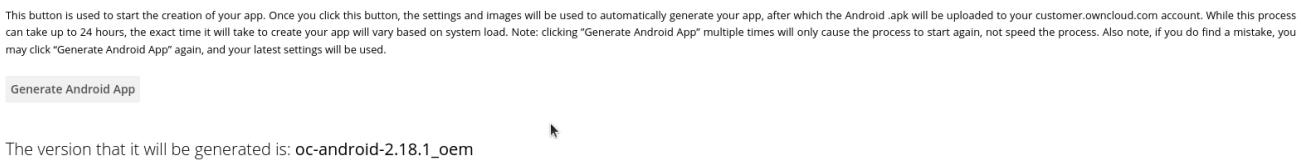
Now you can upload your images into the pre-defined boxes.



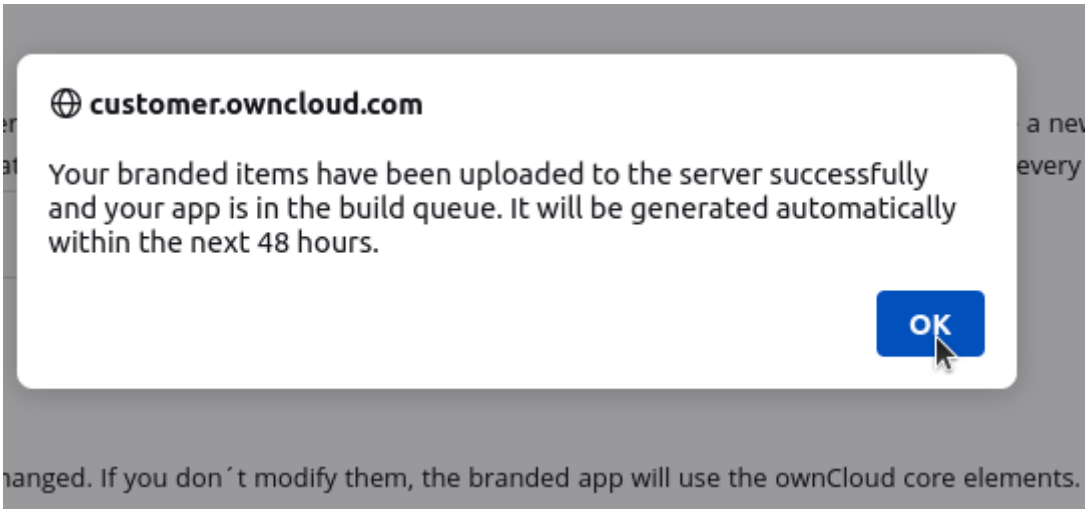
Proceed to the section regarding the signing of your Android client and click on the checkbox if you want this option. Provide the required information for the three fields (Key Store Password, Key Alias and Key Alias Password.) You will need these credentials in the Google Play Store Console.

Generating Your App

After a first-time ownCloud branding or after performing any modifications (updating your app), click the **Generate Android App** button.



The following information will pop up - click ok.

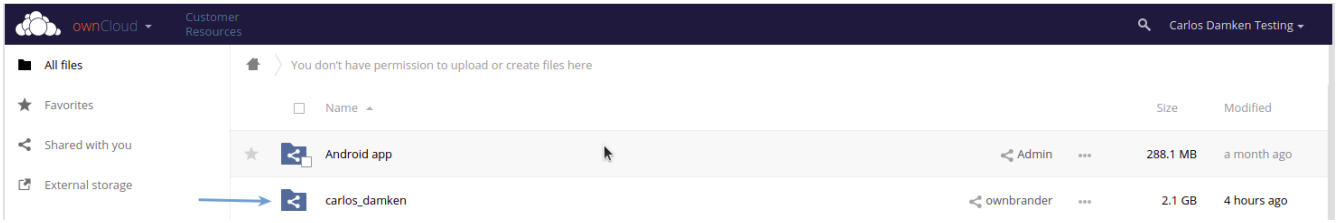


Download Your Branded App

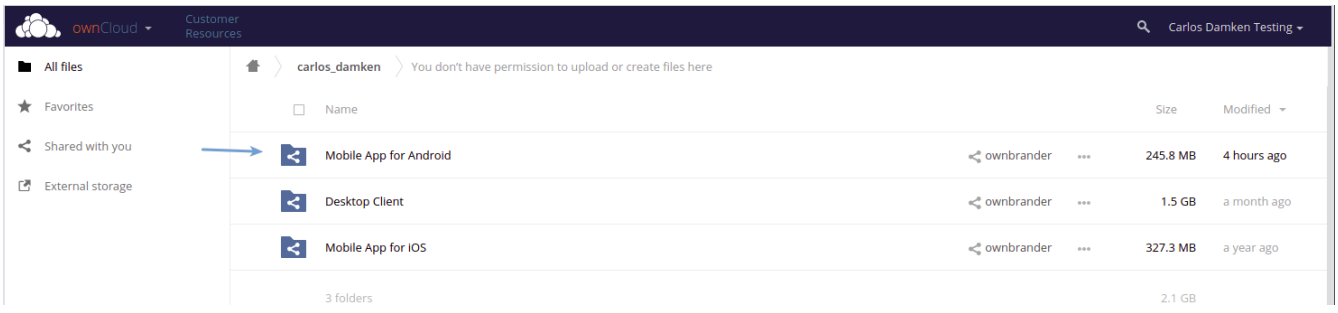
After the above mentioned time frame, open your account at: customer.owncloud.com

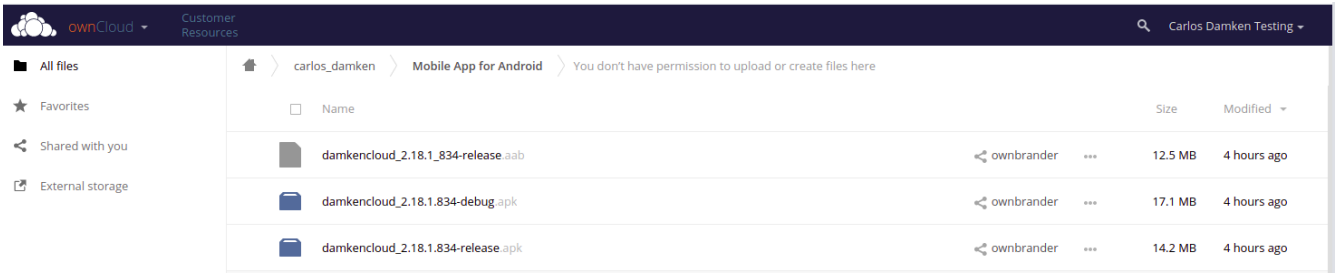
The ownBrander produces three files required to build your app. These are automatically uploaded into your personal folder.

Choose the folder with your name to find the folders for your various branded clients.



Choose the **Mobile App for Android** folder. This folder should contain the three files seen in the image below. The .aab file is of relevance for your app in the Google Play Console.





Now you must sign your app in the Google Play Store in order to distribute it to your users.

Distributing Your Branded Android App

Introduction

Now that you have created your branded Android app with ownCloud's ownBuilder service (building_branded_android_client) how do you distribute it to your users? There are multiple ways: email, publish_server, or publish_google_play. However you distribute it, the first step is to digitally sign your new app. Signing your app verifies authorship and authenticity.

When you create your branded Android app we supply you with two `.apk` files: one for debugging and testing, and one for deployment, like these examples:

```
acmecloud_2.0.0-debug.apk
acmecloud_2.0.0-release-unsigned.apk
```

The second `.apk` file, `acmecloud_2.0.0-release-unsigned.apk`, is the one you will sign and distribute.

Digitally Signing Android Apps

Signing your app is required. You can do this in the ownBrander wizard <sign_android_app>, or after it is built and delivered to you. The most time-consuming part of signing the built app is installing the commands you need to sign it. You need three commands to sign your app: `keytool`, `jarsigner`, and `zipalign`. Follow these steps:

1. Install the signing commands
2. Create a self-signed certificate with `keytool`
3. Use `jarsigner` to sign the app, and to verify signing
4. Use `zipalign` to optimize your app

You only need to create a certificate once, and then use it to sign all of your branded ownCloud apps. If you publish your apps on Google Play they must all be signed with the same certificate.

Installing the App Signing Tools

`keytool` and `jarsigner` are in Java runtimes. Linux users can get these in OpenJDK. For example, on current versions of Debian, Mint, and Ubuntu Linux you need to install two packages. The first one supplies `keytool` and the second one supplies `jarsigner`:

```
sudo apt-get install openjdk-8-jre-headless
sudo apt-get install openjdk-8-jdk
```

Plus some additional 32-bit packages:

```
sudo apt-get install libc6-i386 lib32stdc++6 \
    lib32gcc1 lib32ncurses5-dev zlib1g:i386
```

On SUSE systems, install this package:

```
sudo zypper install java-1_7_0-openjdk-devel
```

It is simpler to get these on CentOS and Red Hat Enterprise Linux, as they have created some nice wrapper scripts around `keytool` and `jarsigner` that you can install standalone:

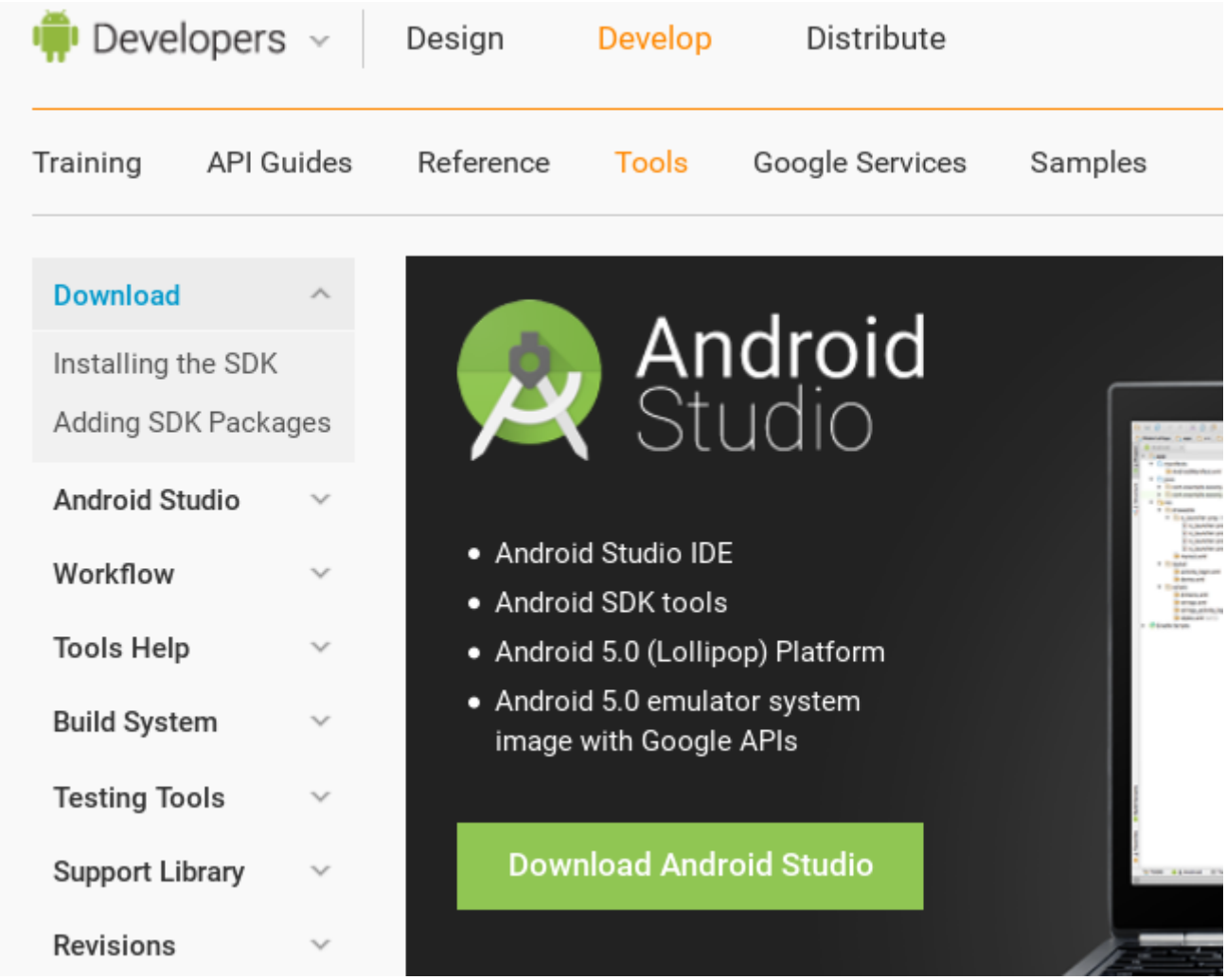
```
sudo yum install keytool-maven-plugin.noarch
sudo yum install maven-jarsigner-plugin.noarch
```

Mac OS X and Windows users can download the Oracle JDK from Oracle's Java Download page.

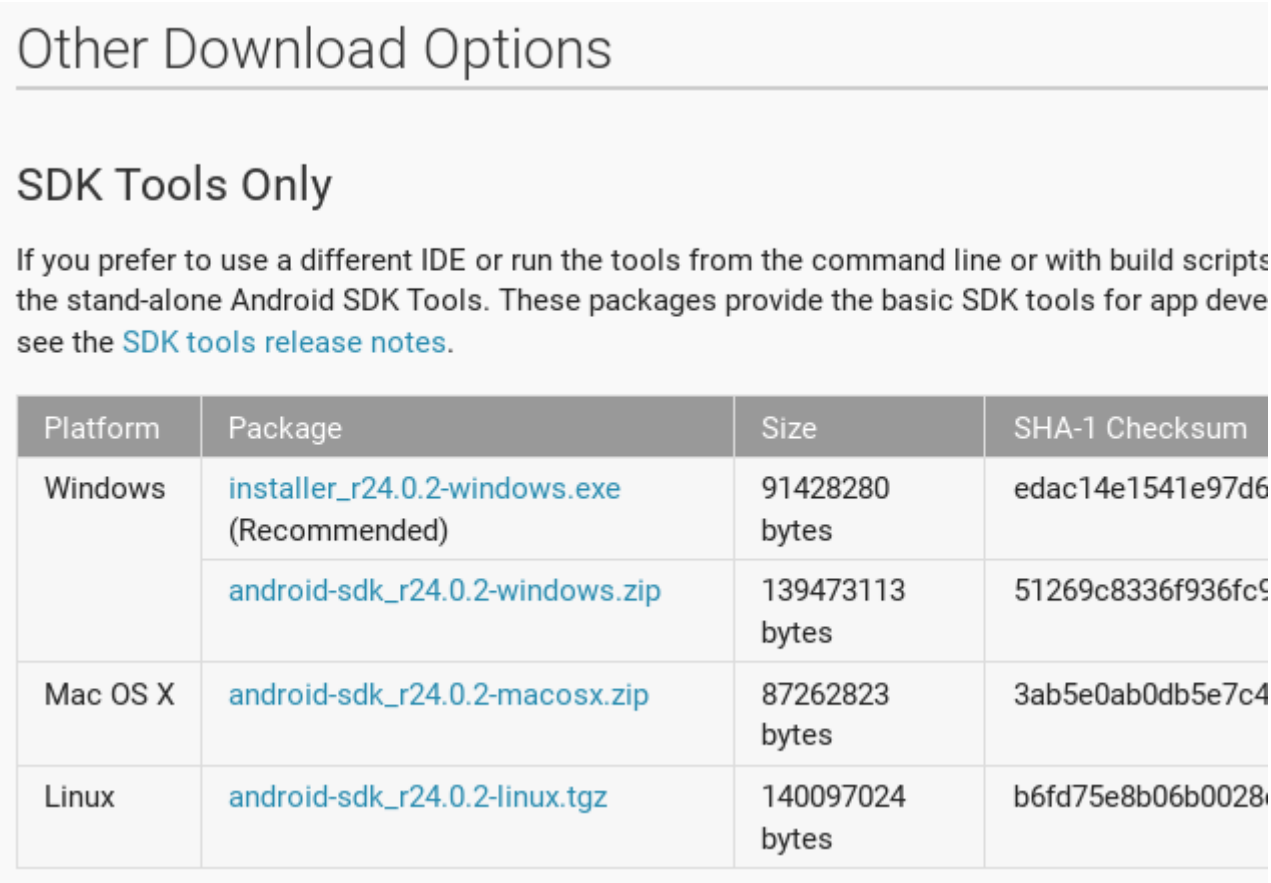
If your operating system provides the `zipalign` package, you can install it with:

```
sudo apt install zipalign
```

In case `zipalign` is not provided as installable package for your OS, you can download it from source via the Android Software Development Kit. It is a large download, but once you have downloaded it you can copy the `zipalign` binary to any computer and use it. Go to Android Software Development Kit and click the "Download Android Studio" button.



Download the appropriate **SDK Tools Only** package for your operating system.



Other Download Options

SDK Tools Only

If you prefer to use a different IDE or run the tools from the command line or with build scripts the stand-alone Android SDK Tools. These packages provide the basic SDK tools for app deve see the SDK tools release notes.

Platform	Package	Size	SHA-1 Checksum
Windows	installer_r24.0.2-windows.exe (Recommended)	91428280 bytes	edac14e1541e97d6
	android-sdk_r24.0.2-windows.zip	139473113 bytes	51269c8336f936fc9
Mac OS X	android-sdk_r24.0.2-macosx.zip	87262823 bytes	3ab5e0ab0db5e7c4
Linux	android-sdk_r24.0.2-linux.tgz	140097024 bytes	b6fd75e8b06b0028

Unpack it and change to the unpacked directory, which is `android-sdk-linux` on Linux systems, `android-sdk-macosx` on Mac systems, and `android-sdk-windows` on Windows systems. There is one more step, and that is to install additional tools. Run this command from the unpacked directory:

```
tools/android update sdk --no-ui
```

This will take some time, as it is a large download. When it's finished you'll find `zipalign` in the `build-tools` directory. For convenience, you could copy `zipalign` to your home folder or other location of your choice, and to any other computer without installing the whole Android SDK.

Digitally Signing Your App

After installing your signing tools, signing your app takes just a few steps. In these examples the name of the app, as supplied by ownBuilder, is `acmecloud_1.7.0-release-unsigned.apk`.

To create your certificate copy the following command, replacing `acme-release-key.keystore` and `acme_key` with your own keystore name and alias, which can be anything you want. The keystore name and alias must both have a password, which can be same for both. Then enter your company information as you are prompted:

```
keytool -genkey -v \
   -keystore acme-release-key.keystore \
   -alias acme_key \
   -keyalg RSA -keysize 2048 \
```

```
    -validity 10000
```

```
Enter keystore password:
Re-enter new password:
What is your first and last name?
 [Unknown]:  Acme Boss
What is the name of your organizational unit?
 [Unknown]:  Acme Headquarters
What is the name of your organization?
 [Unknown]:  Acme, Inc.
What is the name of your City or Locality?
 [Unknown]:  Anytown
What is the name of your State or Province?
 [Unknown]:  CA
What is the two-letter country code for this unit?
 [Unknown]:  US
Is CN=Acme Boss, OU=Acme Headquarters, O="Acme, Inc.", L=Anytown, ST=CA, C=US correct?
 [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA)
with a validity of 10,000 days
       for: CN=Acme Boss, OU=Acme Headquarters, O="Acme, Inc.", L=Anytown, ST=CA, C=US
Enter key password for <acme_key>
       (RETURN if same as keystore password):
[Storing acme-release-key.keystore]
```

Now use `jarsigner` to sign your app. Replace `acme-release-key.keystore` and `acme_key` with your own keystore name and alias:

```
jarsigner -verbose \
    -sigalg SHA1withRSA \
    -digestalg SHA1 \
    -keystore acme-release-key.keystore \
    acmecloud_1.7.0-release-unsigned.apk acme_key
```

```
Enter Passphrase for keystore:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/ACME_KEY.SF
  adding: META-INF/ACME_KEY.RSA
 signing: res/anim/disappear.xml
 signing: res/anim/grow_from_bottom.xml
 [...]
 jar signed.

 Warning:
 No -tsa or -tsacert is provided and this jar is not timestamped.
 Without a timestamp, users may not be able to validate this jar after the signer
```

```
 certificate's expiration date (2042-07-28) or after any future revocation date.
```

You can ignore the warning at the end; you should see a `jar signed` message when it is finished.

Now you can verify that your app is signed:

```
jarsigner -verify -verbose -certs acmecloud_1.7.0-release-unsigned.apk
```

```
sm       943 Thu Mar 12 12:47:56 PDT 2015
res/drawable-mdpi/abs__dialog_full_holo_light.9.png

X.509, CN=Acme Boss, OU=Acme Headquarters, O="Acme, Inc.", L=Anytown, ST=CA, C=US
```

This will spit out hundreds of lines of output. If it ends with the following it's good:

```
...
s = signature was verified
m = entry is listed in manifest
k = at least one certificate was found in keystore
i = at least one certificate was found in identity scope

jar verified.
```

The last step for preparing your `.apk` for release is to run `zipalign` on it. `zipalign` optimizes your file to use less memory. You must specify both an input and an output file, so this is good time to give your app a shorter name, and it should not say "unsigned". Our example file will be renamed to `acmecloud_1.7.0.apk`:

```
zipalign -v 4 acmecloud_1.7.0-release-unsigned.apk acmecloud_1.7.0.apk
```

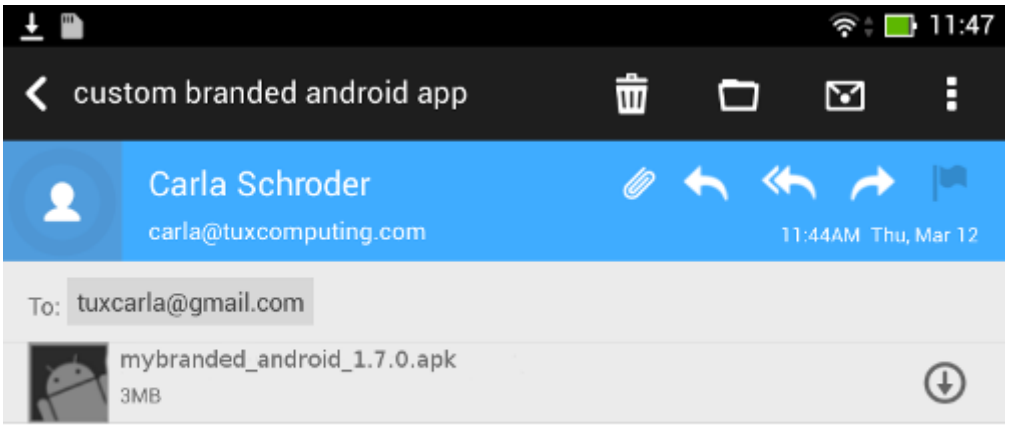
```
Verifying alignment of acmecloud_1.7.0.apk (4)...
     50 META-INF/MANIFEST.MF (OK - compressed)
  13277 META-INF/ACME_KEY.SF (OK - compressed)
  27035 META-INF/ACME_KEY.RSA (OK - compressed)
  28206 res/anim/disappear.xml (OK - compressed)
  [..]
  Verification succesful
```

Again, this emits a lot of output, and when you see **Verification succesful** at the end you know it succeeded, and it is ready to distribute.

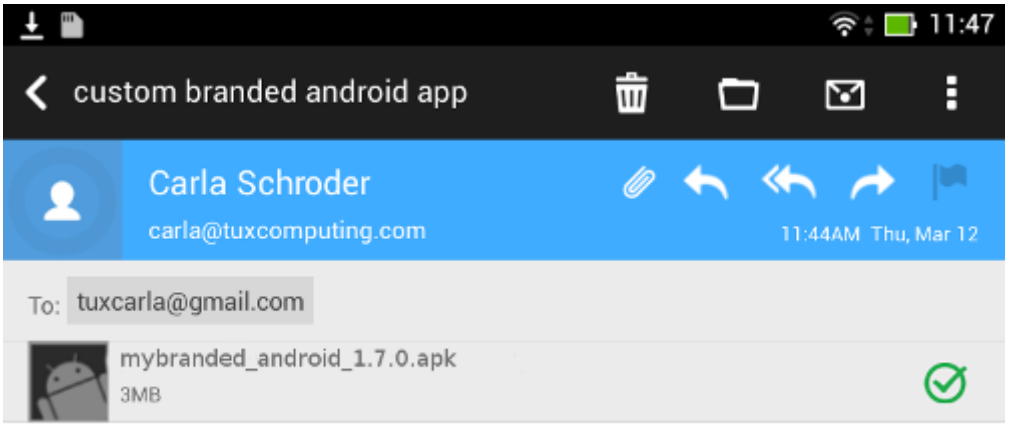
Distribution via Email

You can download your branded Android app from your account on customer.owncloud.com, and

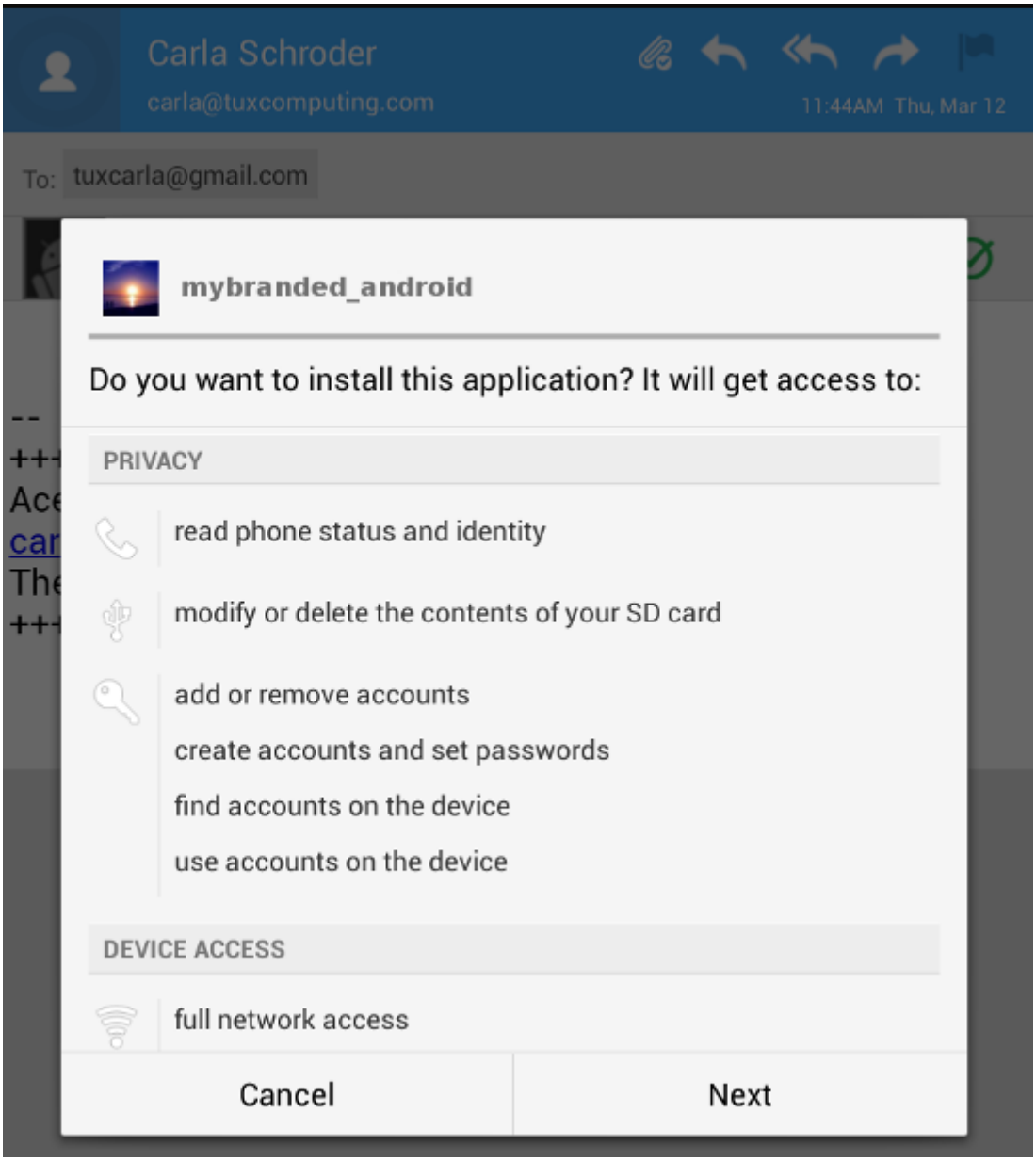
send it as an email attachment to your users. (This is not the optimal way to distribute it as it is over 2 megabytes in size.) When they open your email on their Android phone or tablet, they must first click the the download arrow (bottom right of the screenshot) to download your app.



When the arrow changes to a green checkbox, it has been downloaded.



Now your user must click on the green checkbox, and this launches the app installer, and all they have to do is follow the installation wizard to install your branded app.



When the installation is complete, the ownCloud Android App Manual contains instructions for using the app.

Publish On Your ownCloud Server

You can distribute your branded app from your ownCloud server. Simply upload it to your ownCloud server and share it like any other file: you can create normal ownCloud shares with ownCloud users and groups, and you may create a link share to share it with anyone. (See the **Sharing Files** section of the ownCloud Web Manual to learn more about sharing files.)

Publish to the Google Play Store

You may elect to publish your app in the Google Play store, either as a free or paid app. There are several steps to publishing a free app:

1. Create a Google Play Publisher account.
2. Sign your branded app with your own signing certificate.

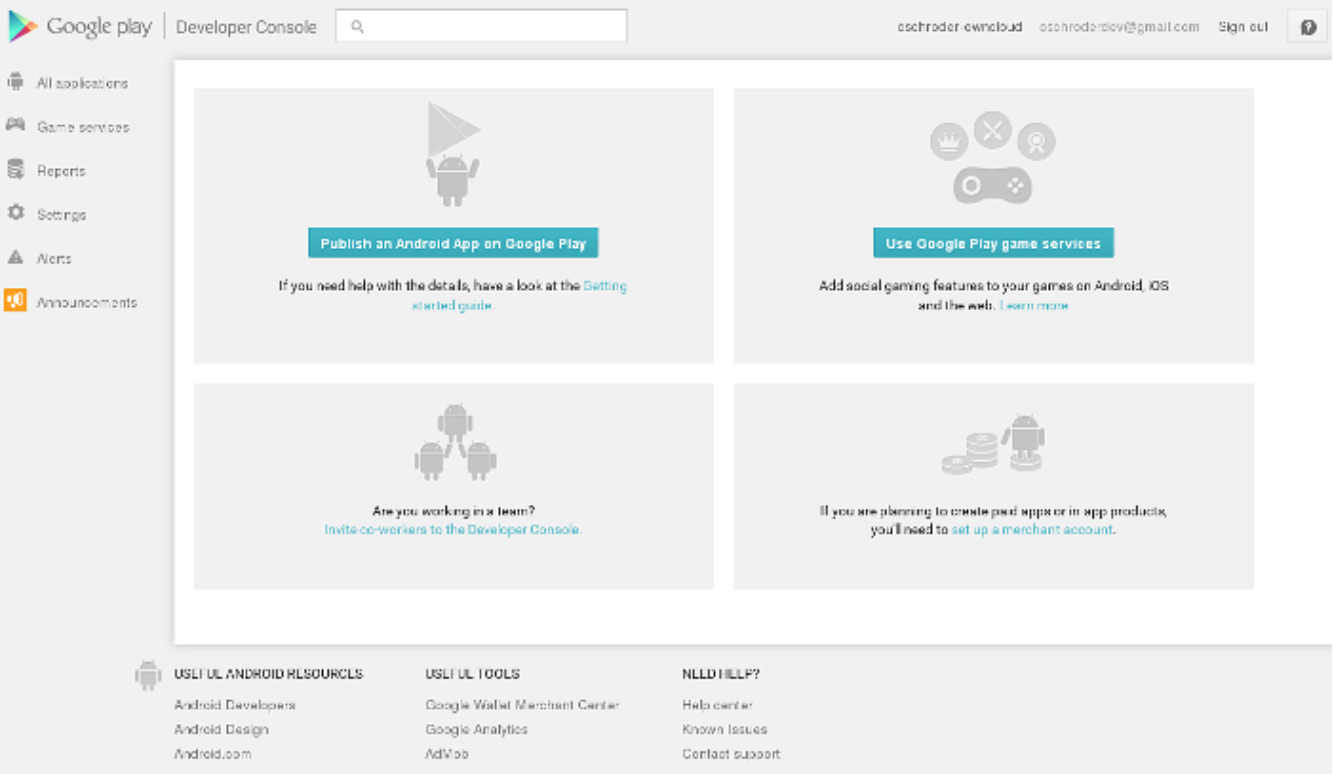
3. Upload your signed branded app to your Google Play Publisher account.

As part of creating your Google Play Publisher account you will have to create some screenshots of your app in specific sizes, and create a store description.

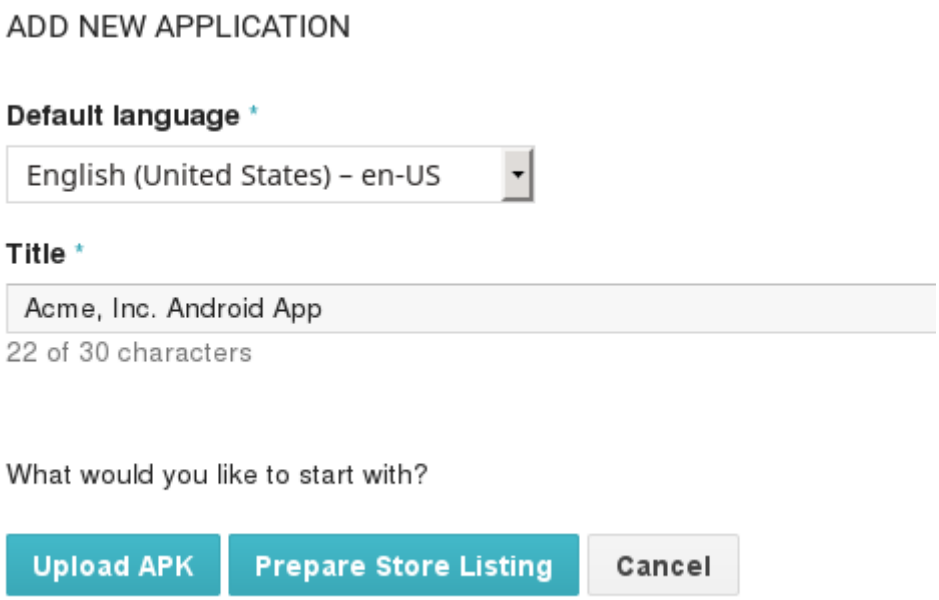
Create a Google Play Publisher Account

Start at Google's Get Started With Publishing page. Have a credit card ready, because it costs $25. If you already have a Google account, it is usually better to create a separate new account just for publishing apps to the Google Play Store.

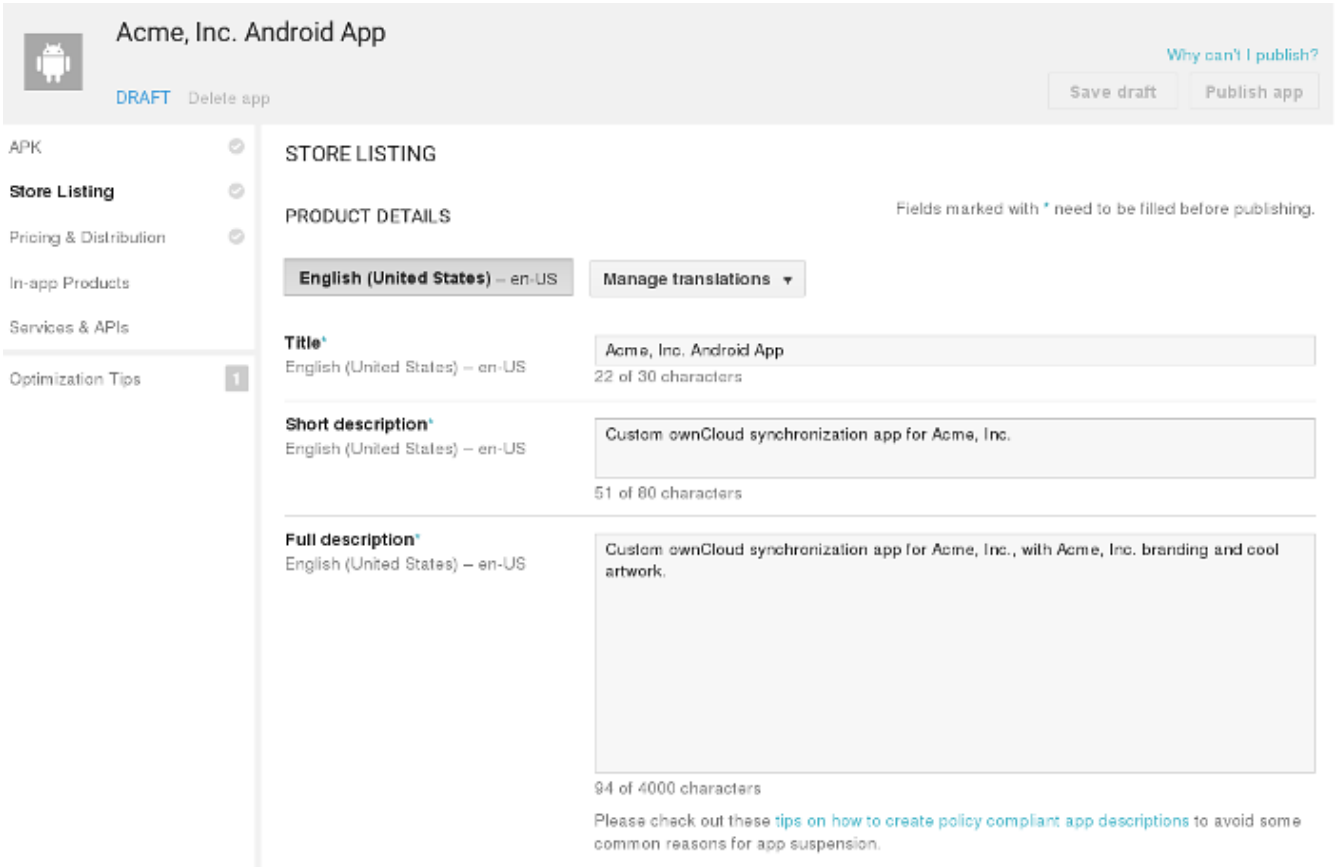
Google's process for uploading apps is fairly streamlined, and the most time-consuming task is creating all the required graphics. After registering, you'll see the welcome screen for the Google Dev Console. Click **Publish an Android app on Google Play**.



This opens the **Add New Application** screen. Click the **Prepare Store Listing** button. (Note that as you navigate the various screens, you can click the Save Draft button to preserve your changes.)



On the next screen, enter your product description.



Then you'll have to upload a batch of graphics in various sizes for the **Graphic Assets** section, like these images for a smartphone and seven-inch tablet. You are required to upload at least two images.

Screenshots *

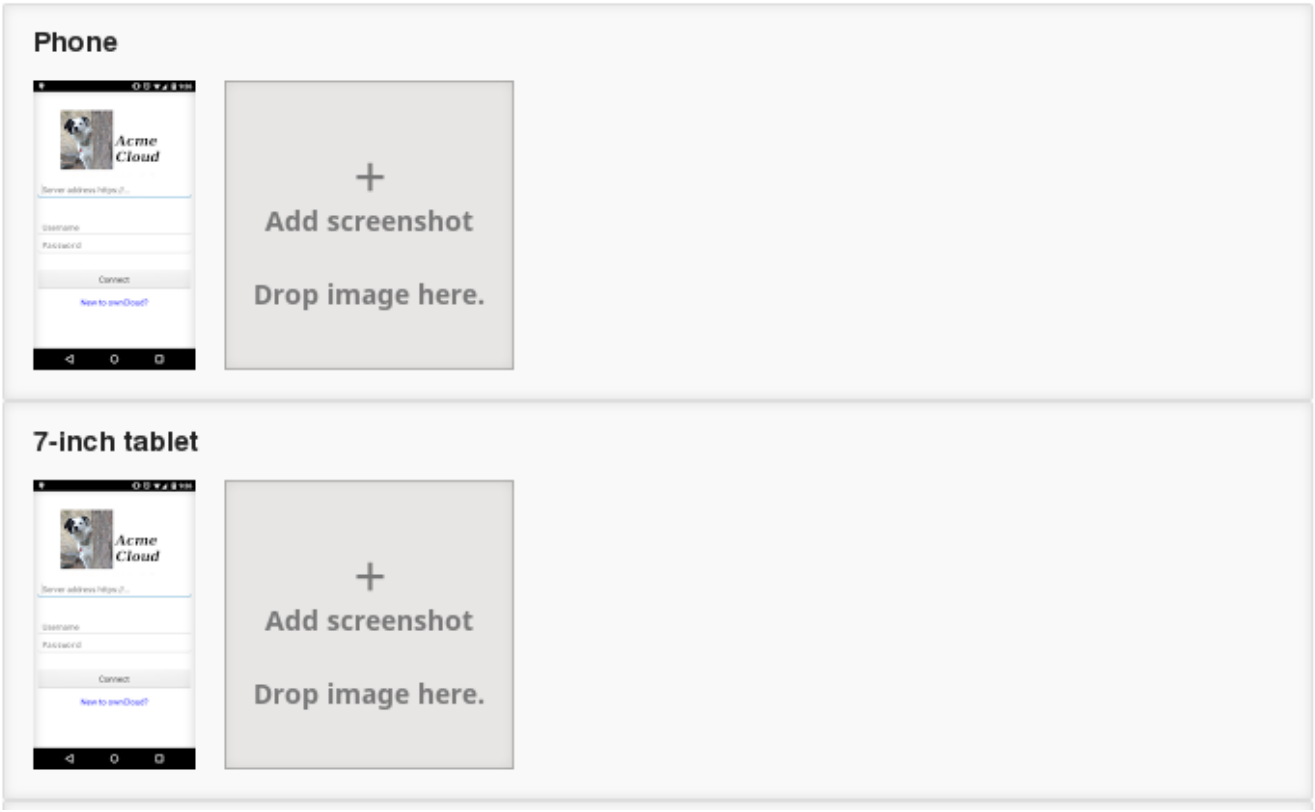
Default – English (United States) – en-US

JPEG or 24-bit PNG (no alpha). Min length for any side: 320px. Max length for any side: 3840px.

At least **2 screenshots are required** overall. **Max 8 screenshots per type**. Drag to reorder or to move between types.

For your app to be showcased in the 'Designed for tablets' list in the Play Store, you need to upload at least one 7-inch and one 10-inch screenshot. If you previously uploaded screenshots, make sure to move them into the right area below.

Learn how tablet screenshots will be displayed in the store listing.



You must also upload a 512x512-pixel logo, and a 1024x500 banner.

Hi-res icon *

Default – English (United States) – en-US

512 x 512

32-bit PNG (with alpha)

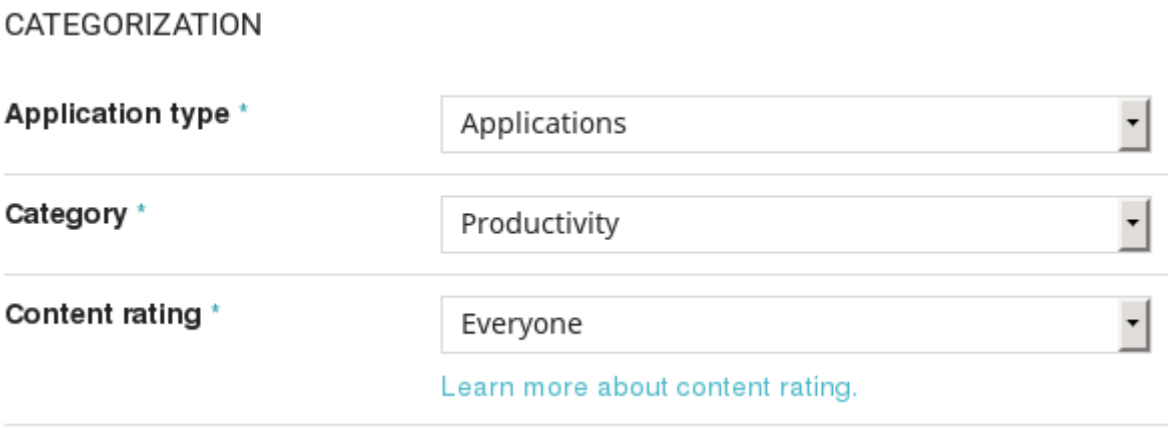
Feature Graphic *

Default – English (United States) – en-US

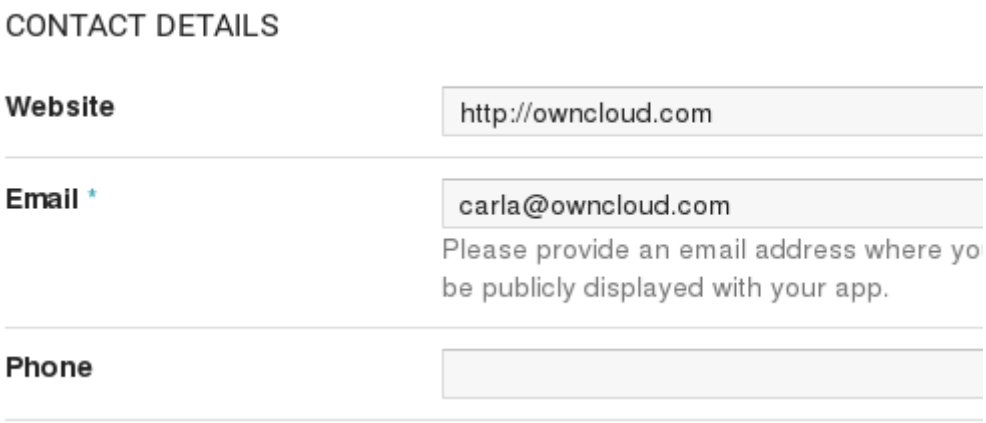
1024 w x 500 h

JPG or **24-bit PNG** (no alpha)

Now choose the store categories for your app.



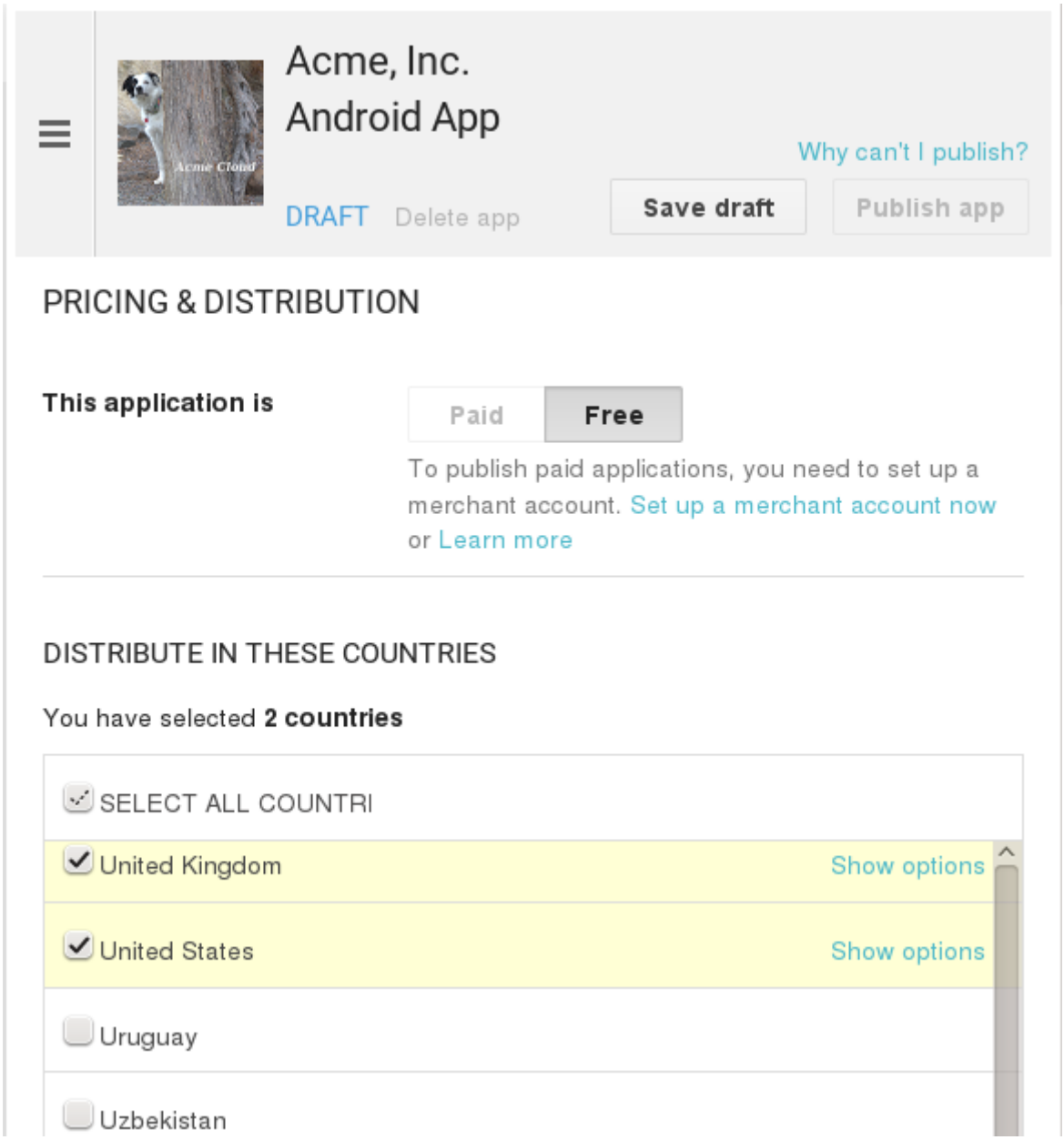
Then enter your contact information, which will be visible on your store listing.



On the next line you may optionally link to your privacy policy. It is recommended to have a privacy policy.

When you're finished with the **Store Listing** page, go to the **Pricing and Distribution** page. You may make this a paid or free app. You cannot convert a free app to paid. You may convert a paid app to free, but then you can't convert it back to paid. You'll have numerous options for paid apps, such as Android Wear, Android TV, and various Google marketing tie-ins, and many more.

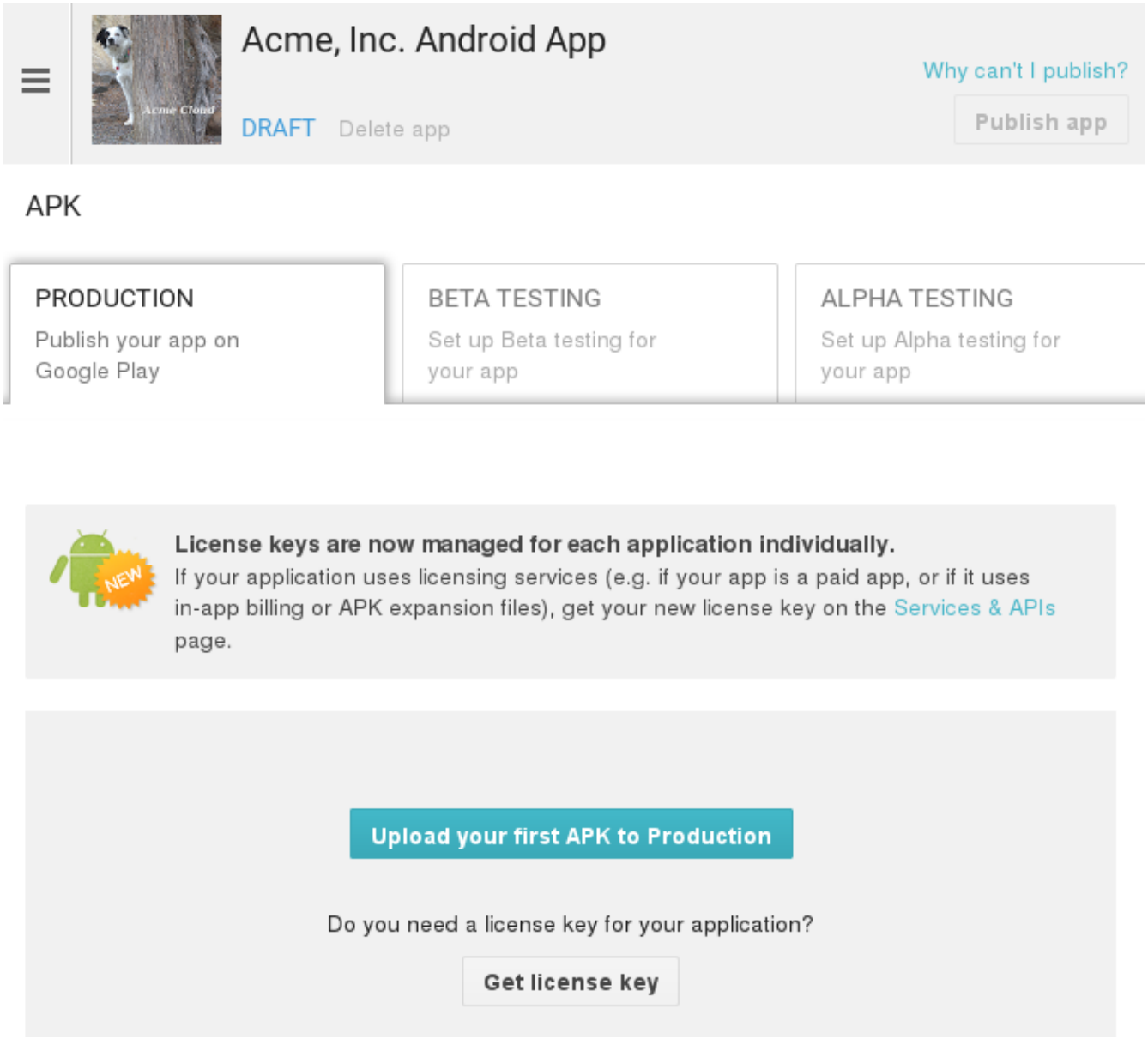
For now let's make this a free app, so click the Free button and select the countries you want to distribute it in.



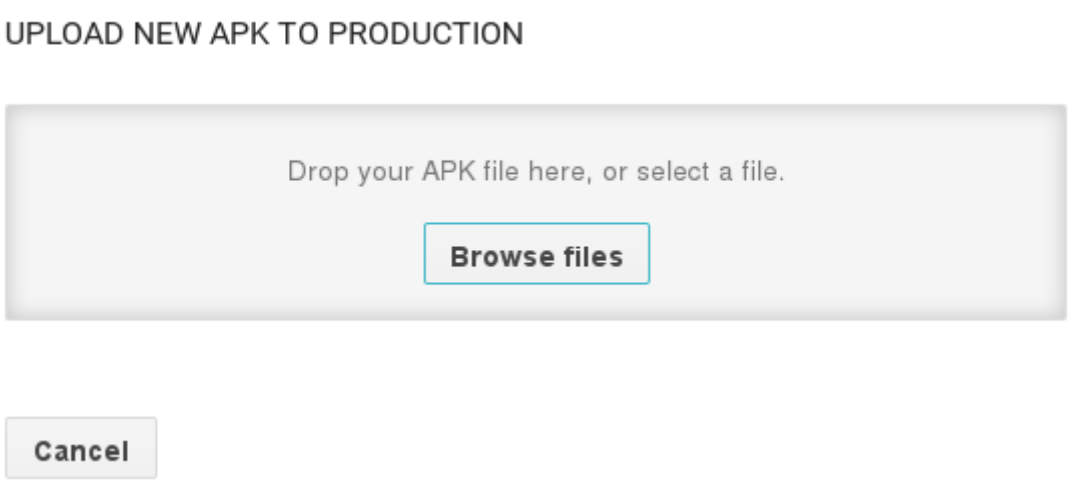
Now you may upload your app.

Uploading to Google Play Store

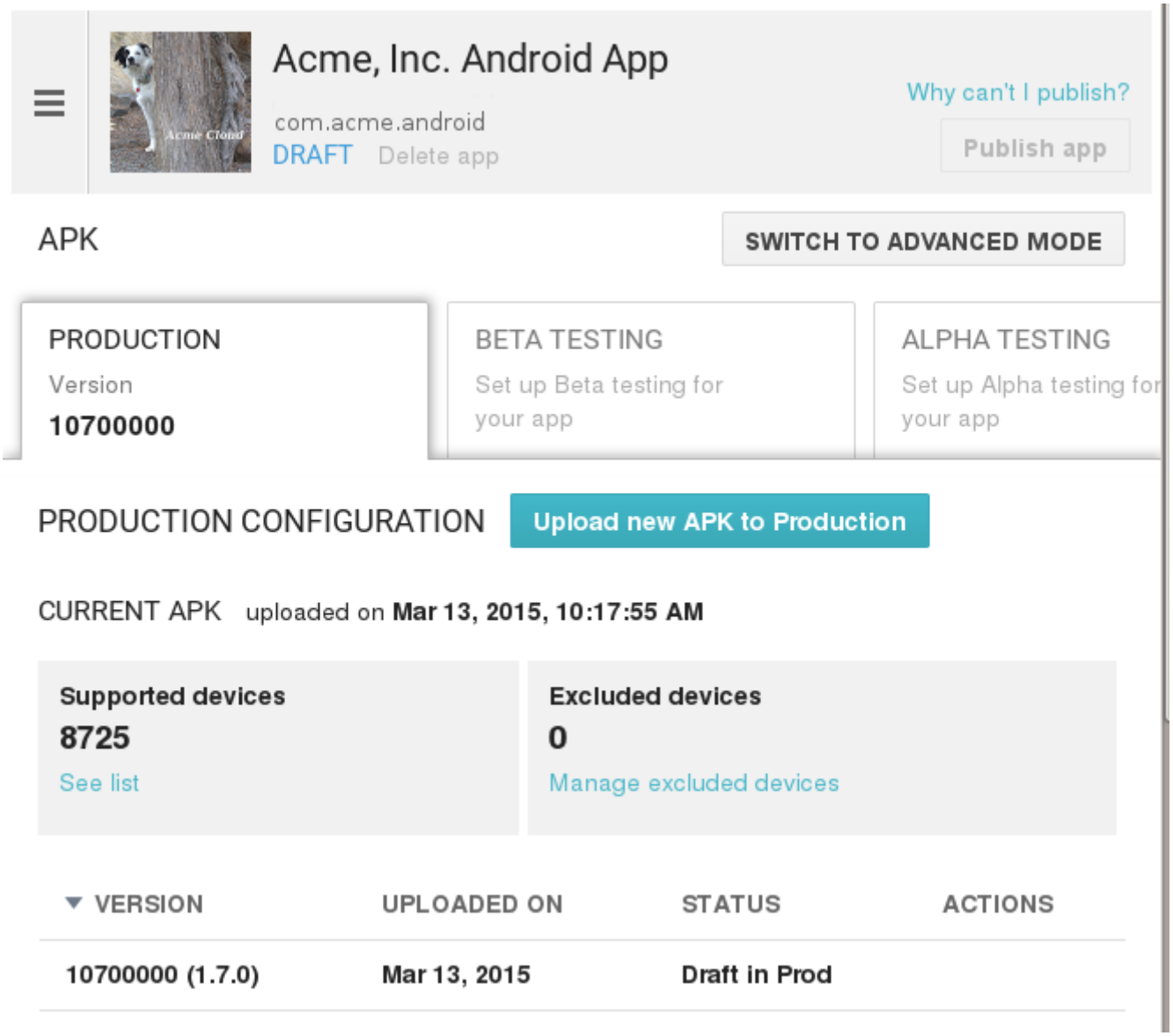
Now you can upload your app to your Google Play Store page. Go to the **APK** page and click **Upload your first APK to Production**. You don't need a license key for a free app.



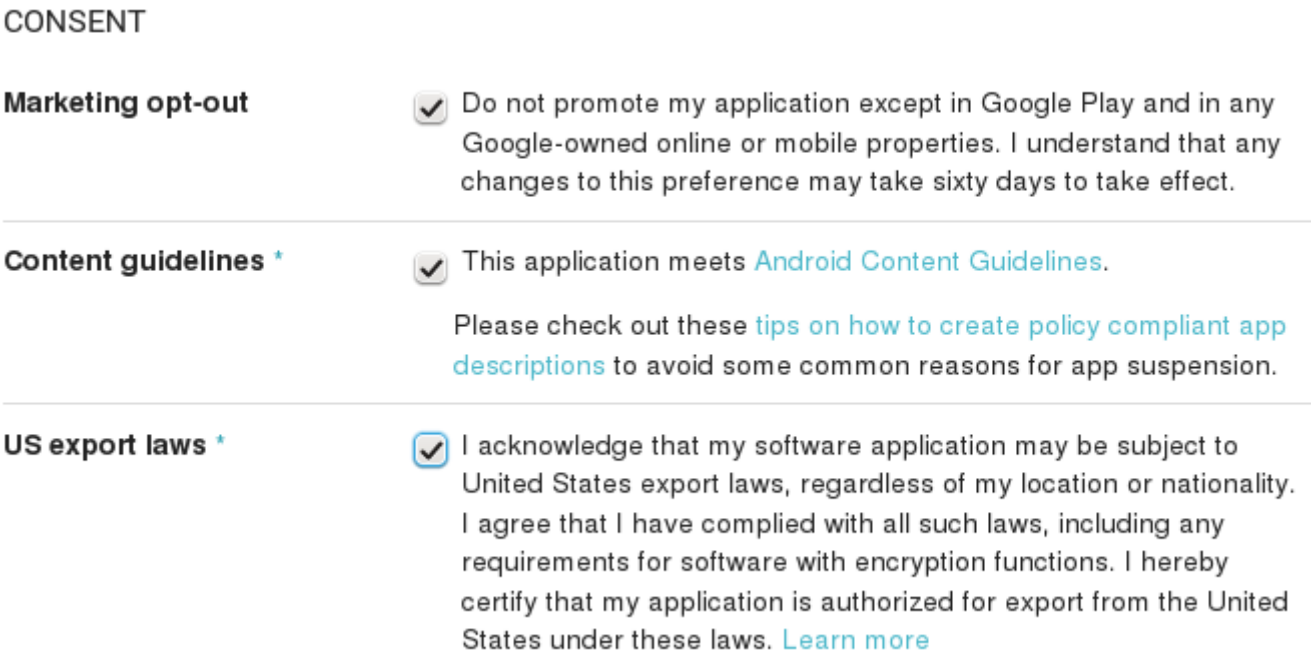
Drag-and-drop, or browse to select your app.



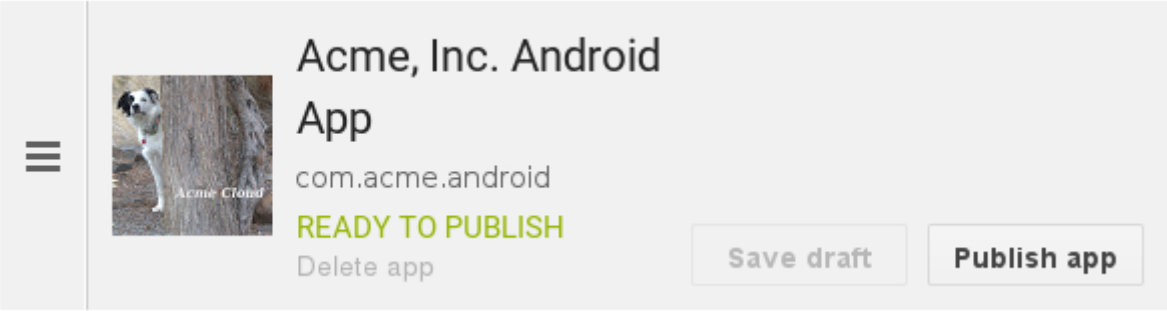
A successful upload looks like this:



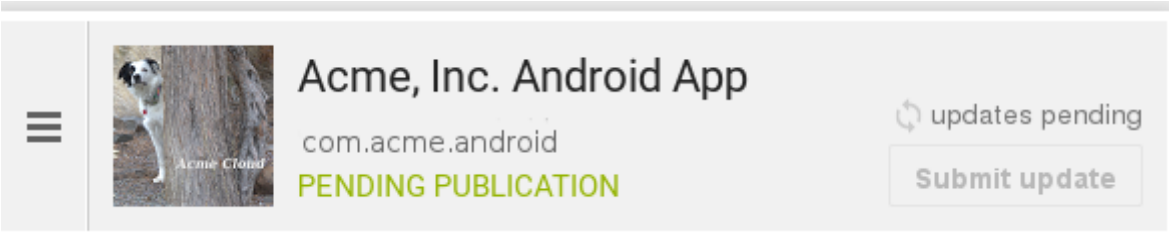
Your app is not yet published, but only uploaded to your account. There is one more step to take before you can publish, and that is to go back to the **Pricing & Distribution** page and fill out the **Consent** section.



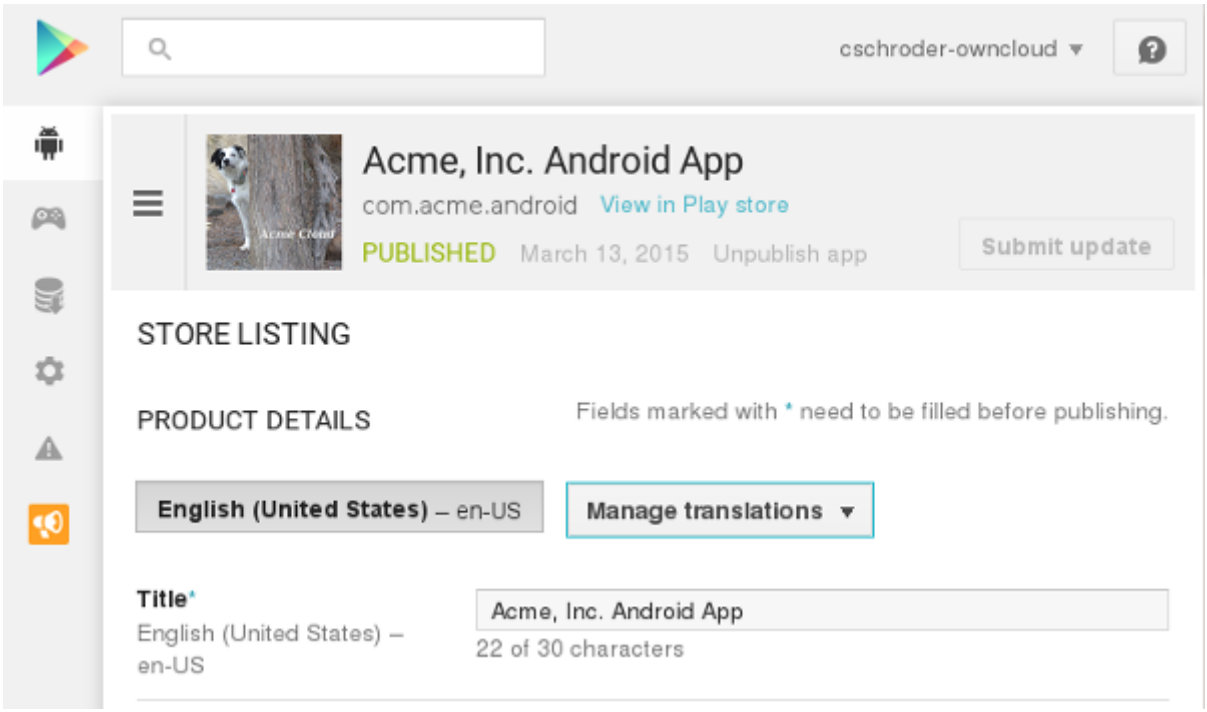
Click the Save Draft button, and if you followed all the required steps you should now see a **Publish App** button.



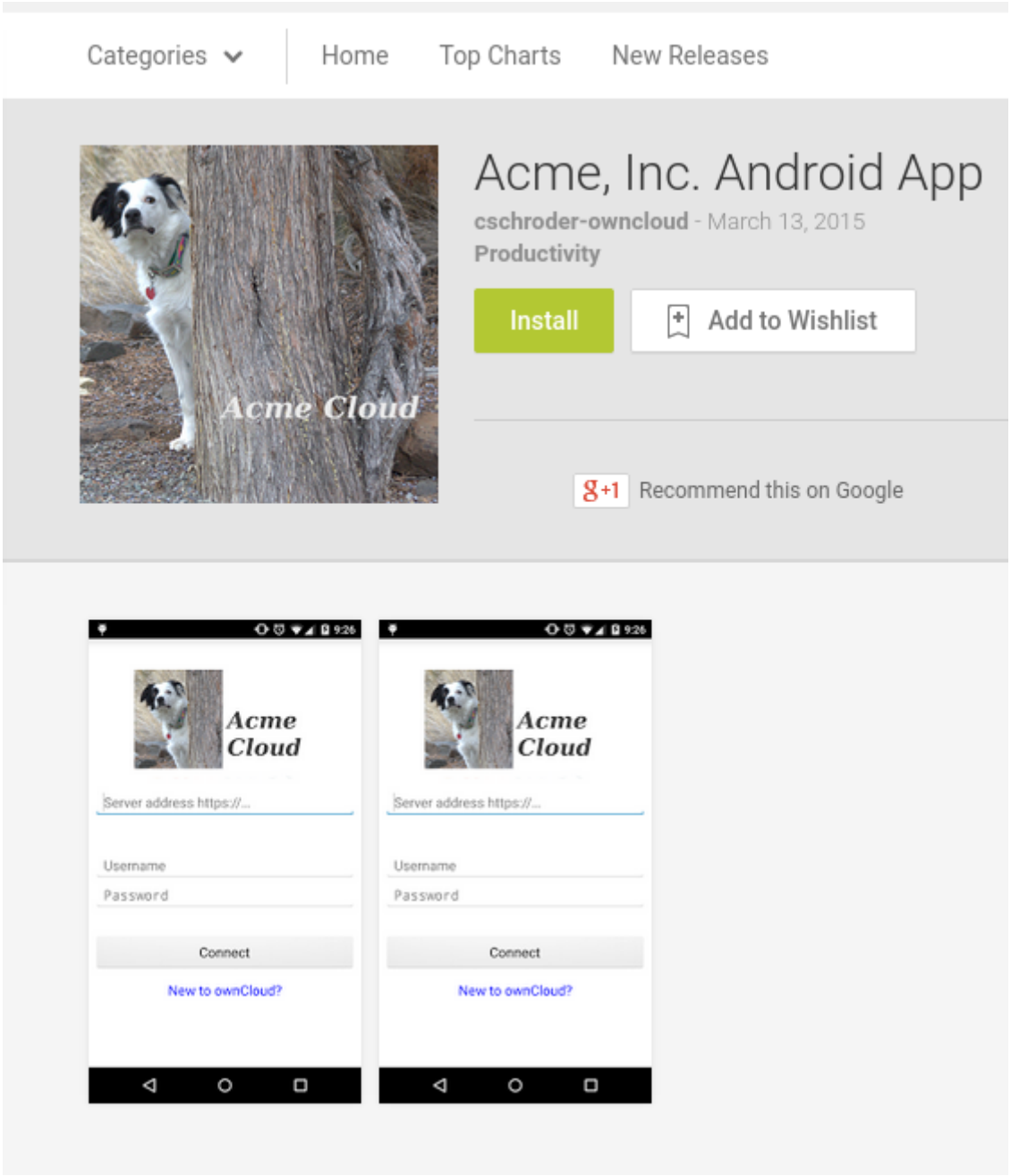
It will not be published immediately, but after review by Google, which usually takes just a few hours.



After it has been published, your store listing is updated as PUBLISHED, and it includes a link to your Play Store listing.



Now all you need to do is distribute the URL to your users, and they can install it either from their Web browsers, or from their Google Play Store apps. This is how it looks to your users.



Customize Download Link

You may configure the URLs to your own download repositories for your ownCloud desktop clients and mobile apps in config/config.php. This example shows the default download locations:

```
<?php

  "customclient_desktop" => "https://owncloud.com/desktop-app/",
  "customclient_android" =>
  "https://play.google.com/store/apps/details?id=com.owncloud.android",
  "customclient_ios"     =>
  "https://itunes.apple.com/us/app/owncloud/id543672169?mt=8",
```

Simply replace the URLs with the links to your own preferred download repos.

You may test alternate URLs without editing config/config.php by setting a test URL as an environment variable:

```
export OCC_UPDATE_URL=https://test.example.com
```

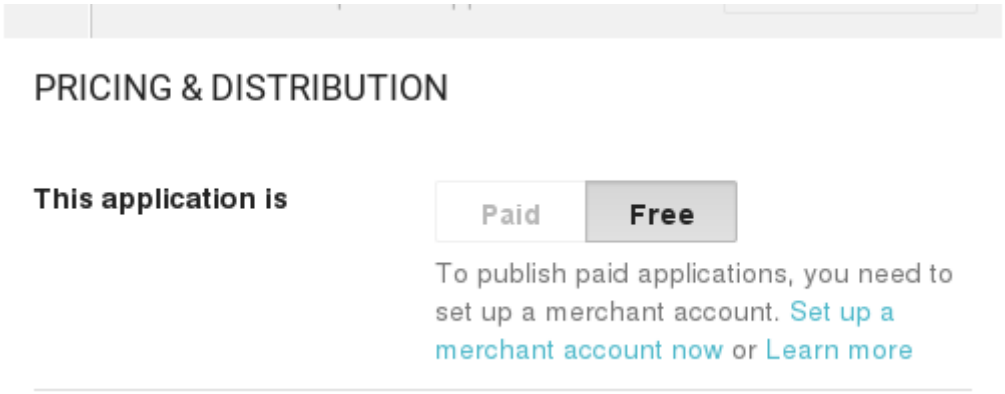
When you're finished testing you can disable the environment variable:

```
unset OCC_UPDATE_URL
```

Publishing a Paid App in Google Play

If you would rather not give your branded app away you can sell it on Google Play. You may convert a paid app to free, but you may not convert a free app to paid.

You must establish a Google Wallet Merchant Account. On your Google Dev Console click the **Learn more** link under the Free/Paid button for a nice thorough review of the process and tools. It requires verifying your business information and bank account, and you should expect it to take 3-4 days.



When you're ready to set it up, click the **Set up a merchant account now** link under the Free/Paid button.

Resources

- Get Started With Publishing
- Signing Your App Manually
- Developer Console

Update to Android App Bundle (after August 2021)

Introduction

Since August 2021, Google Play requires the Android App Bundle (.aab) for publishing new apps. [1]. With this change, the APK has been replaced as the standard publishing format. The ownBrander now generates 3 artifacts with every build:

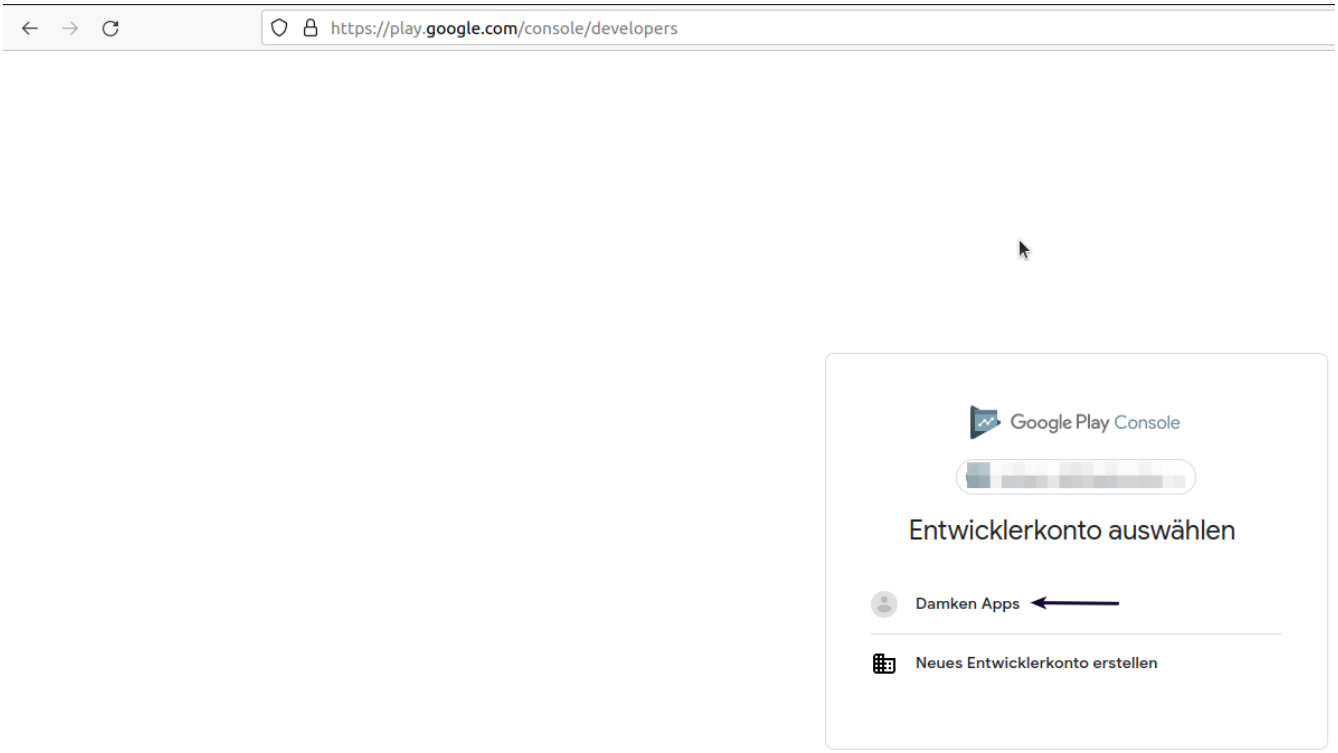
- `*-release.aab`: Android App Bundle for Play Store distribution after August 2021
- `*-release.apk`: Needs signing. Use for distribution methods other than Play Store
- `*-debug.apk`: Install directly to your device for debugging purposes

For the Android App Bundles, Play App Signing is required [2]. Play App Signing is a safety feature provided by Google. Every new release will automatically be signed by Google. (With this, apps no longer need to be signed locally.)

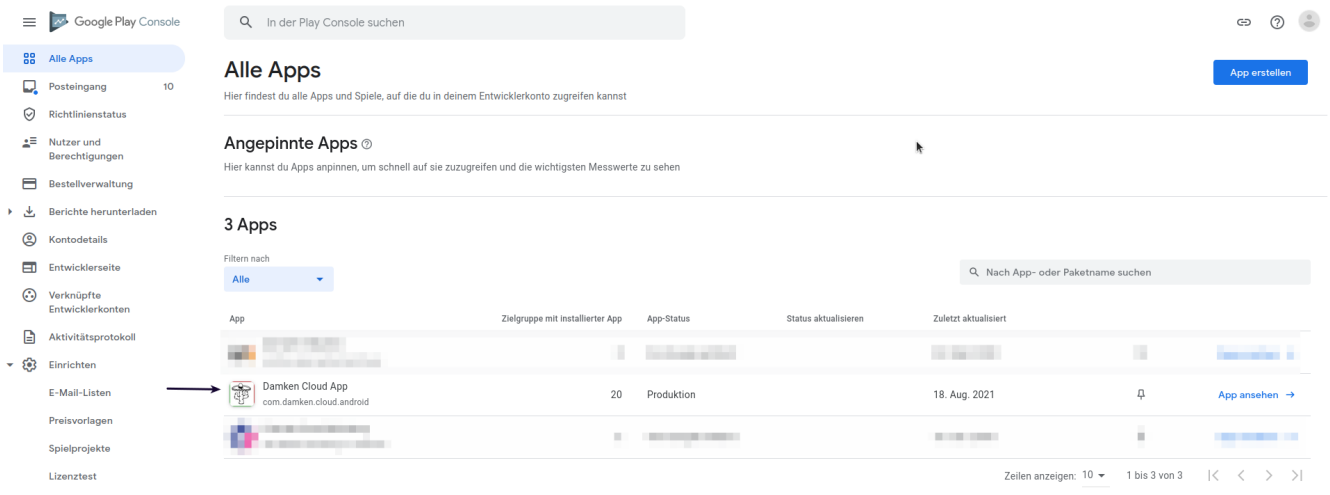
Navigate to the Google Play Console Page and click on the Play Console Button.

https://play.google.com/console/about/

You will then land on the developer account sign-on page. https://play.google.com/console/developers. After successful log-on, choose the appropriate developers account.

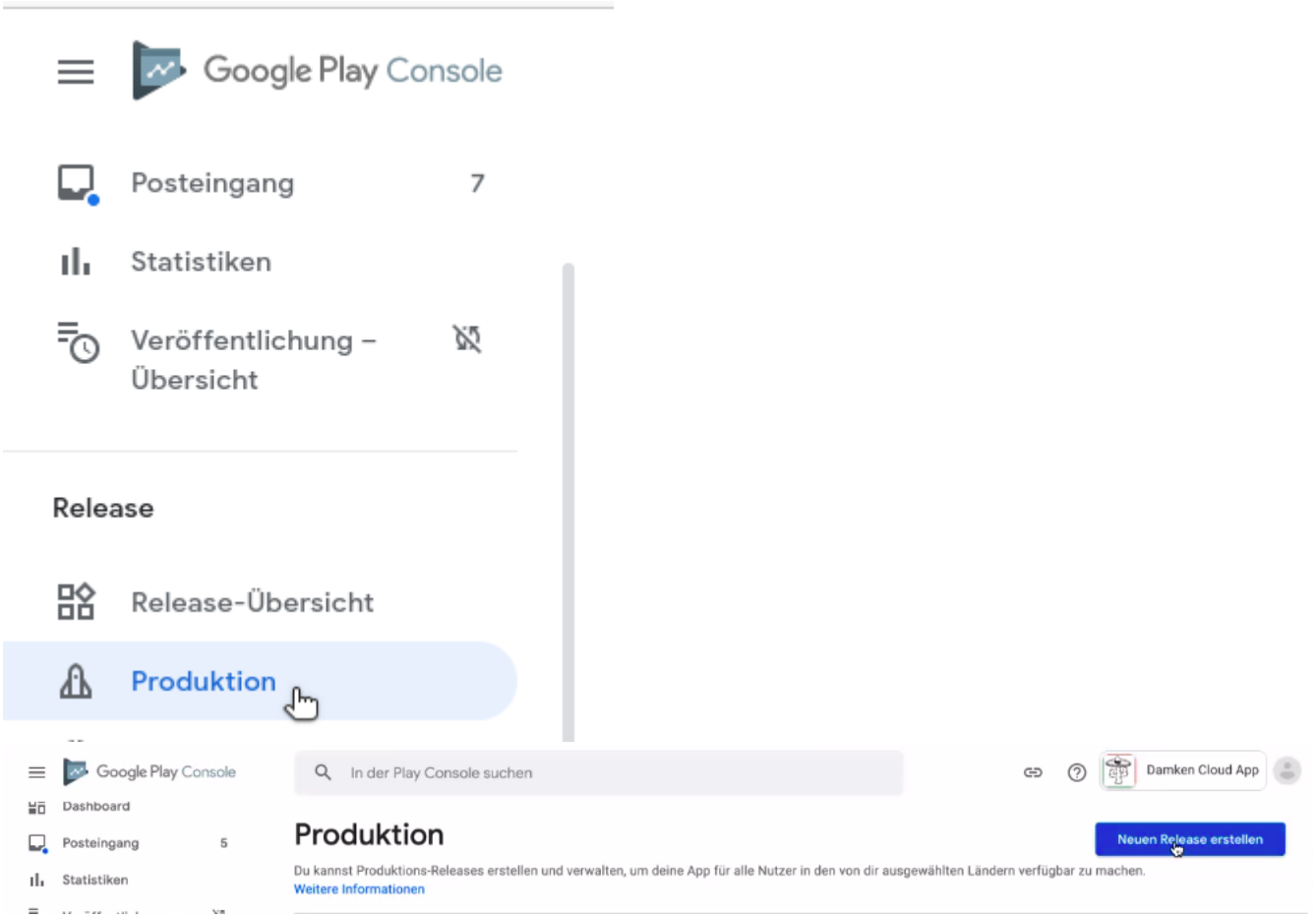


The page should appear like this. Choose the designated app, which is to be signed and/or updated.

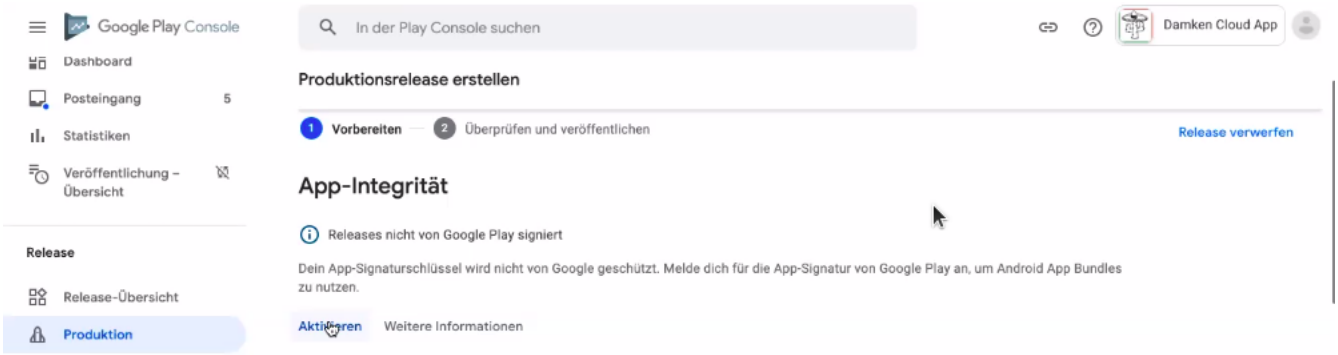


Create New Release and Activate App Integrity

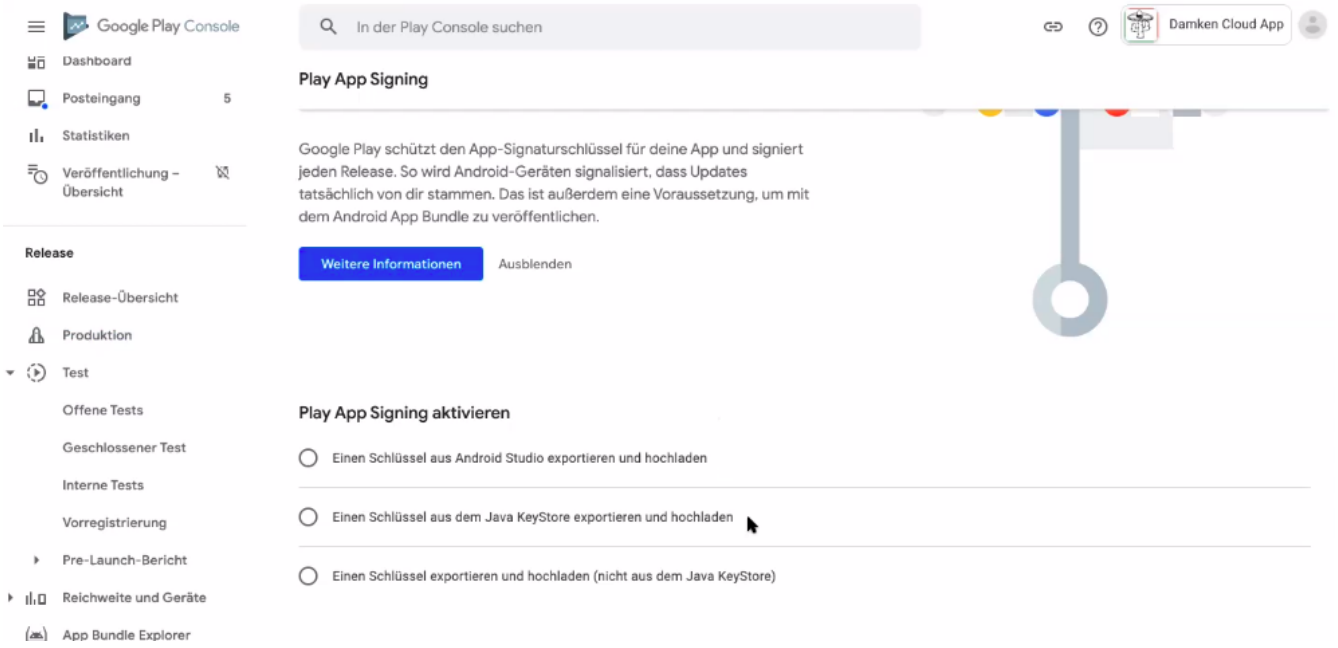
Navigate to Production, click and continue by clicking the "create new release" button.



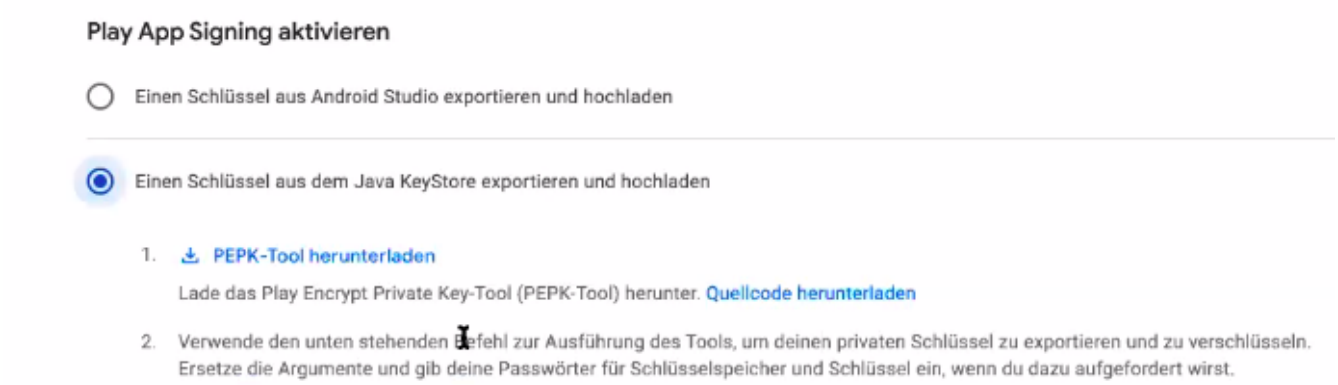
Google Play Console will guide you through this. Follow the steps to validate your app-Integrity. Click the "activate" button.



Since this is a new release version candidate, and the key comes from the Java KeyStore, choose this option. Otherwise, choose according to your existing key scheme.



Follow the instructions pertaining to the chosen key scheme. (Here: JavaKeyStore) Download the PEPK-Tool as instructed.



Gather your keystore parameters:

- KeyStore Alias
- Key Store Password
- Key Alias Password

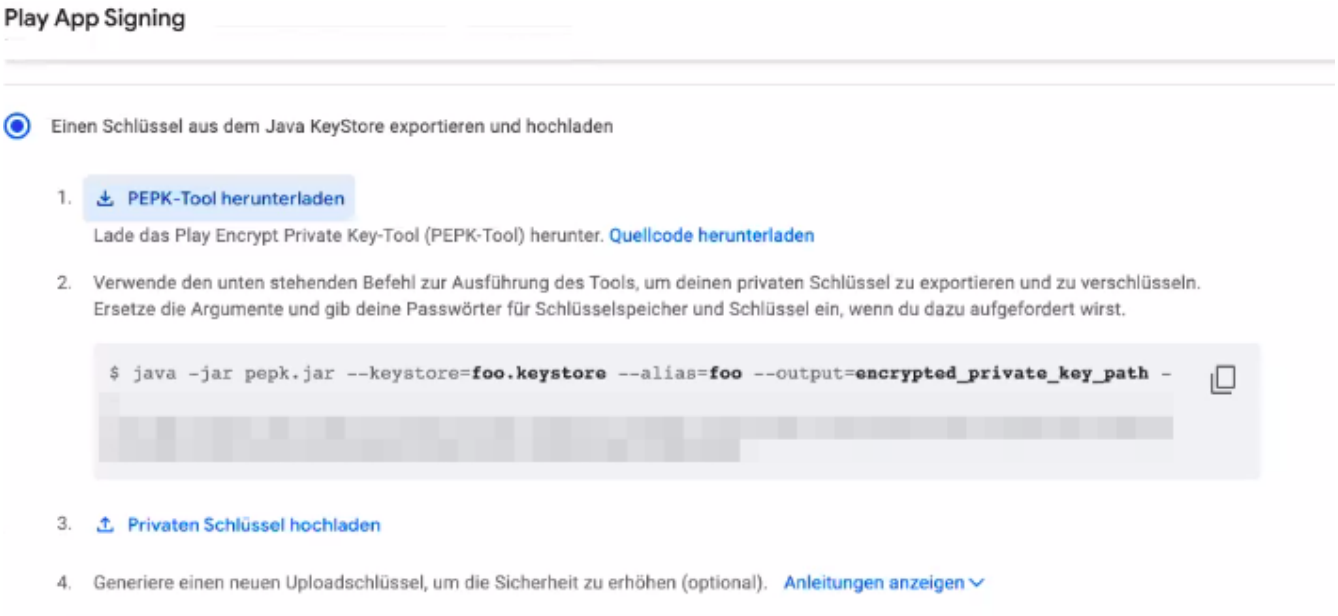
These are the parameters you previously used to sign your app. Perhaps you kept them stored in the ownBrander.

Input the command below into an open terminal window. As seen in the screen after the command, you can click on the copy to clipboard icon (Step 2) to copy the entire command block, but you must modify the "foo" parts of the command as follows:

- `foo.keystore` must be replaced by the `keystore name` (here: damken)
- `Alias` is the `app name` (here: damkencloud) aka the `Key Alias`.

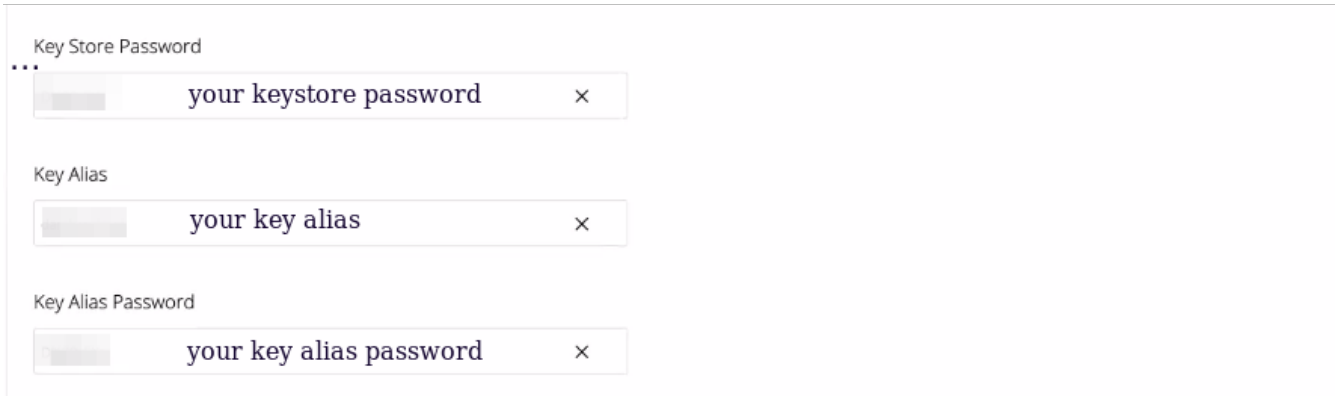
```
java -jar pepk.jar --keystore=foo.keystore --alias=foo --output
```

```
=encrypted_private_key_path
```

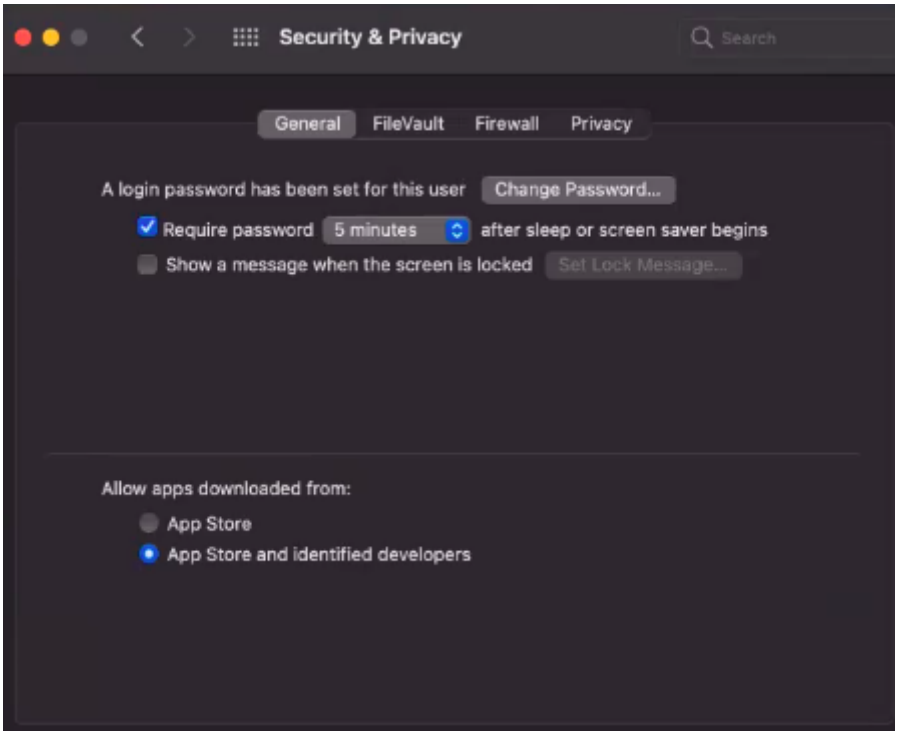


After executing the above command, you will be prompted to enter both "your keystore password" and then "your key alias password". Keep in mind that these passwords remain invisible while you type them. (Multiple entries or typos lead to errors.)

You can also check your signing parameters in ownBrander, if you have uploaded them for previous versions.



In case you download the PEPK tool on macOS, you'll need additional permissions in the macOS "Security & Privacy" settings:



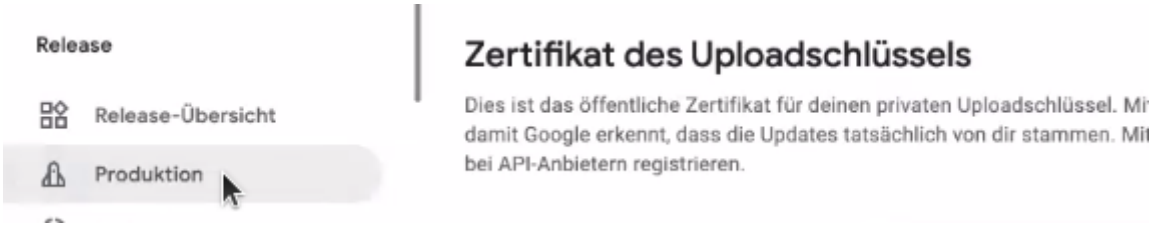
Proceed by clicking on the button to upload your private key.

Double-click the file to upload it. After a successful upload, click on the "save" button (bottom right of the page).

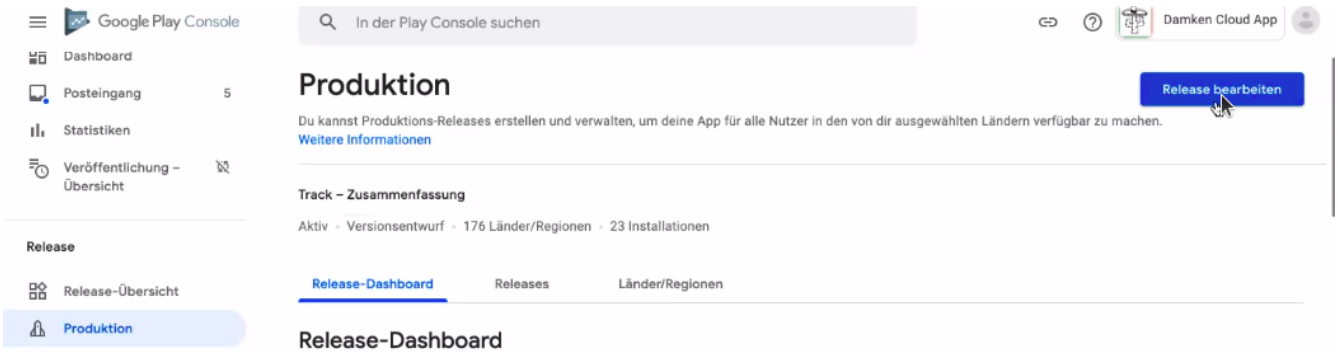
The following is a depiction of the upload file.

You will be redirected to the "Terms of Use" page. Click "Accept" after reading them.

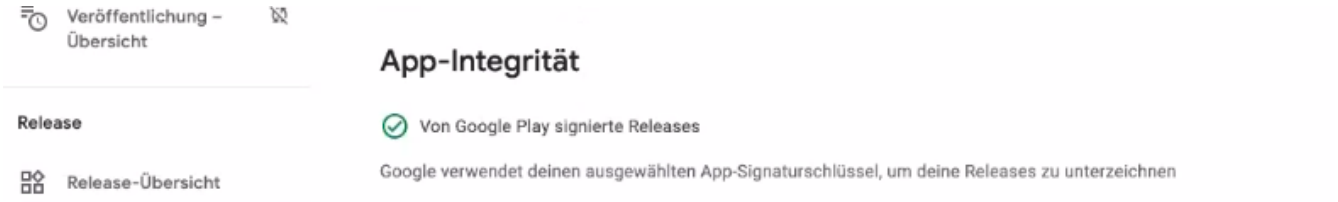
Navigate back to "Production". The certificate is now displayed.



In the "Production" bar move to the release review button.

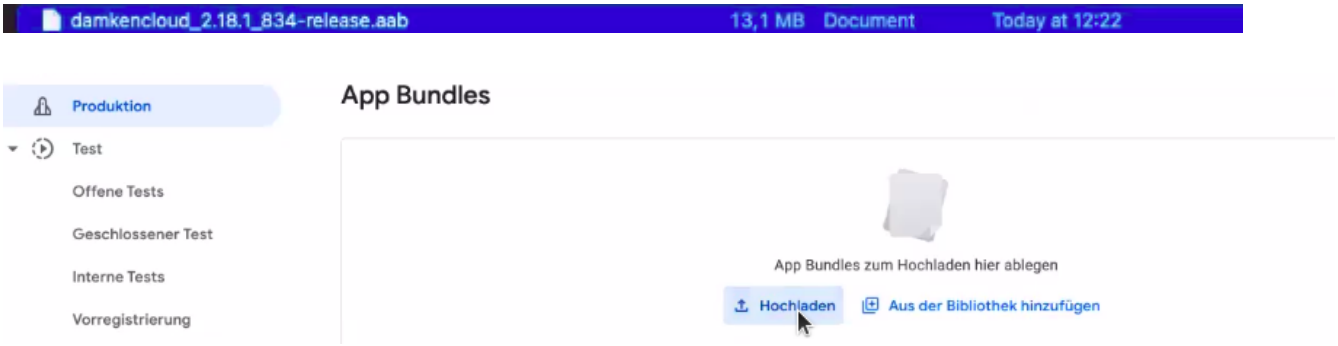


Notice the check-mark by the "App Integrity" field.

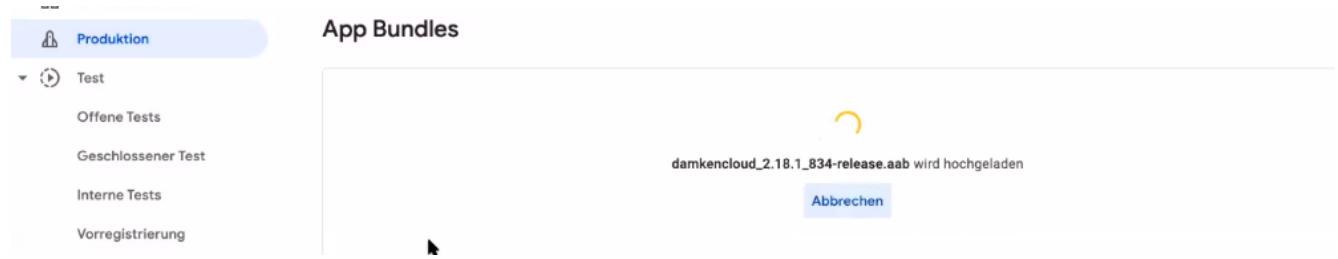


Upload Android App Bundle

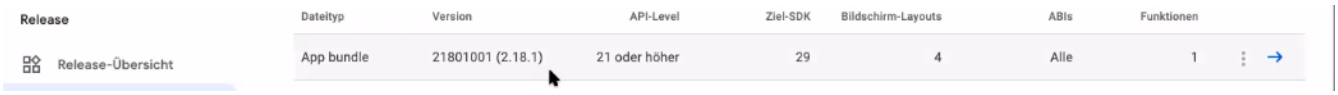
Proceed by clicking on "upload" in order to upload the `*-release.aab` file you previously downloaded from your shared account (personal folder) on customer.owncloud.com.



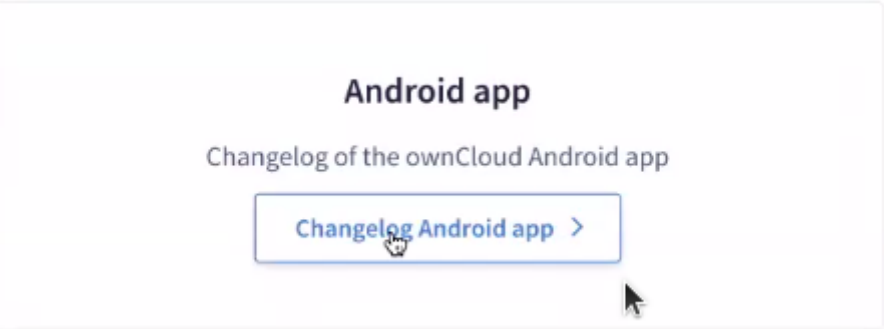
You should see Google's colors changing during the upload process, then a preview of the app release candidate.



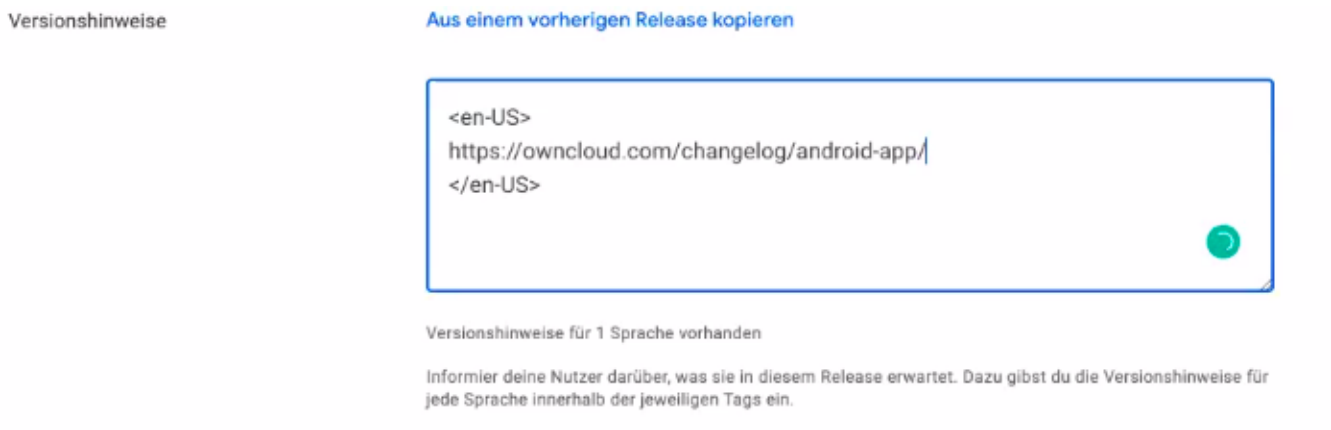
The new version is available and should be thoroughly examined before releasing it to the production environment.



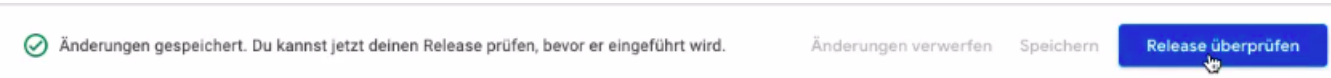
Optionally, you may choose to provide your users with information regarding the change log so they know which changes have been implemented.



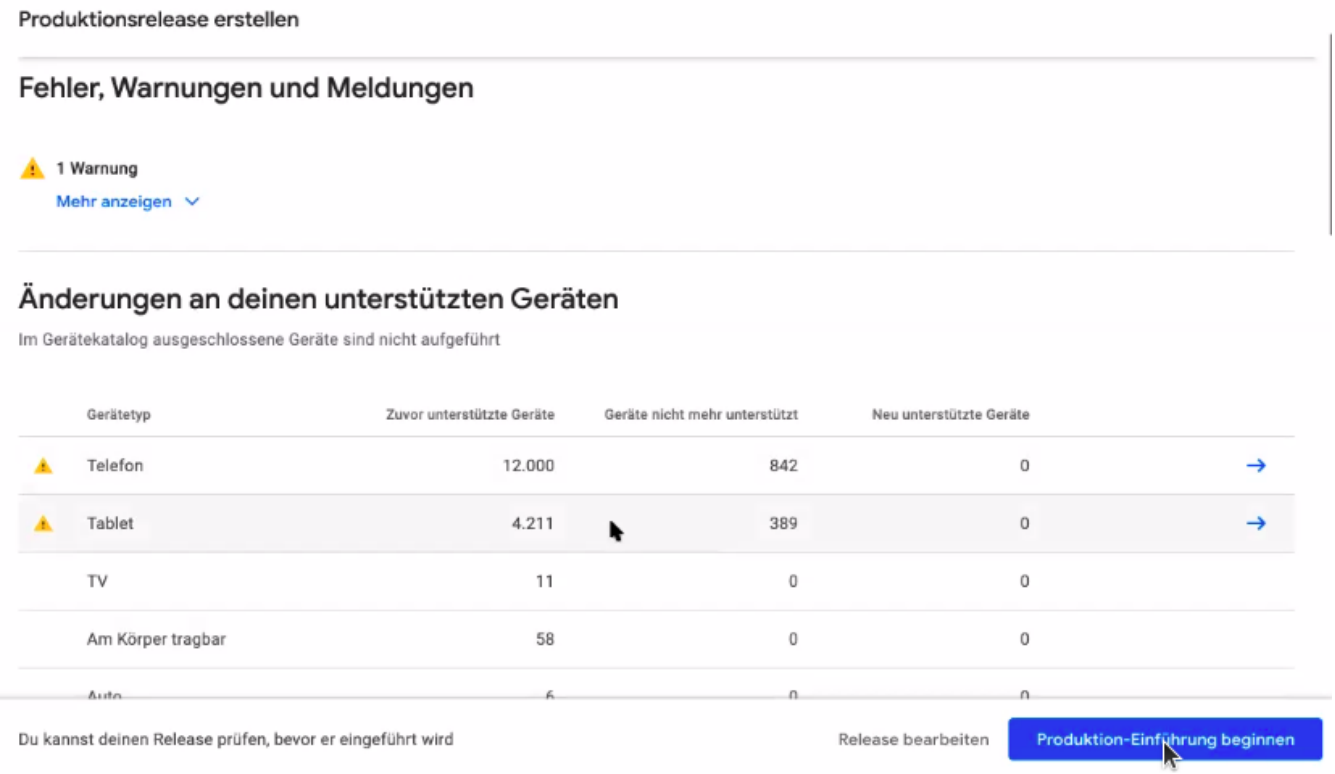
If available from a previous release, just copy it.



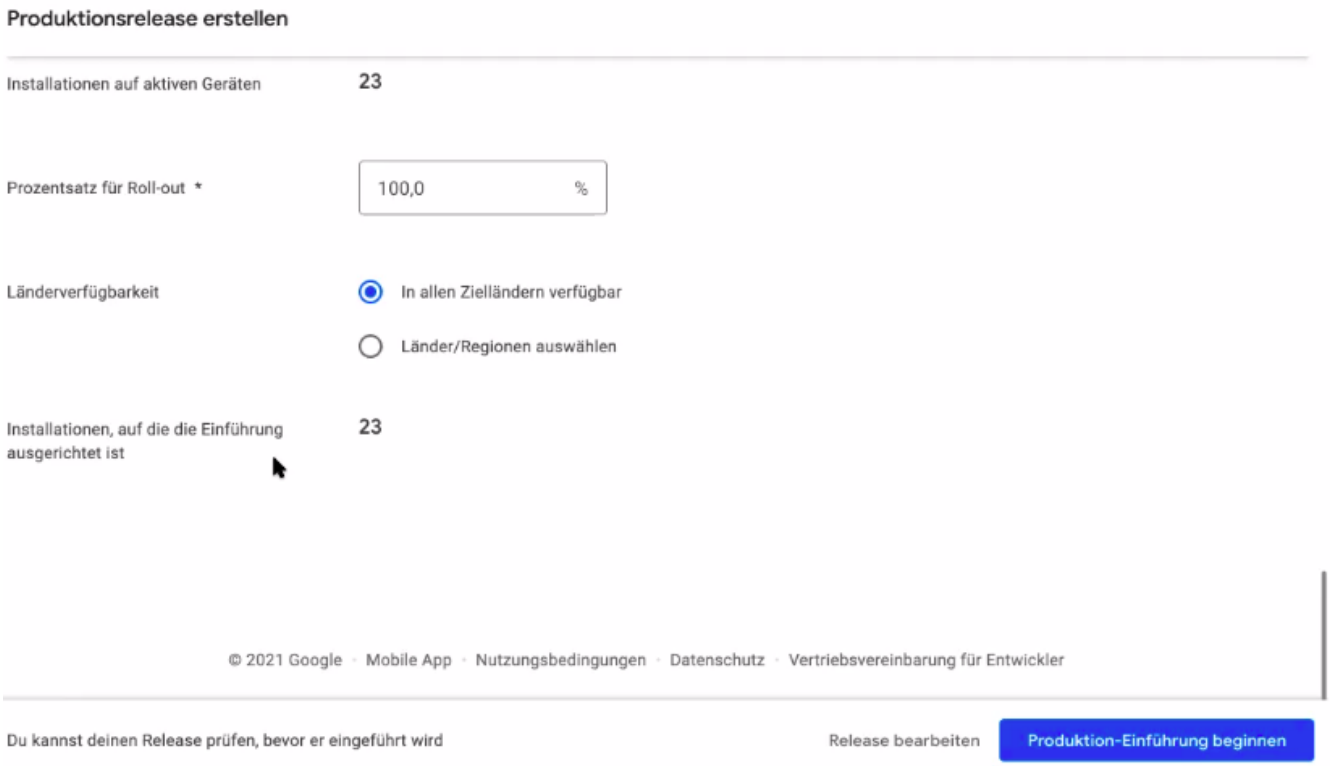
After saving any modifications, proceed by clicking the "check release" button.



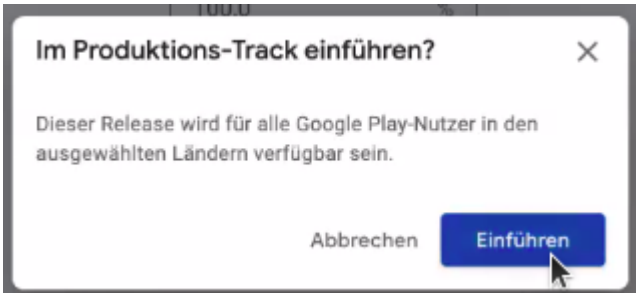
You may see some warning signs. (If of importance, check to see in what regards they are.) Scroll down.

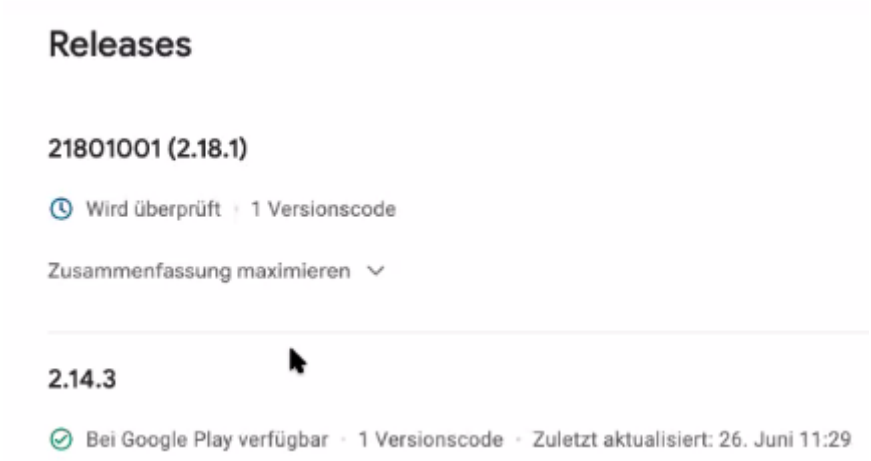


If you opt for a complete roll-out in all of the chosen distribution countries, just click the "Begin Production Release" button.

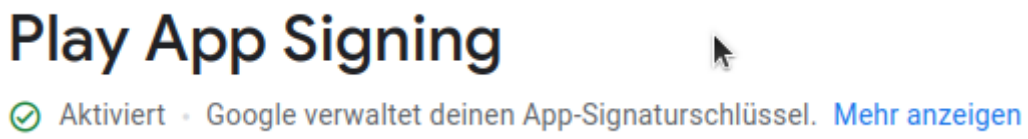


Review the chosen distibution and hit the "Release" button. Thereafter, you will receive a release status notification.

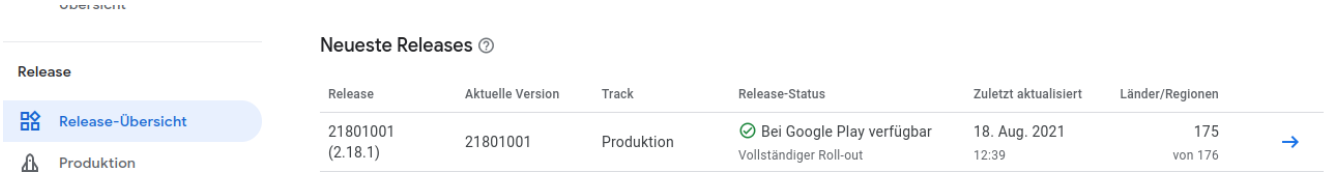




Notice, that Play App Signing has been successfully implemented.



Afterwards, you can find the status of your release/update release candidate in the tab "Release-Overview" or "Release Dashboard?"



Building and Distributing Your Branded iOS App

Introduction

Building and distributing your branded iOS ownCloud app involves a large number of interdependent steps. The process is detailed in this chapter over several pages. Follow these instructions exactly and in order, and you will have a nice branded iOS app that you can distribute to your users.

Prerequisites

- A Mac OS X computer with Xcode (free download) and Keychain Access (included in Utilities). This computer is essential to the entire process and will be linked to to your iOS Developer account. You will use it create and store distribution certificates, and to upload your app to iTunes Connect.
- An iOS developer account on developer.apple.com/ios, which costs $99 per year. Or an Enterprise account for $299/yr. The developer account limits you to testing on 100 devices of each type (Apple TV, Apple Watch, iPad, iPhone, iPod Touch) which must be registered in your account. The Enterprise account allows testing on unlimited, unregistered devices.
- An ownCloud Enterprise Subscription, with the ownBrander app enabled on customer.owncloud.com
- Some iPhones or iPads for testing your app. Again, if you have the $99 developer account each device must have its UDID registered in your account on developer.apple.com.

Procedure

You need the Apple tools to build eight provisioning profiles (4 Ad Hoc and 4 App Store) and a P12 certificate. You will email the four Ad Hoc profiles and P12 certificate to support@owncloud.com after building your app with the ownBrander app on customer.owncloud.com. You must create the provisioning profiles and P12 certificate first, before building your app, because you must supply a unique **bundle ID** and an **app group** to build your app. These are created in your account on developer.apple.com, and with Keychain Access on your Mac computer.

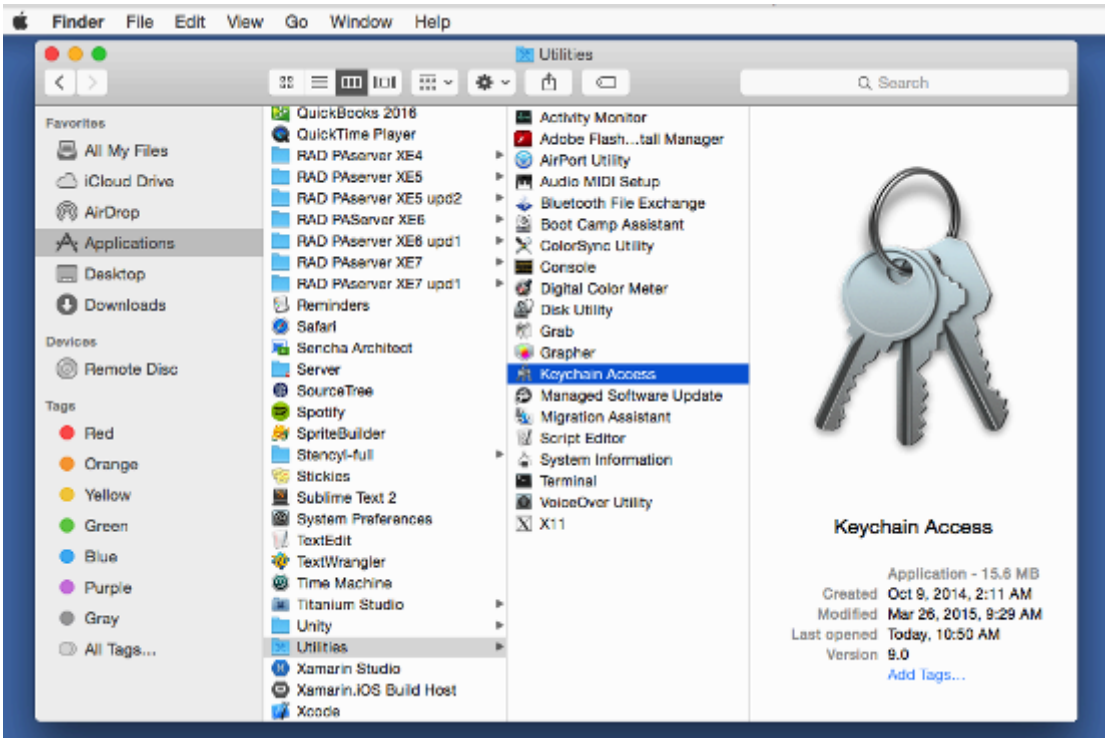
We use the 4 Ad Hoc provisioning profiles and P12 certificate to complete building your app, and then in 24-48 hours your new branded app is loaded into your account on customer.owncloud.com.

The next step is to test your app. When it passes testing, the final step is to upload it to your iTunes Connect account for distribution.

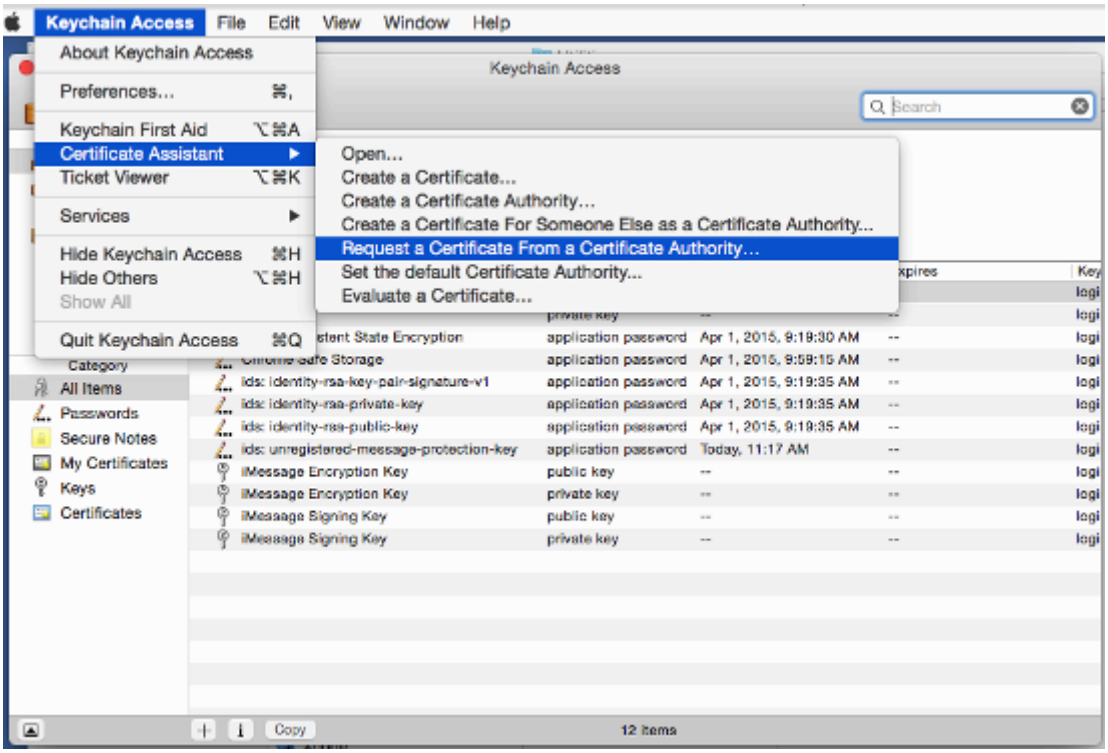
You will need a lot of graphics for building your app, and for your iTunes store listing, in specific sizes and file formats. The ownBrander app and iTunes detail all the image specifications you will need.

Create Certificate Signing Request

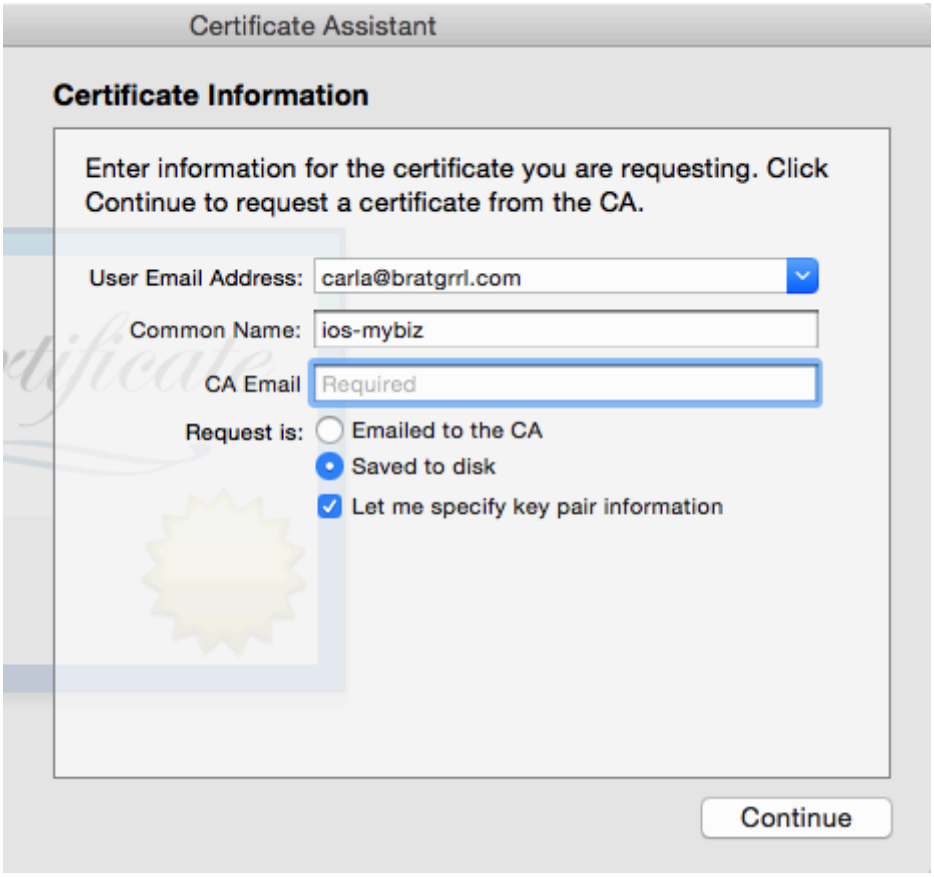
Start by creating a .certSigningRequest (CSR) file on your Mac, using Keychain Access. Open Finder, and then open Keychain Access from the Utilities folder.



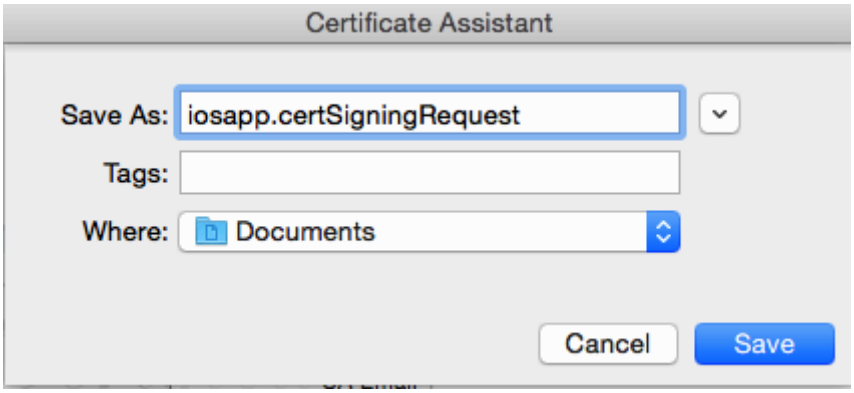
Next, open **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority**.



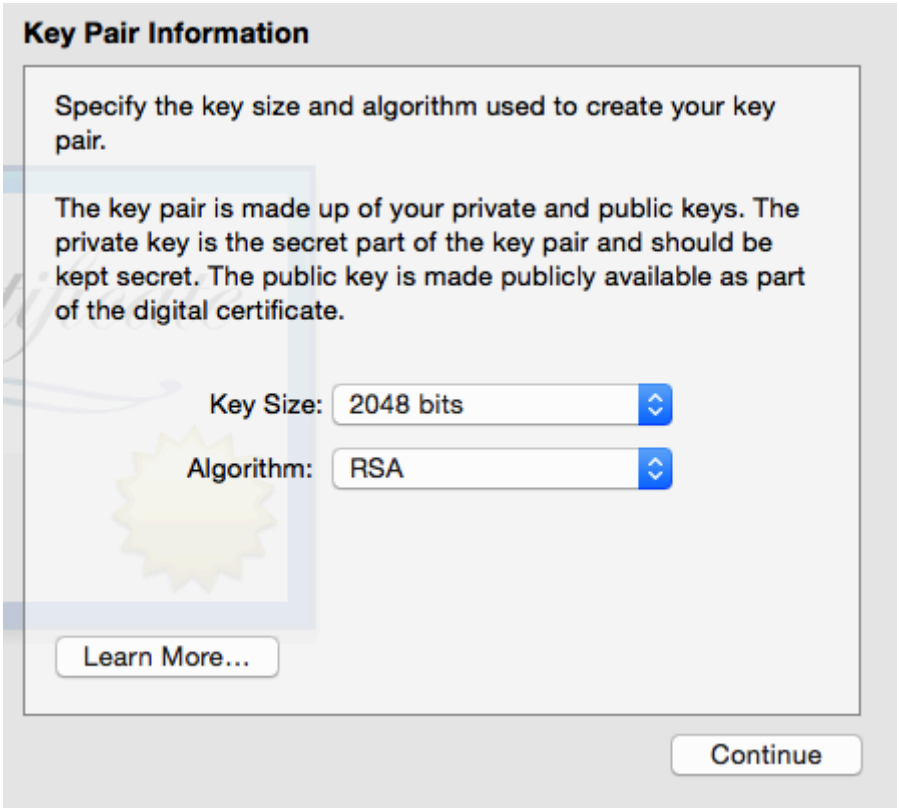
Enter the email address that you use in your Apple developer account, and enter a common name. The common name can be anything you want, for example a helpful descriptive name like "ios-mybiz". Check **[Saved to disk]** and **[Let me specify key pair information]**, then click **[Continue]**.



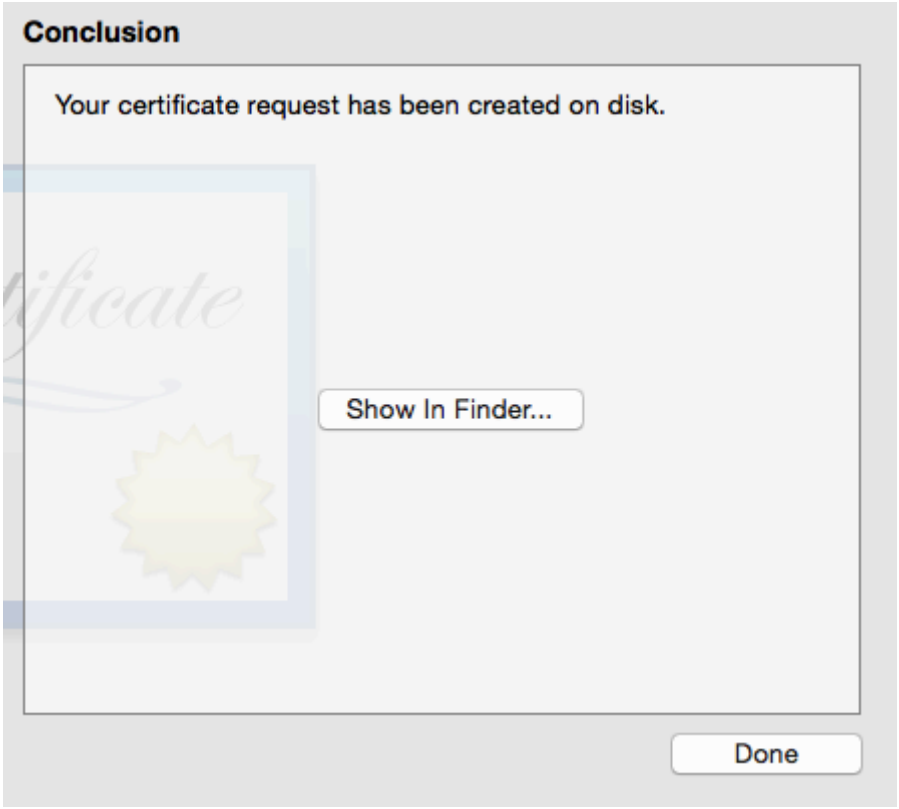
Give your CSR a helpful descriptive name, such as **iosapp.certSigningRequest**, and choose the location to save it on your hard drive, then click **[Save]**.



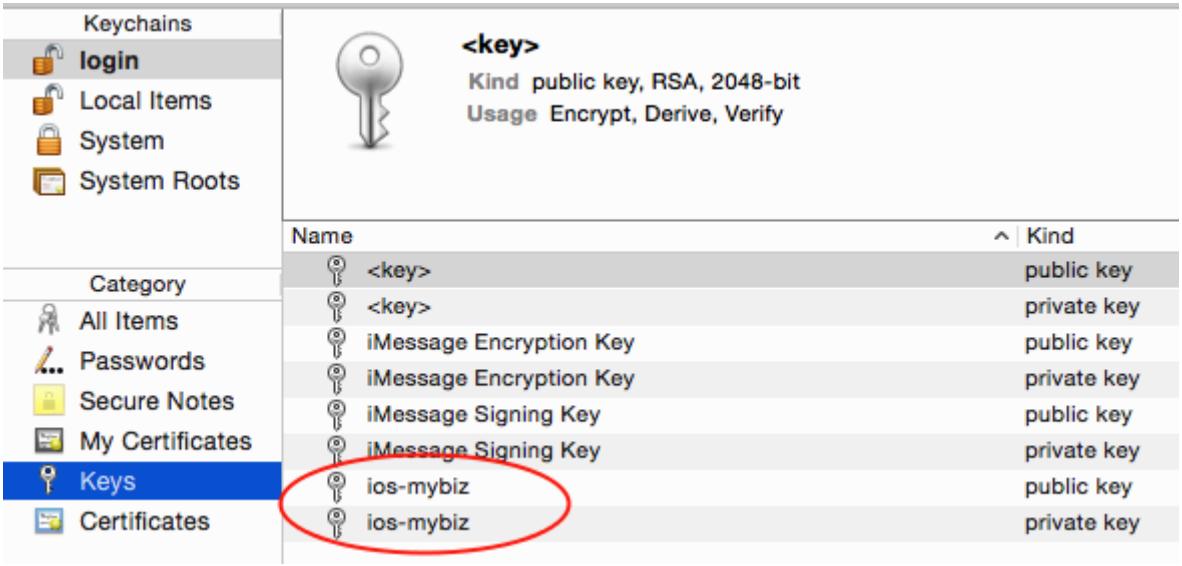
In the next window, set the **Key Size** value to **2048 bits** and **Algorithm** to **RSA**, and click **[Continue]**. This will create and save your certSigningRequest file (CSR) to your hard drive.



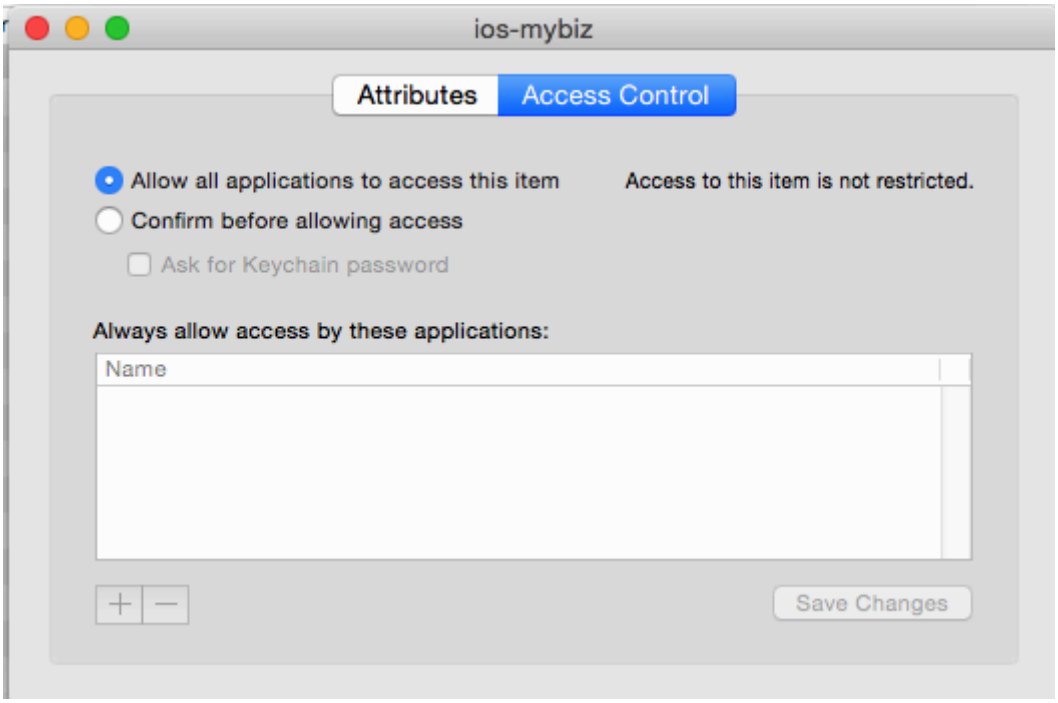
In the next screen your certificate creation is verified. Click a button to view it, or click **[Done]** to go to the next step.



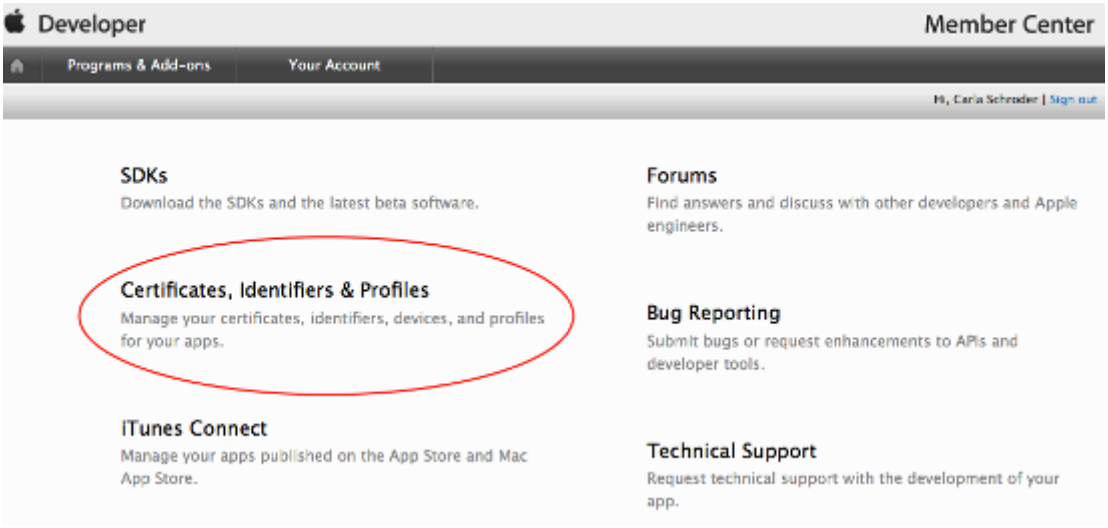
You also get a corresponding public and private key pair, which you can see in the **Login** > **Keys** section of Keychain.



Double-click on your new private key to open the Access Control dialog. Check **[Allow all applications to access this item]**.

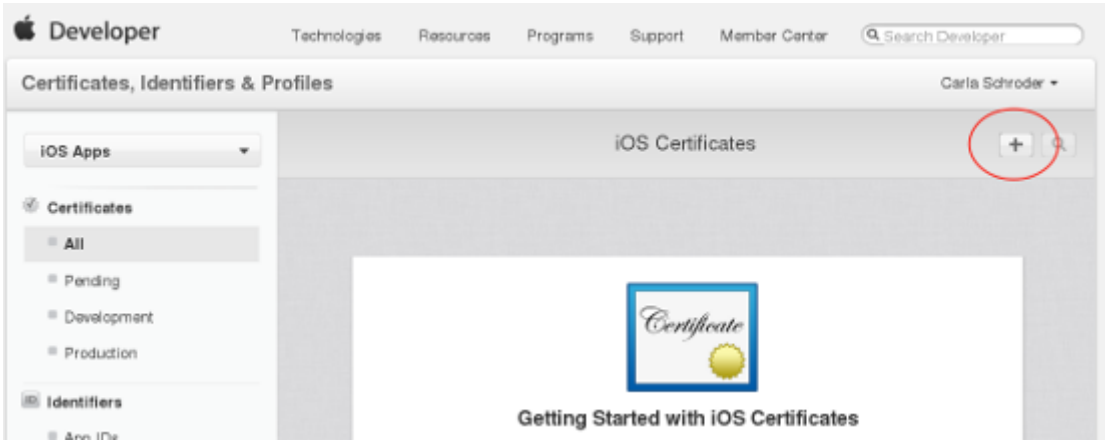


Now login to the **Member Center** on https://developer.apple.com/. Click **[Certificates, Identifiers & Profiles]**.

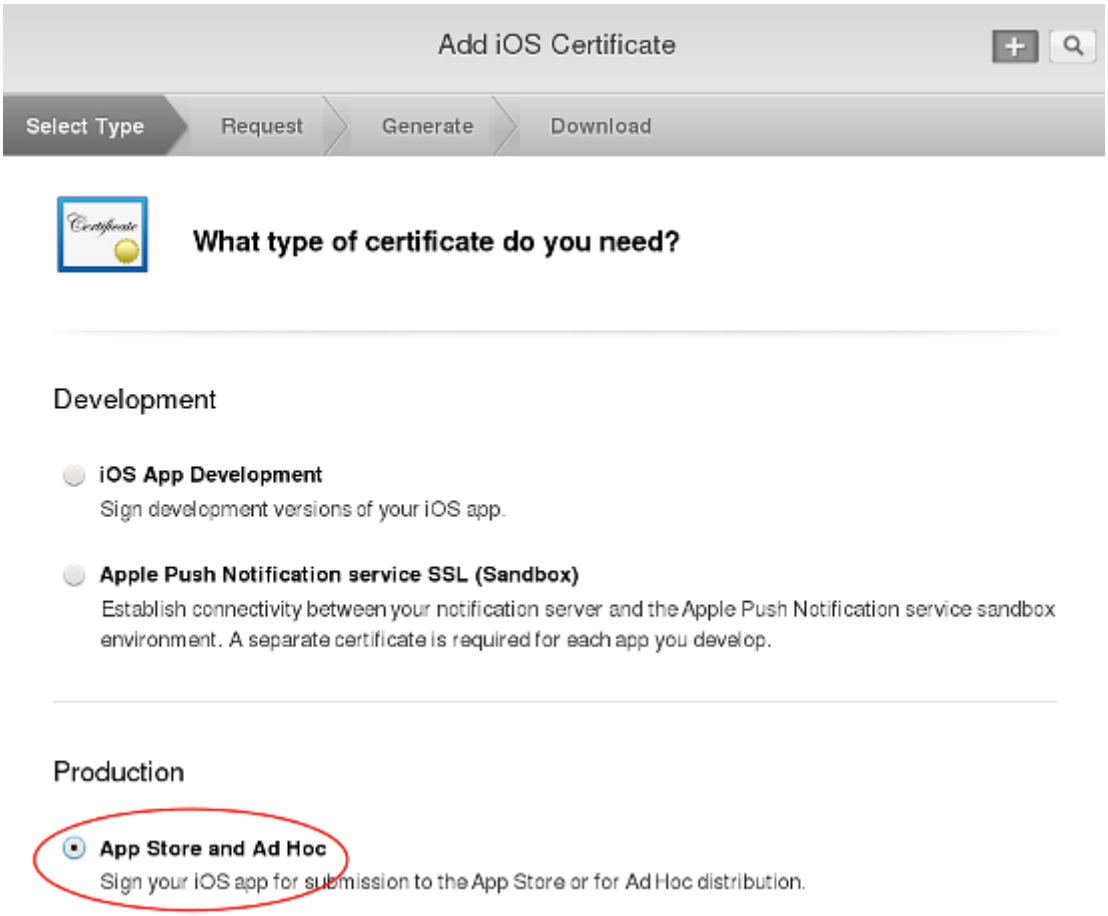


Then click **iOS Apps** > **Certificates**.

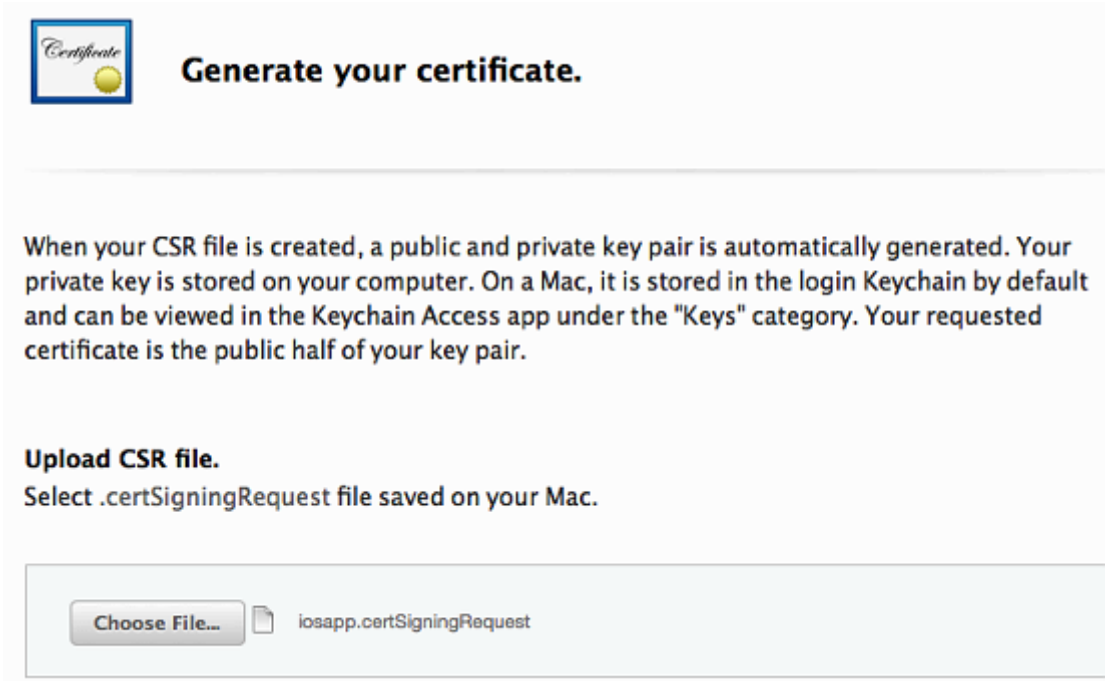
Next, click the **[add]** button (the little plus sign) in the top right corner of the **iOS Certificate** page.



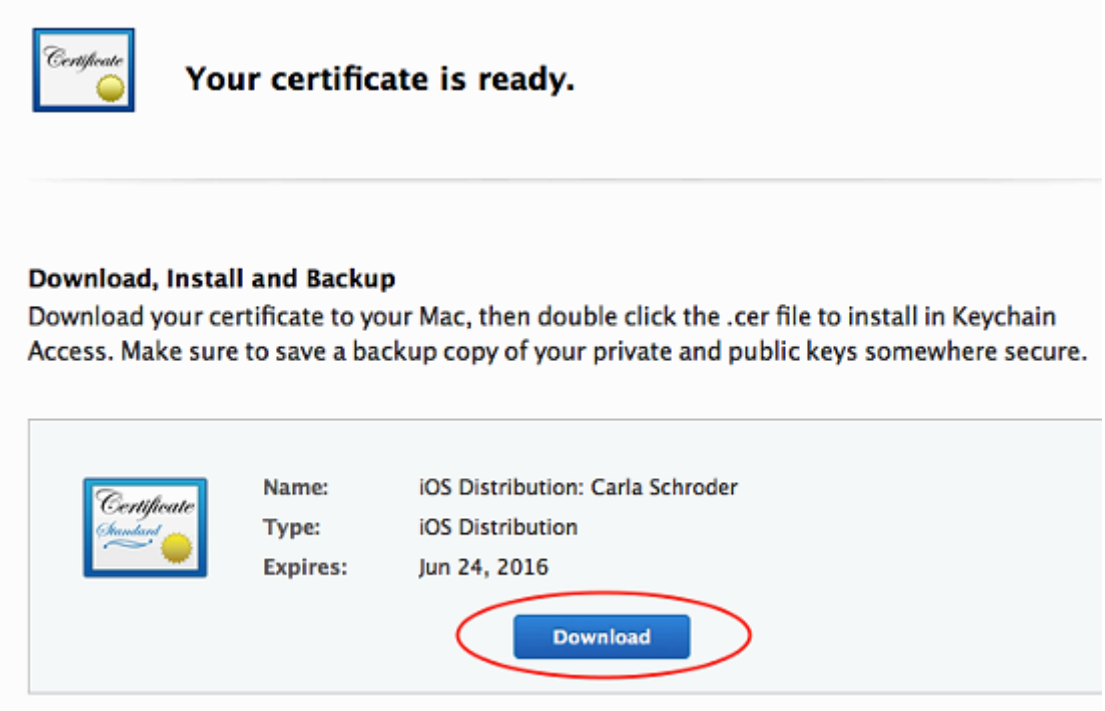
Under "What type of certificate do you need?" check **[App Store and Ad Hoc]**, then click the **[Continue]** button at the bottom of the page.



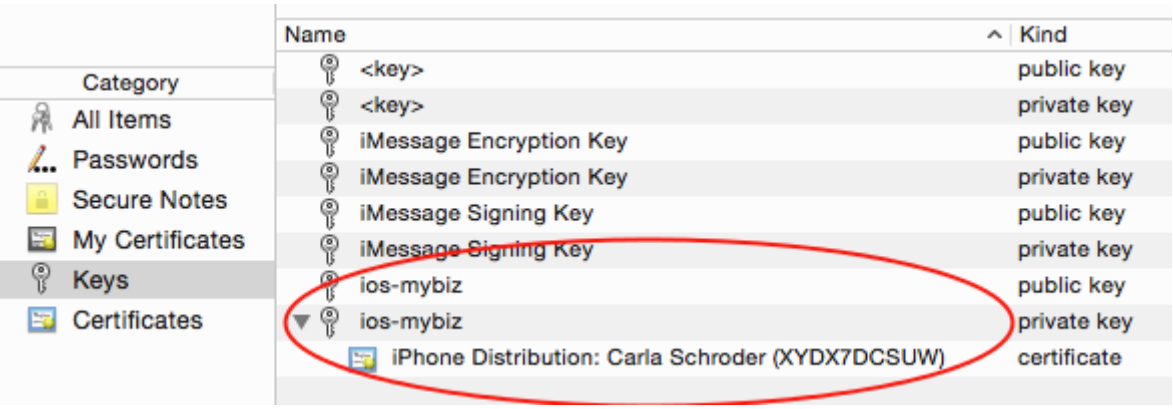
The next screen, **About Creating a Certificate Signing Request (CSR)** has information about creating a CSR in Keychain Access. You already did this, so go to the next screen. "Add iOS Certificate", to upload the CSR you already created, then click the **[Generate]** button.



Your new certificate is named **ios_distribution.cer**. Download it to your Mac; then find it and double-click on it to install it properly in Keychain.



After installing it, you should see it stored with its corresponding private key in Keychain.



Remember to make backups of your keys and certificates and keep them in a safe place.

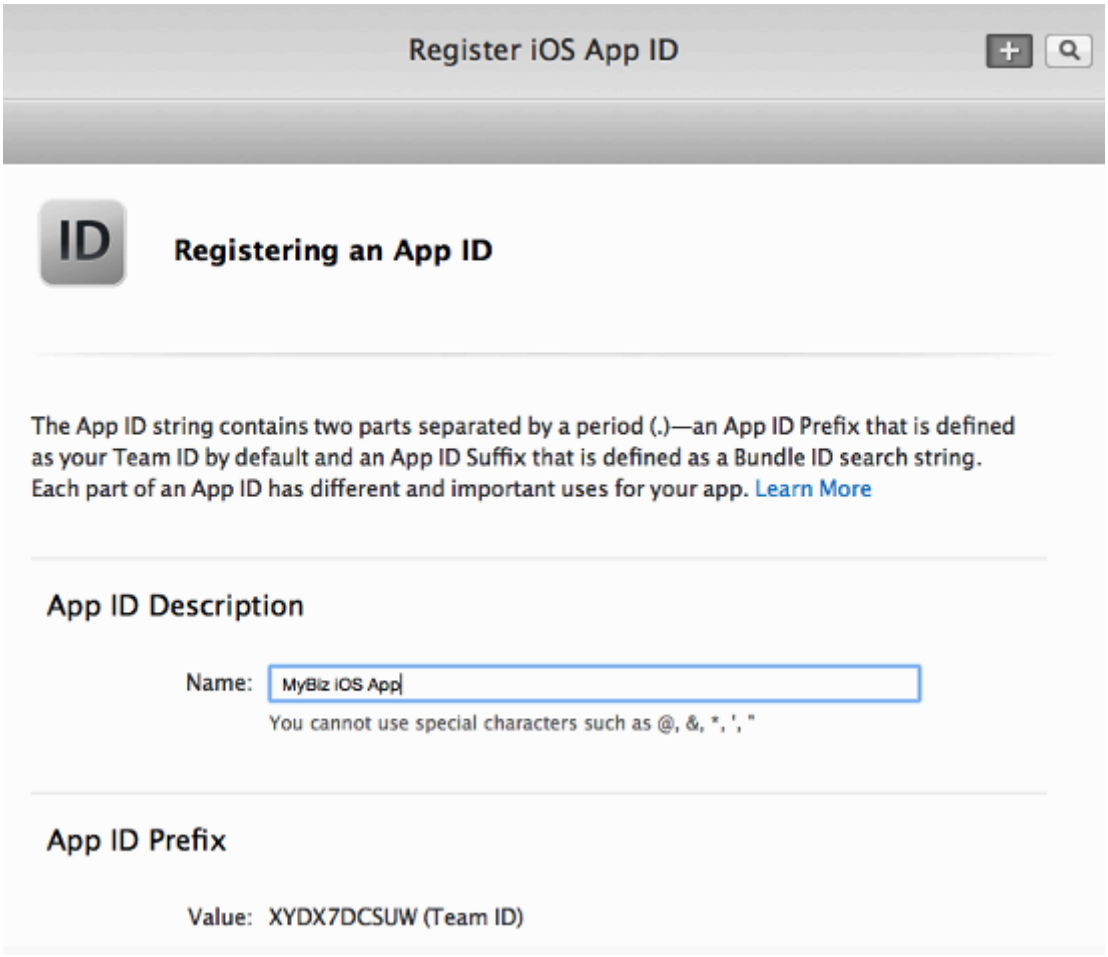
Create Bundle IDs

Create Bundle IDs

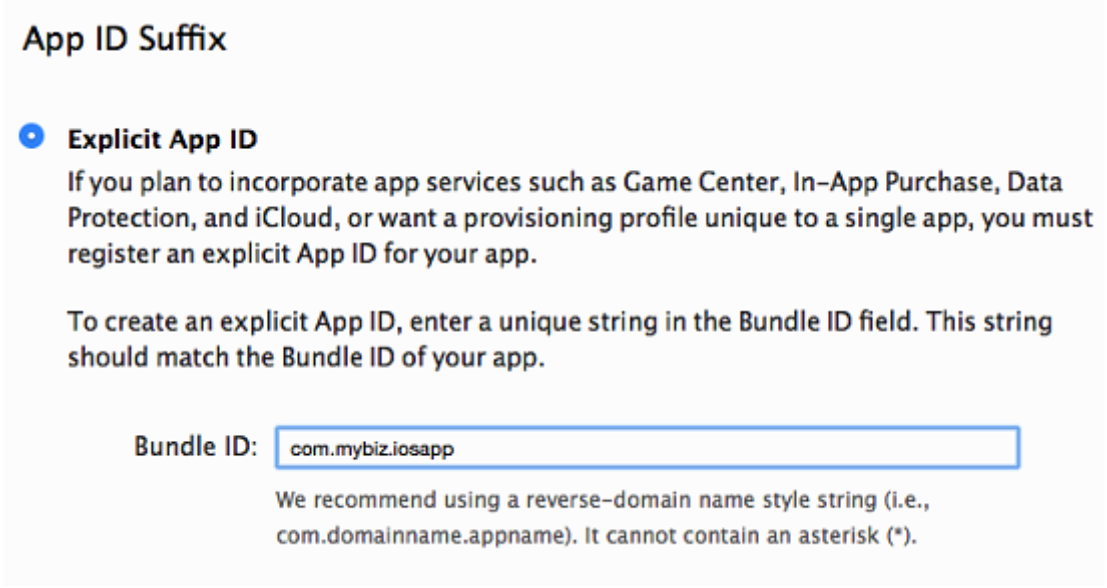
The next step is to create four **Bundle IDs**. These are unique identifiers for your branded iOS app. You must also create an **App Group** and place your three **Bundle IDs** in your **App Group**. You will need your base **Bundle ID** and **App Group** when you build your app with the ownBrander app on customer.owncloud.com.

Create App ID

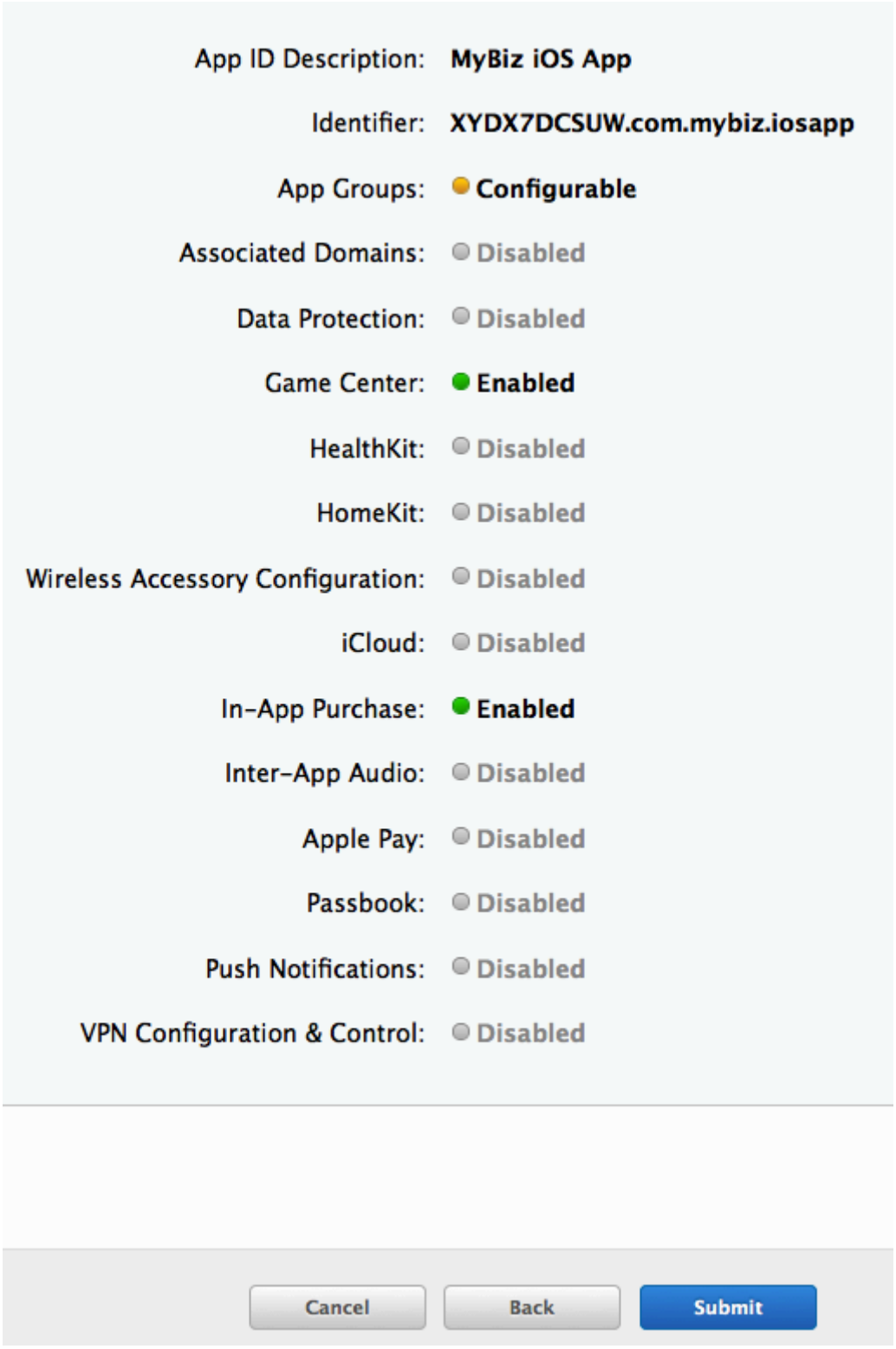
Now you must create your App ID. Go to **Identifiers** > **App IDs** and click the **[plus button]** (top right) to open the "Register iOS App ID" screen. Fill in your **App ID Description**, which is anything you want, so make it helpful and descriptive. The **App ID Prefix** is your Apple Developer Team ID, and is automatically entered for you.



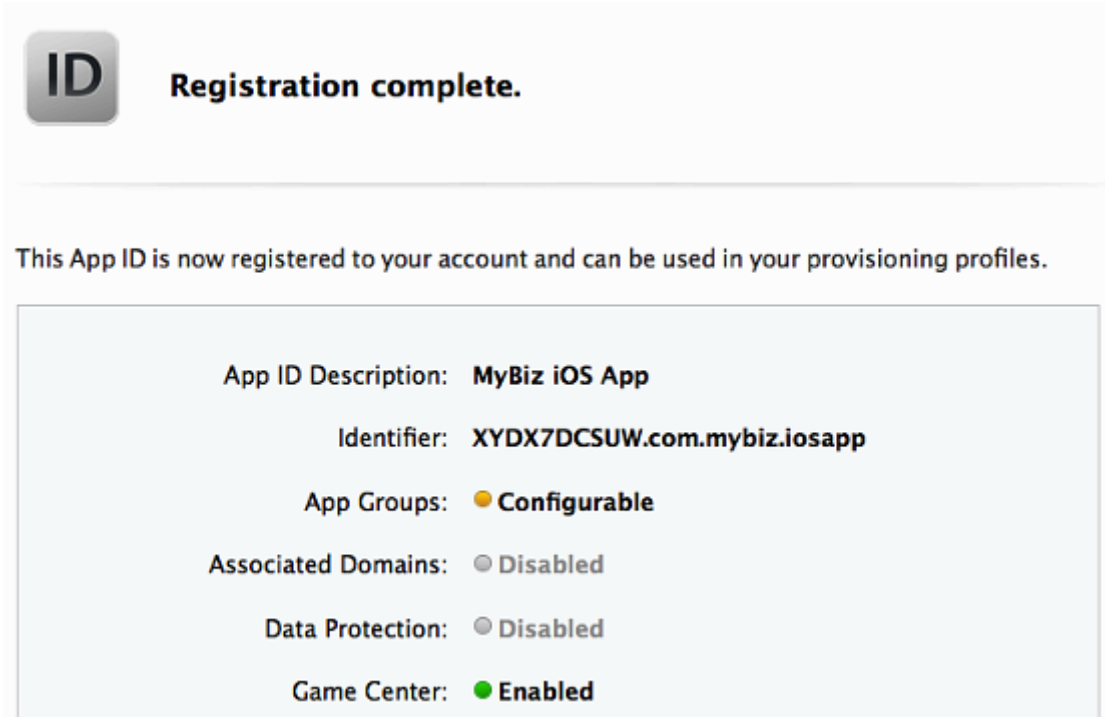
Scroll down to the **App ID Suffix** section and create your **Bundle ID**. Your **Bundle ID** is the unique identifier for your app. Make a note of it because you will need it as you continue through this process. The format for your **Bundle ID** is reverse-domain, e.g. *com.MyCompany.MyProductName*.



The next section, **App Services**, is where you select the services you want enabled in your app. You can edit this anytime after you finish creating your **App ID**. Check **App Groups**, make your other selections and then click the **[Continue]** button at the bottom. Now you can confirm all of your information. If everything is correct click **[Submit]**; if you need to make changes use the **[Back]** button.

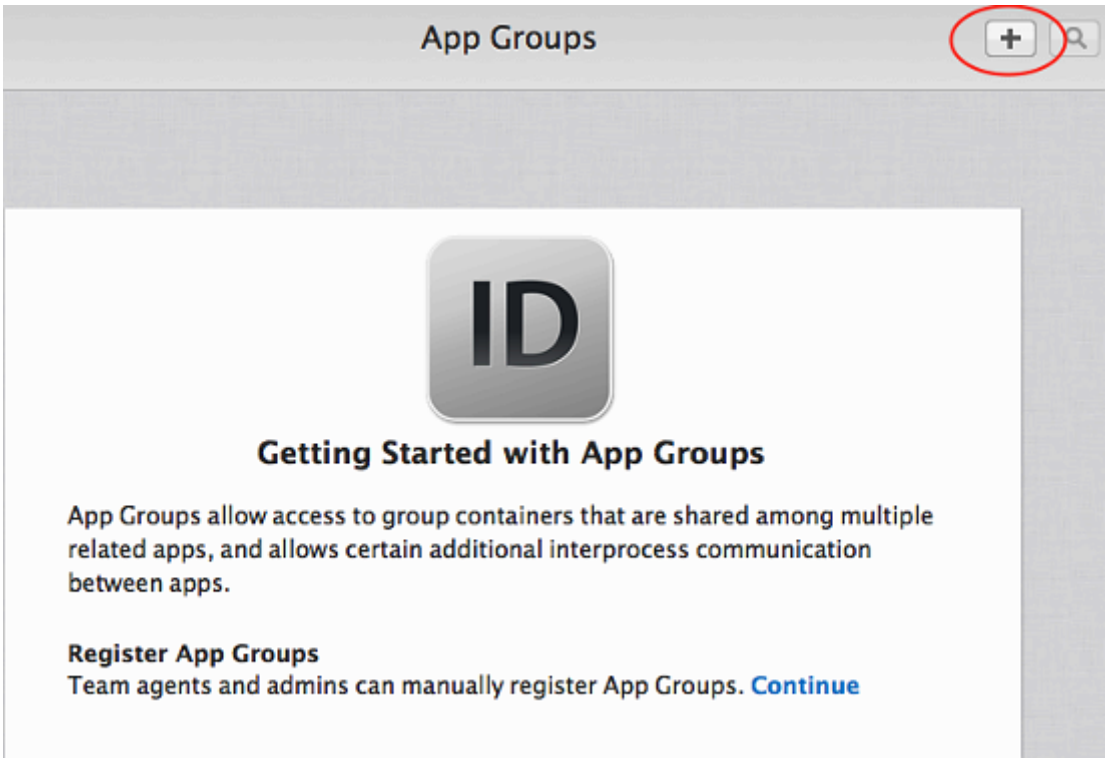


When you are finished you will see a confirmation. Click the **[Done]** button at the bottom.

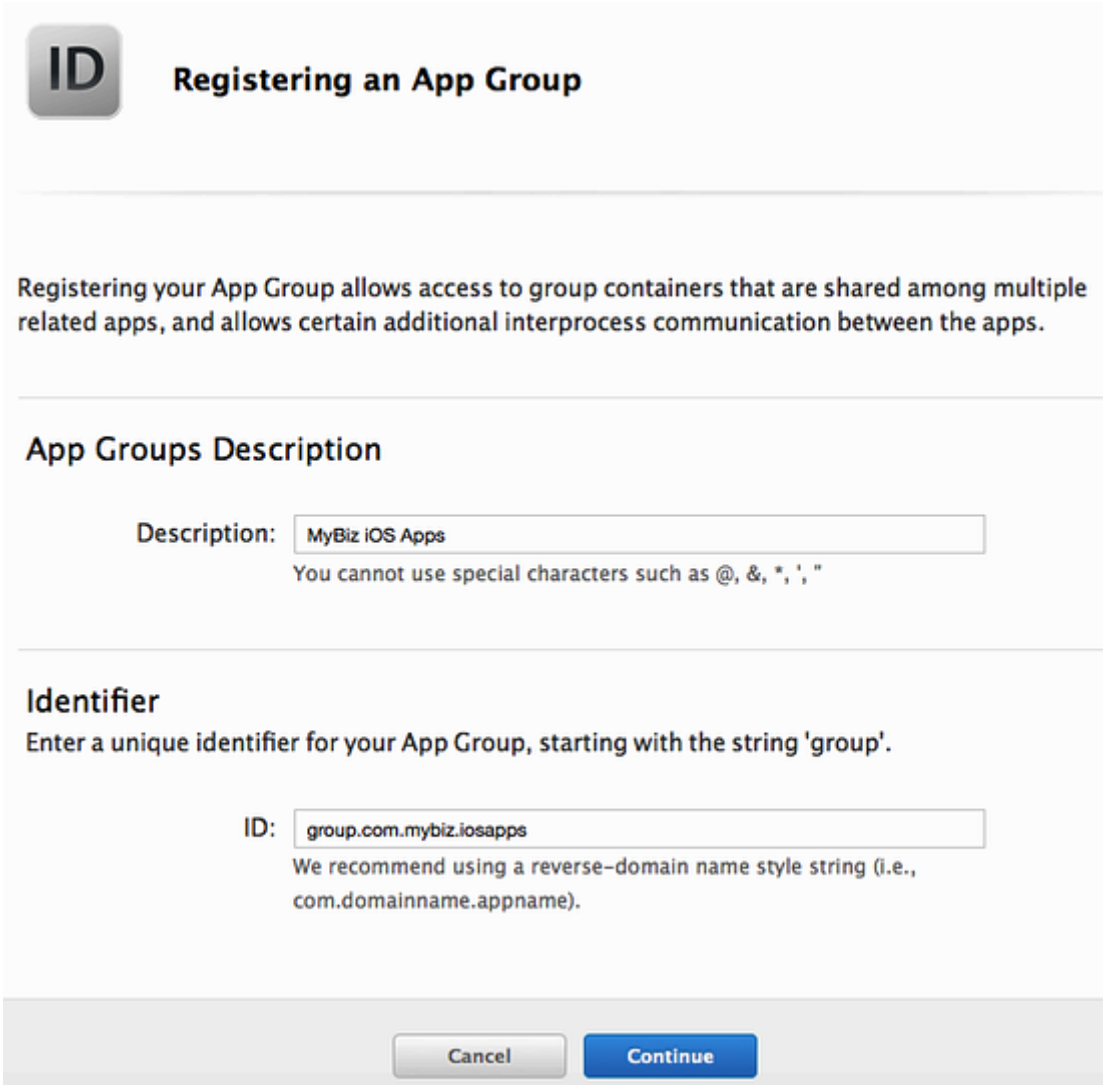


Create App Group

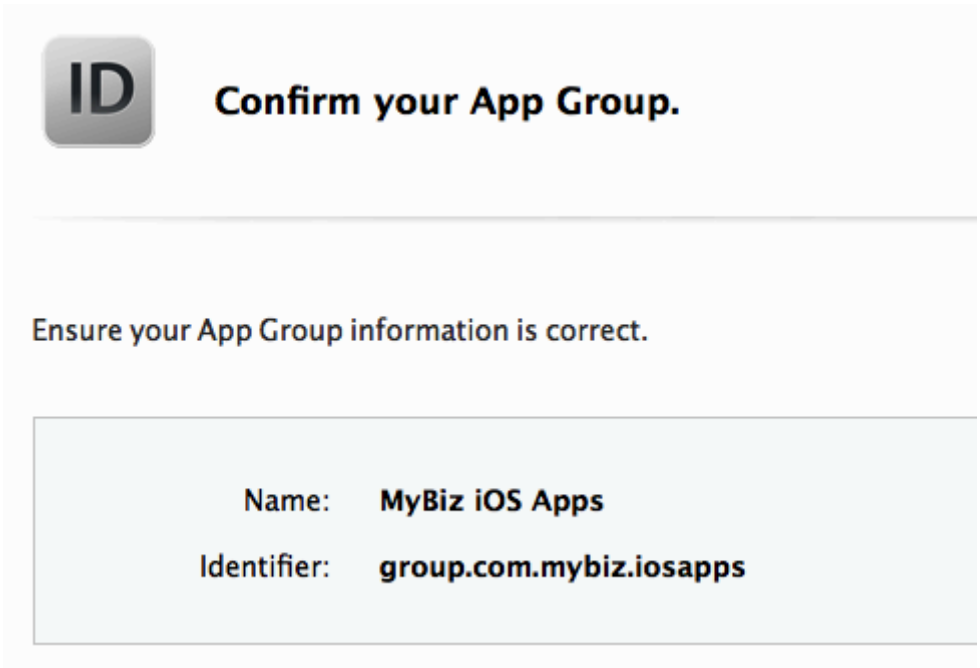
The next step is to create an App Group and put your App ID in it. Go to **Identifiers** > **App Groups** and click the **[plus button]** (top right).



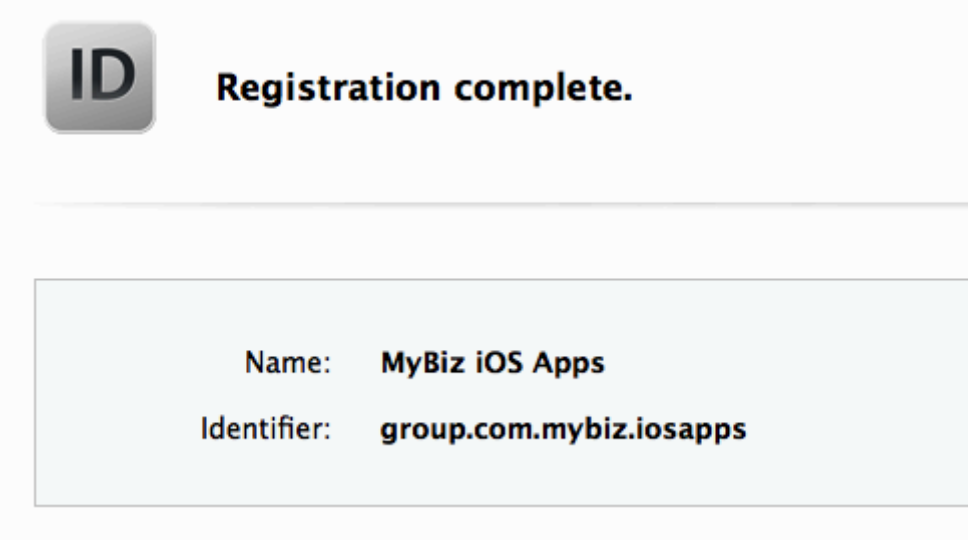
Create a description for your app group, and a unique identifier in the format *group.com.MyCompany.MyAppGroup*. Then click **[Continue]**



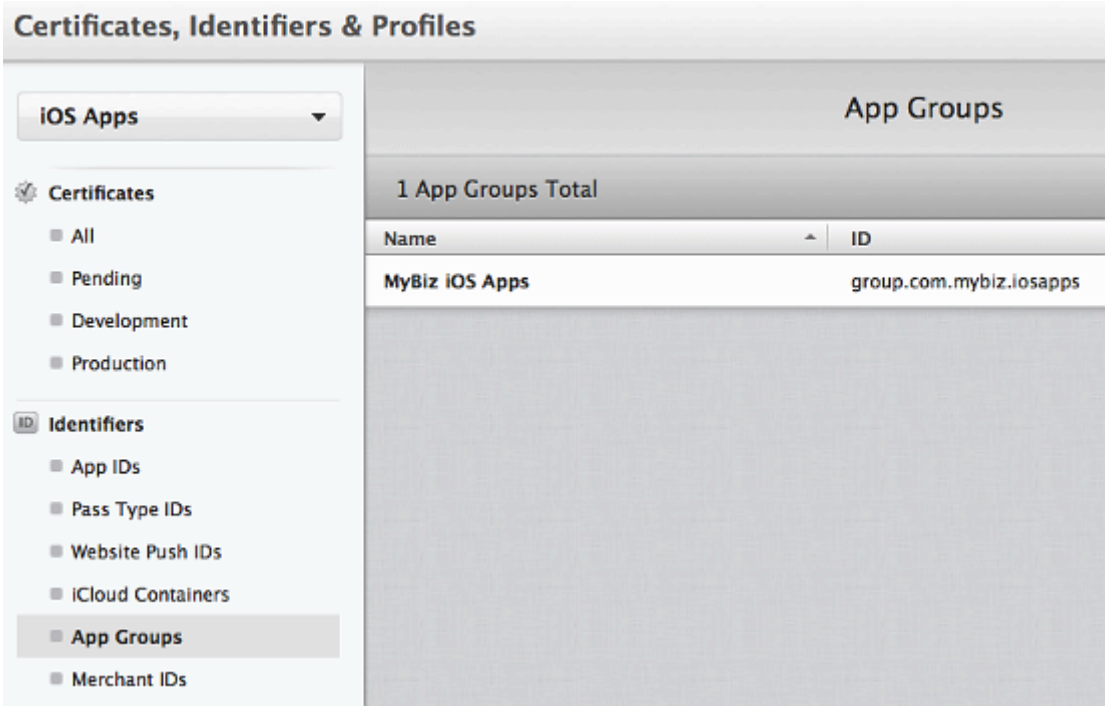
Review the confirmation screen, and if everything looks correct click the **[Register]** button.



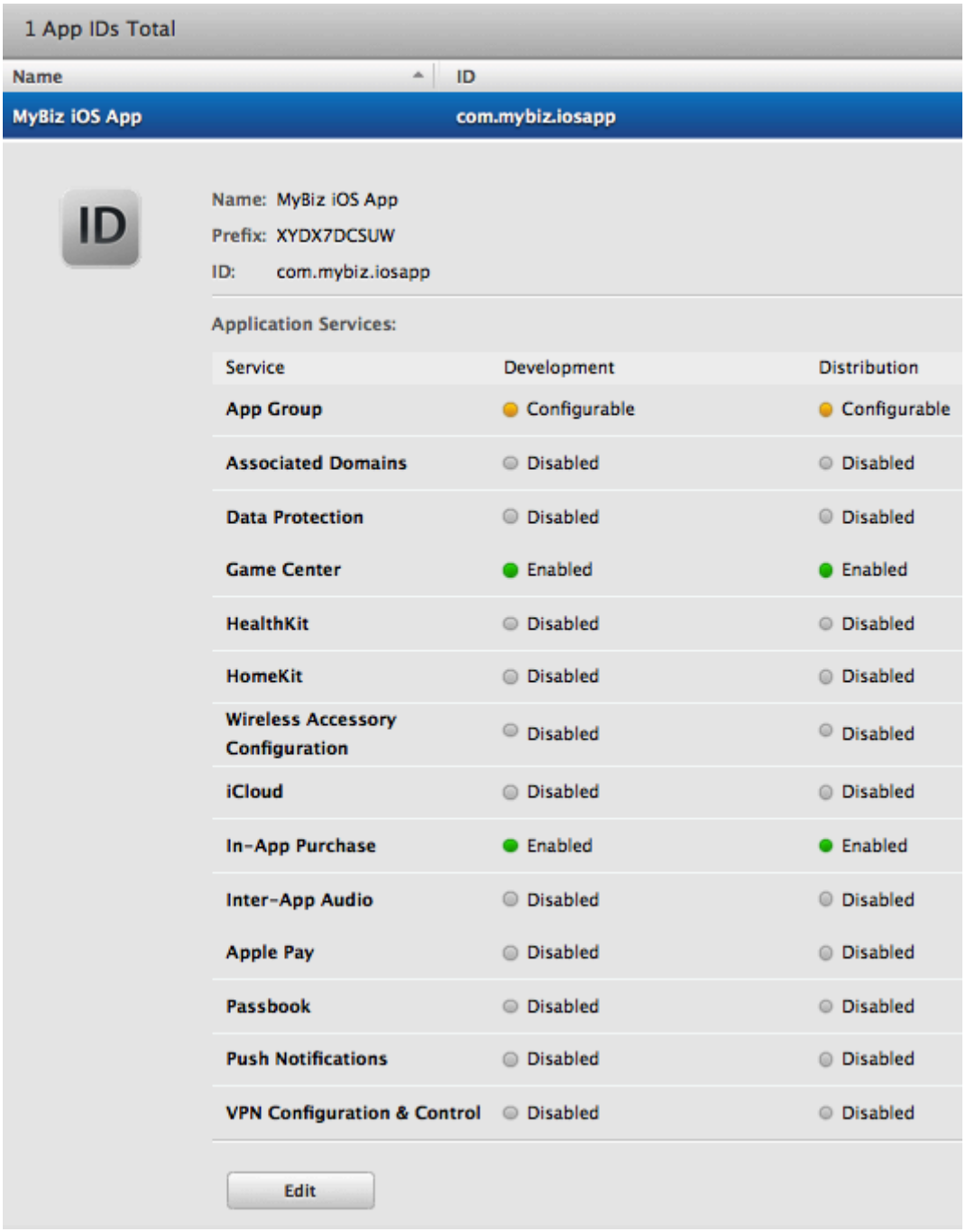
You'll see a final confirmation screen; click **[Done]**.



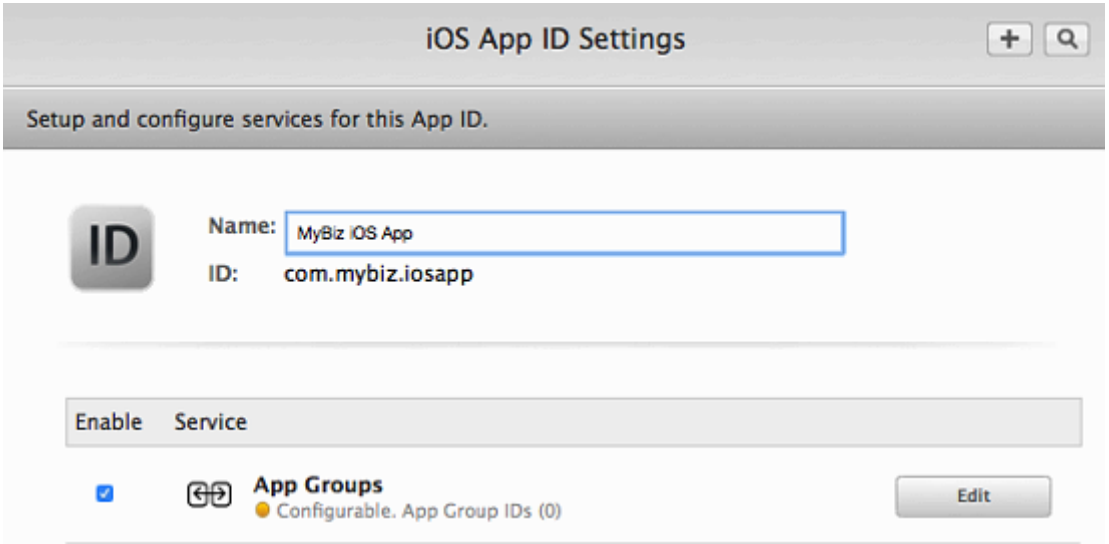
When you click on **[App Groups]** you will see your new app group.



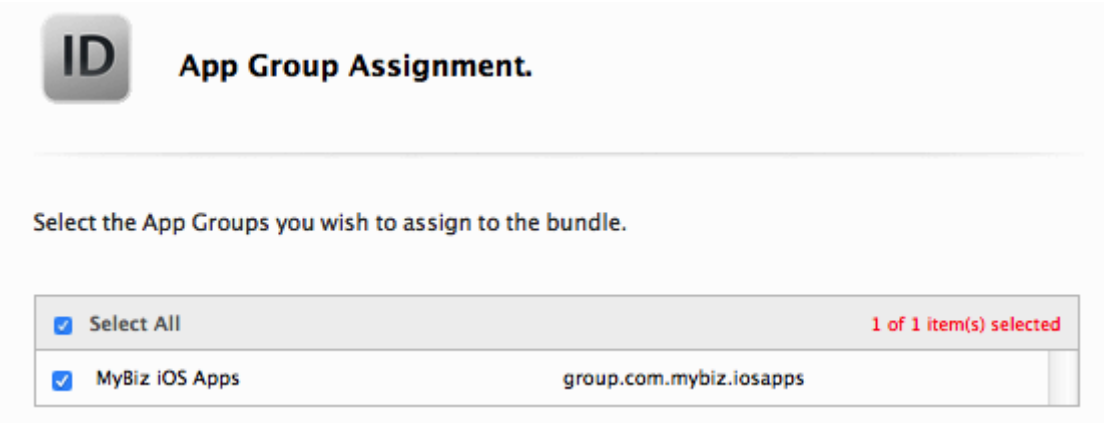
Now go back to **Identifiers** > **App IDs** and click on your **[App ID]**. This opens a screen that displays all your app information. Click the **[Edit]** button at the bottom.



Click the **[Edit]** button next to **[App Groups]**.



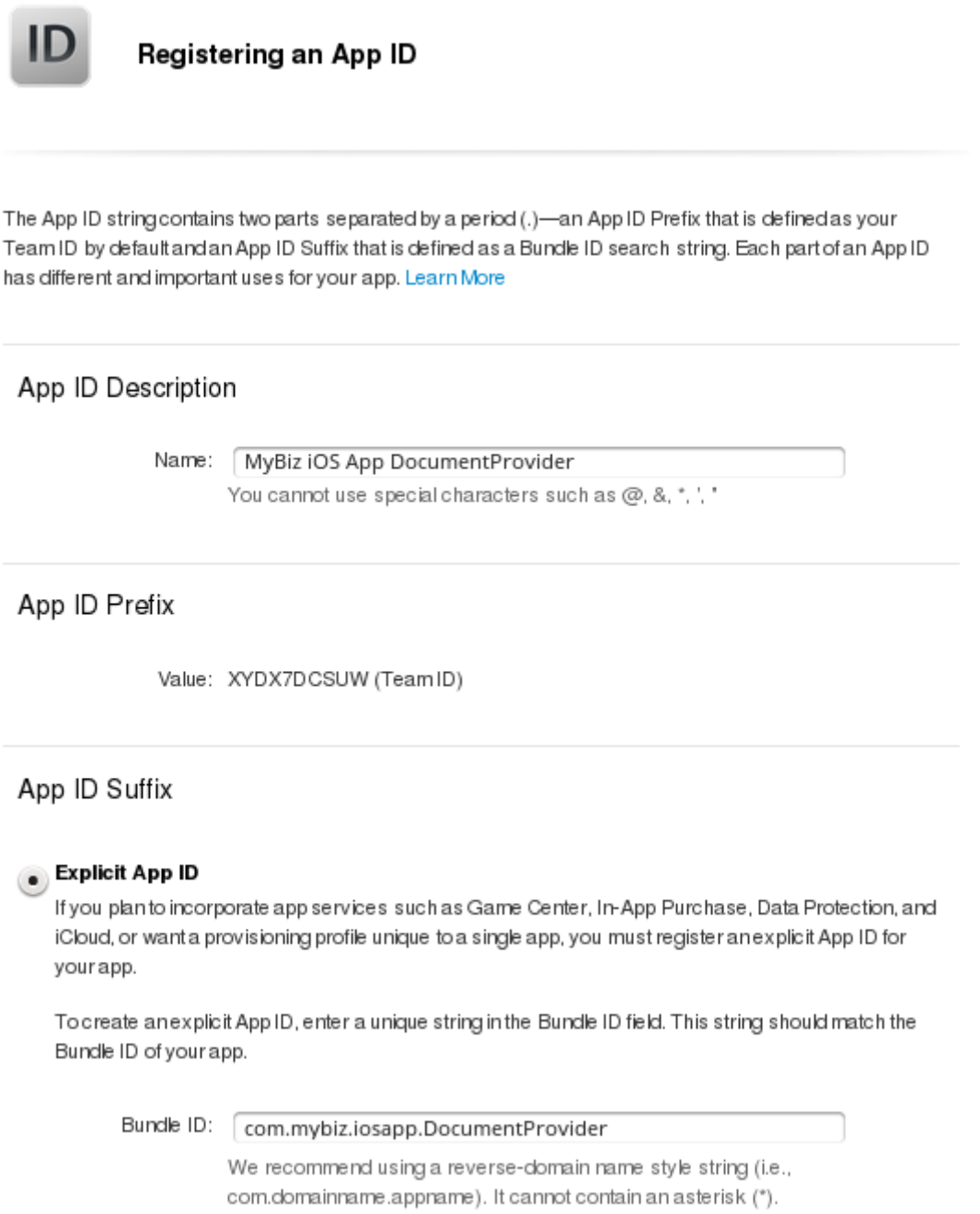
Check your app and click the **[Continue]** button.



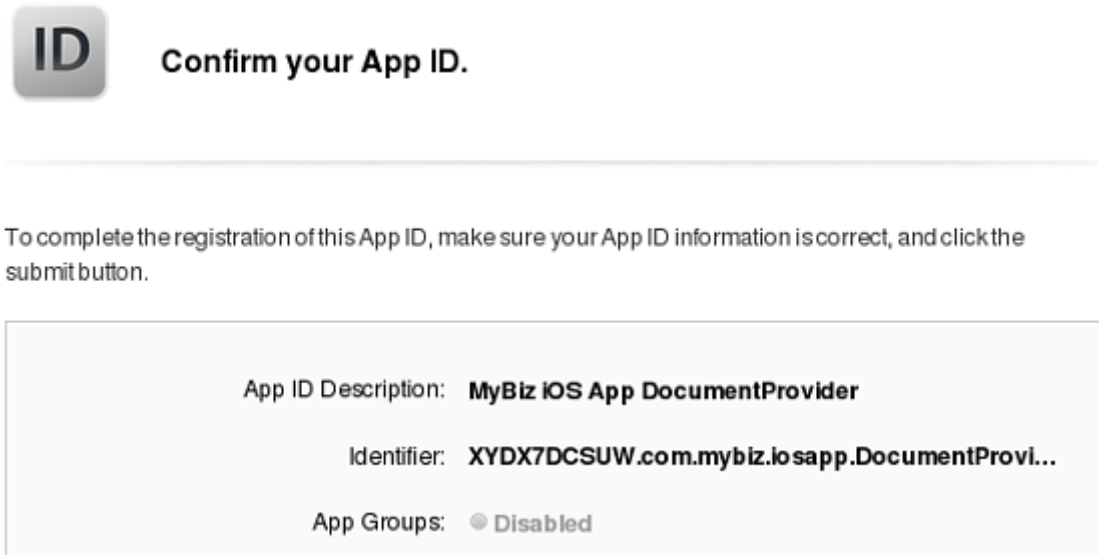
The next screen asks you to "Review and confirm the App Groups you have selected". Click the **[Assign]** button to confirm. The next screen announces "You have successfully updated the App Groups associations with your App ID", and you must click yet another button, the **[Done]** button at the bottom.

Create a DocumentProvider Bundle ID

Now you must return to **Identifiers** > **App IDs** and click the **[plus button]** to create a DocumentProvider Bundle ID. Follow the same naming conventions as for your App ID, then click **[Continue]**.

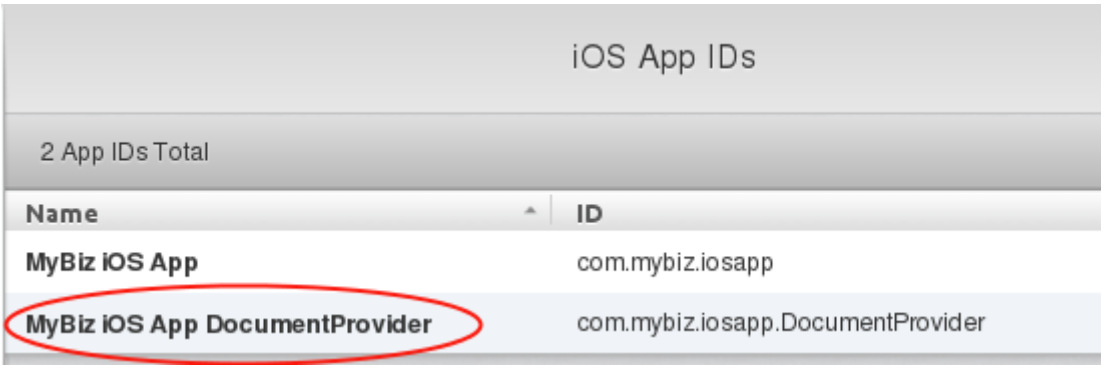


Confirm your new App ID and click **[Submit]**.

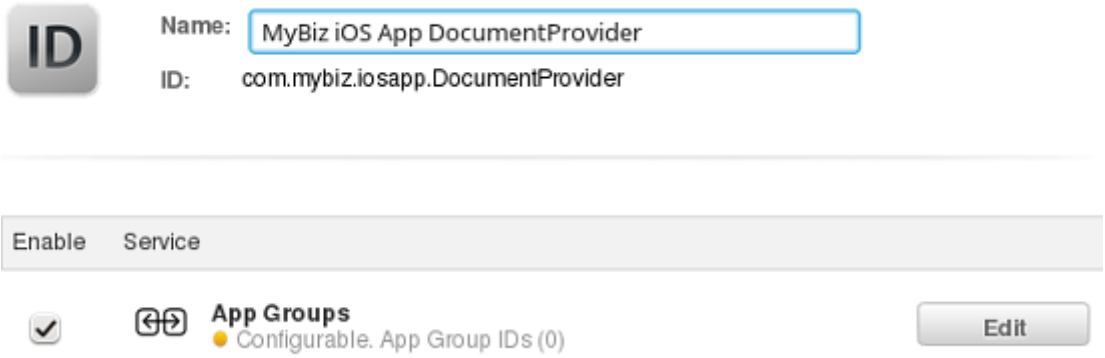


You will see one more confirmation: "Registration complete. This App ID is now registered to your account and can be used in your provisioning profiles." Click **[Done]**.

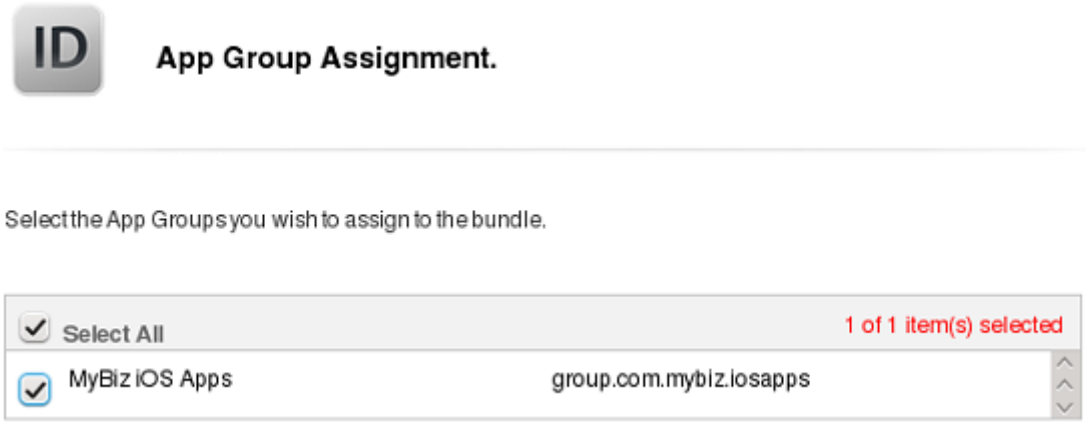
Now you need to add it to your App Group. Go to **Identifiers** > **App IDs** and click on your new **[DocumentProvider Bundle ID]** to open its configuration window, and then click the **[Edit]** button at the bottom.



Select **[App Groups]** and click the **[Edit button]**.



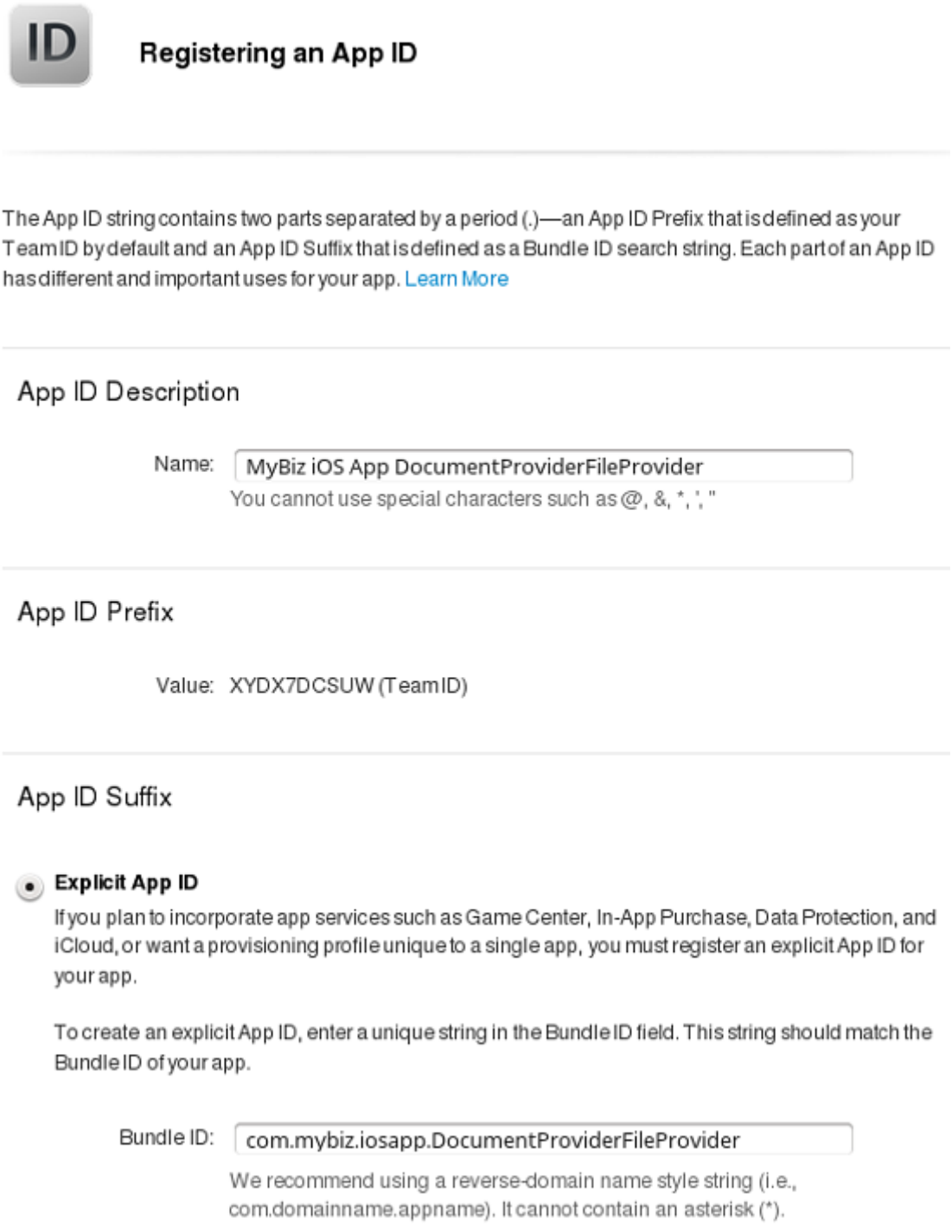
Select your group and click **[Continue]**.



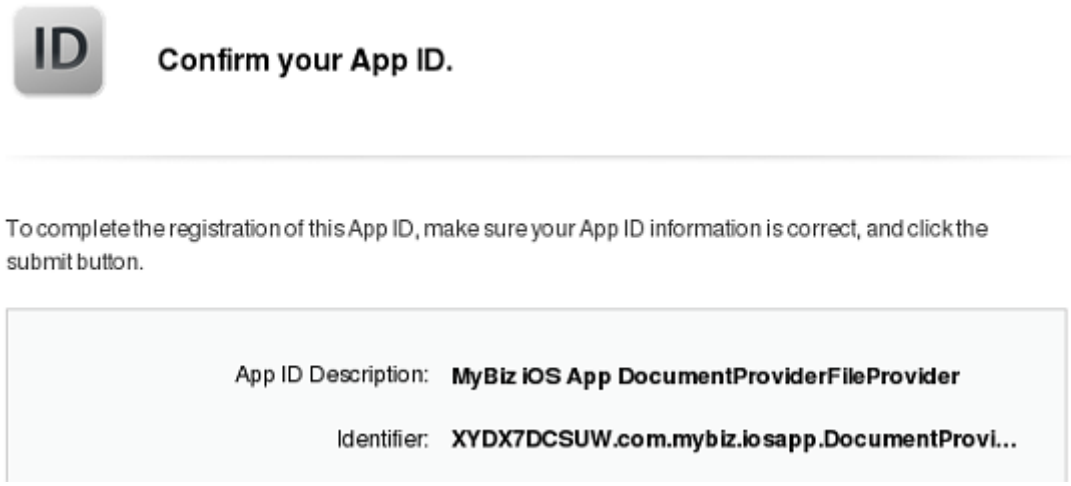
Once again you will asked if you really mean it. On the confirmation screen click **[Assign]**, and you'll see the message "You have successfully updated the App Groups associations with your App ID."

Create a DocumentProviderFileProvider Bundle ID

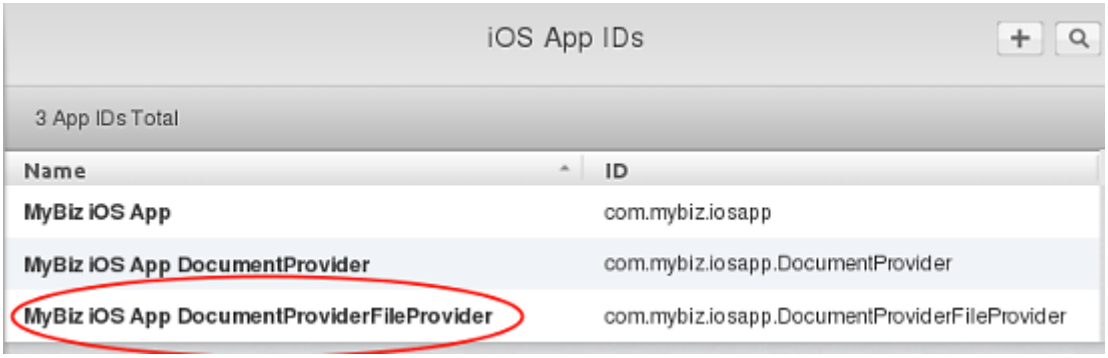
One more time, go to **Identifiers** > **App IDs** and click the **[plus button]** to create a DocumentProviderFileProvider Bundle ID. Follow the same naming conventions as for your App ID, then click **[Continue]**.



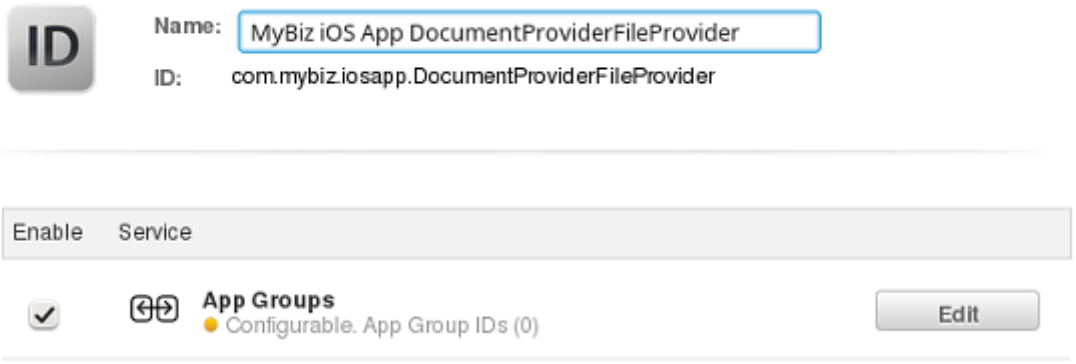
Confirm your new App ID and click **[Submit]**.



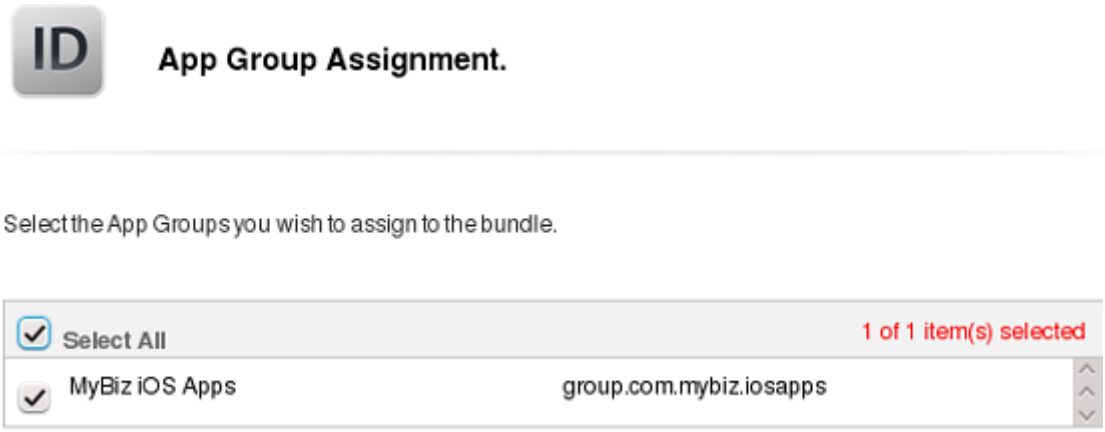
You will see one more confirmation; review it and click **[Done]**. Now you need to add it to your App Group. Go to **Identifiers** > **App IDs** and click on your new **[DocumentProviderFileProvider Bundle ID]** to open its configuration window, and then click the **[Edit]** button.



Select **[App Groups]** and click the **[Edit]** button.



Select your group and click **[Continue]**.

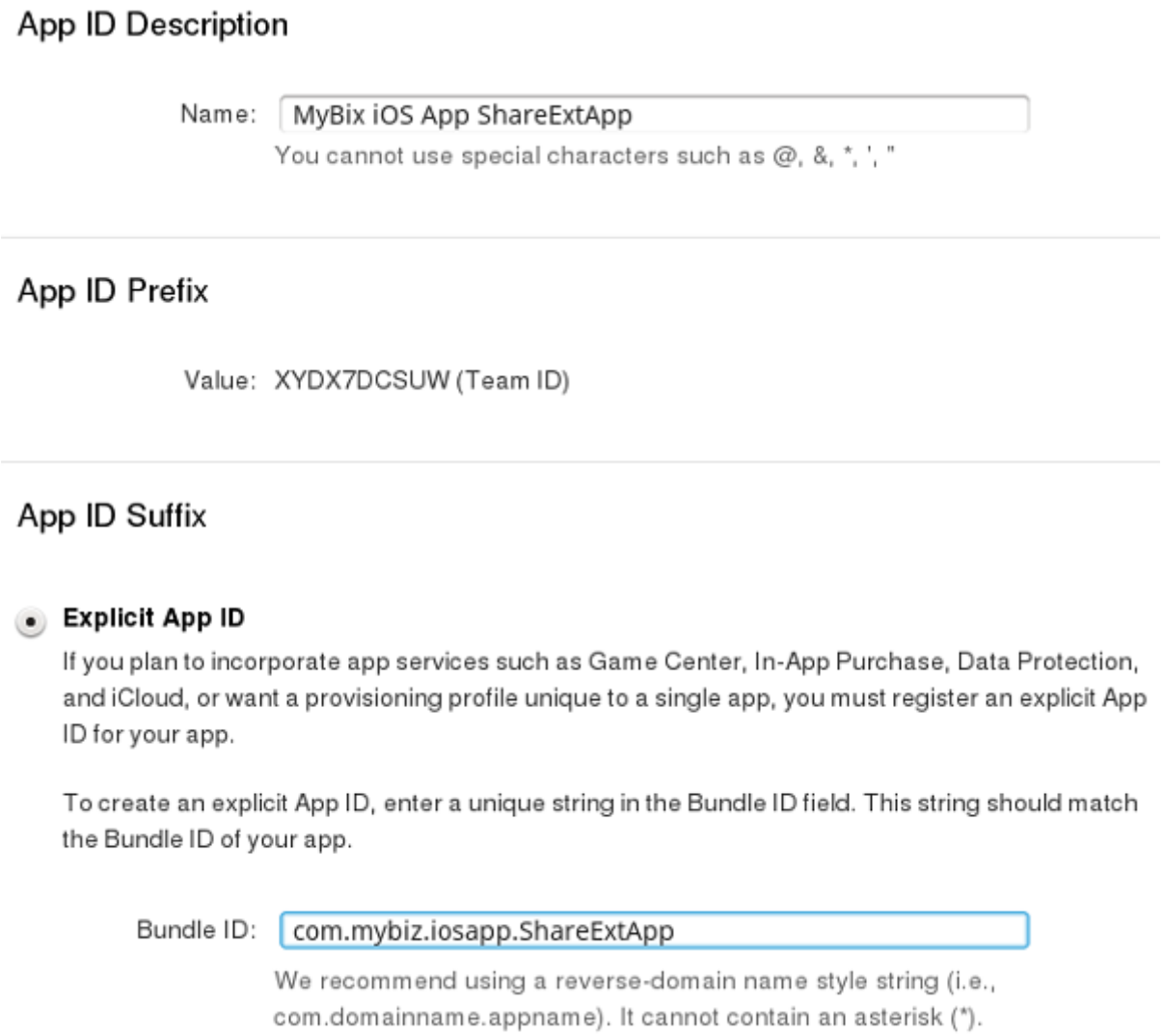


On the confirmation screen click **[Assign]**, and you'll see the message "You have successfully updated the App Groups associations with your App ID."

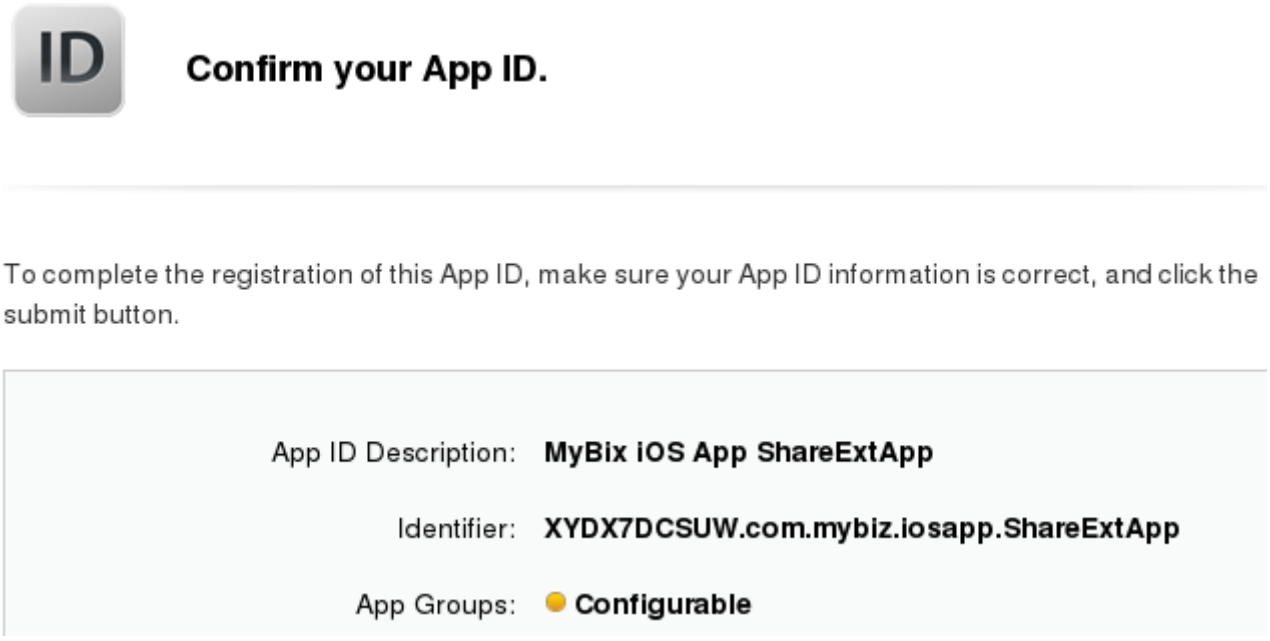
Create a ShareExtApp Bundle ID

This supports Apple's ShareIN extension.

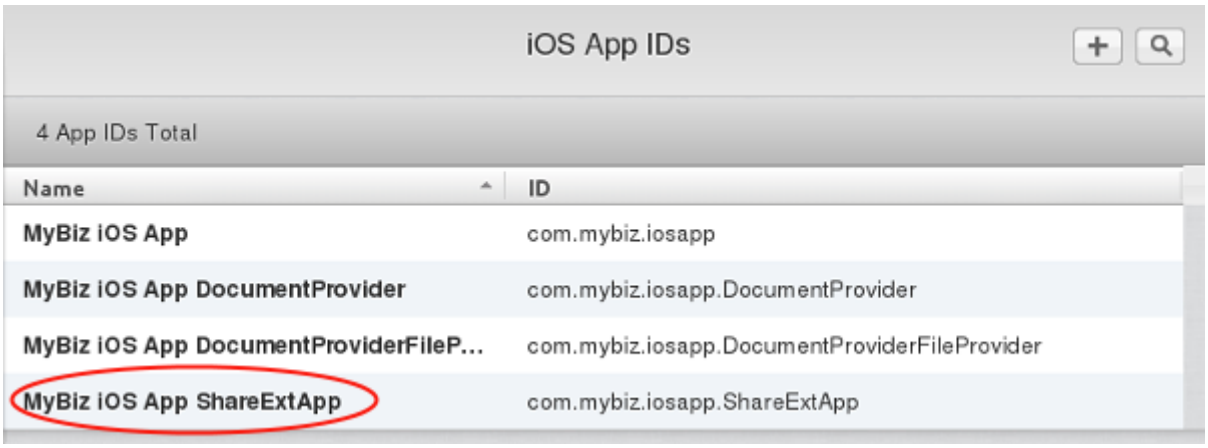
Yet again, go to **Identifiers** > **App IDs** and click the **[plus button]** to create a ShareExtApp Bundle ID. Follow the same naming conventions as for your App ID, then click **[Continue]**.



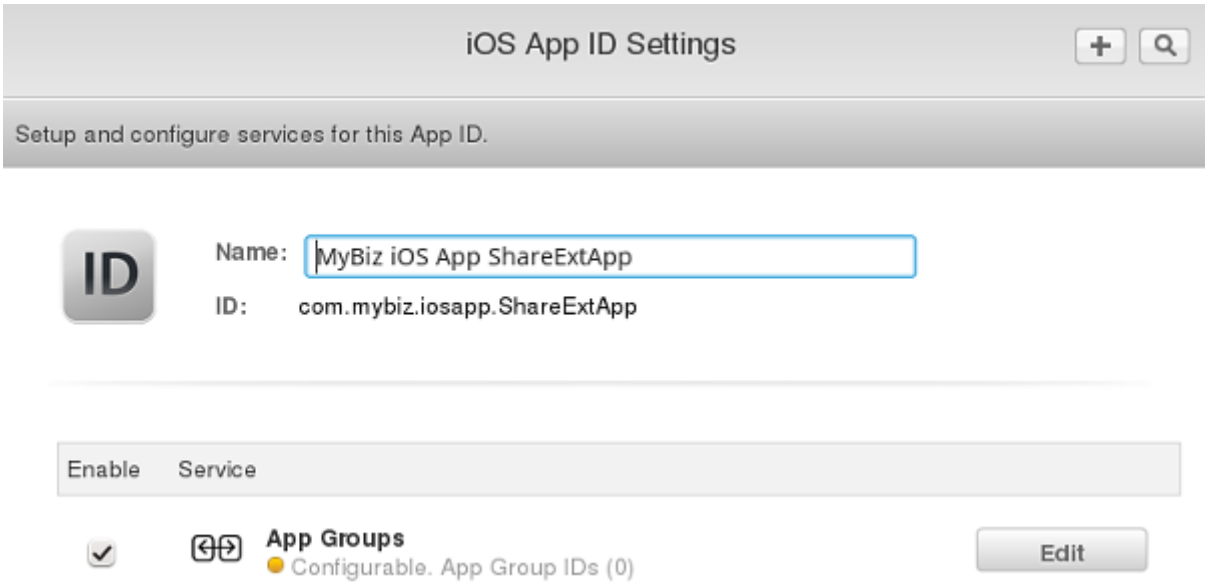
Confirm your new App ID and click **[Submit]**.



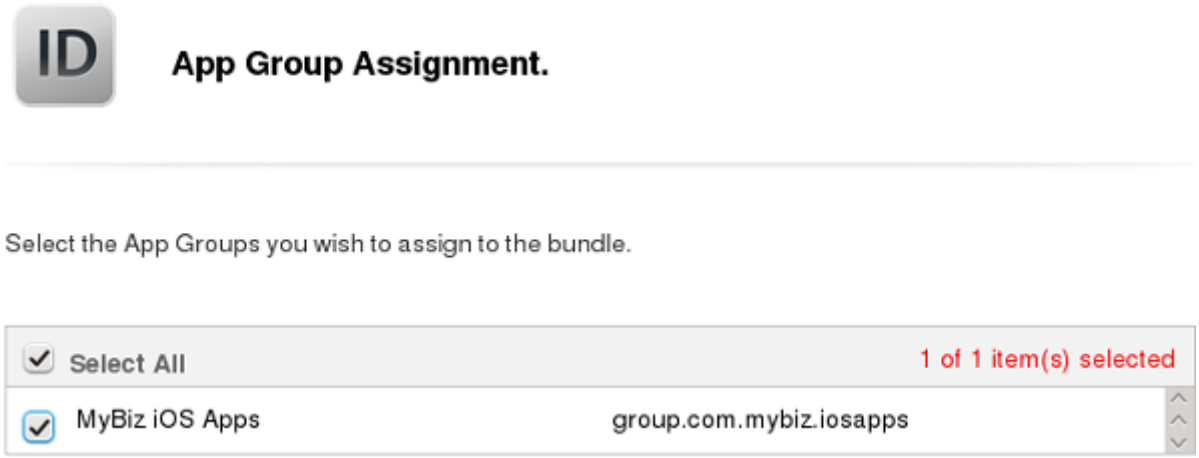
You will see one more confirmation; review it and click **[Done]**. Now you need to add it to your App Group. Go to **Identifiers** > **App IDs** and click on your new **[ShareExtApp Bundle ID]** to open its configuration window, and then click the **[Edit]** button.



Select **[App Groups]** and click the **[Edit]** button.



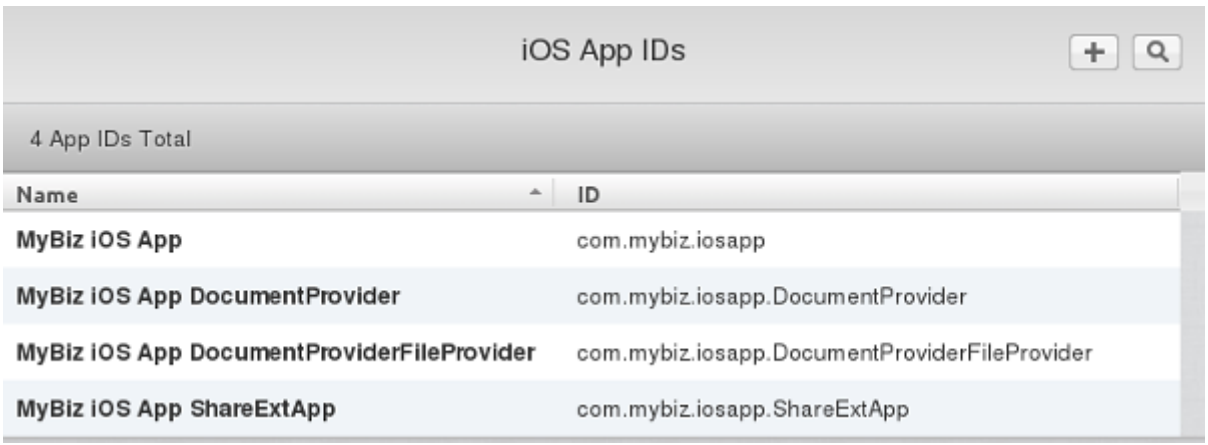
Select your group and click **[Continue]**.



On the confirmation screen click **[Assign]**, and you'll see the message "You have successfully updated the App Groups associations with your App ID."

Four Completed App IDs

Now you should have four new App IDs, and all of them should belong to your App Group.



Setting up Testing Devices

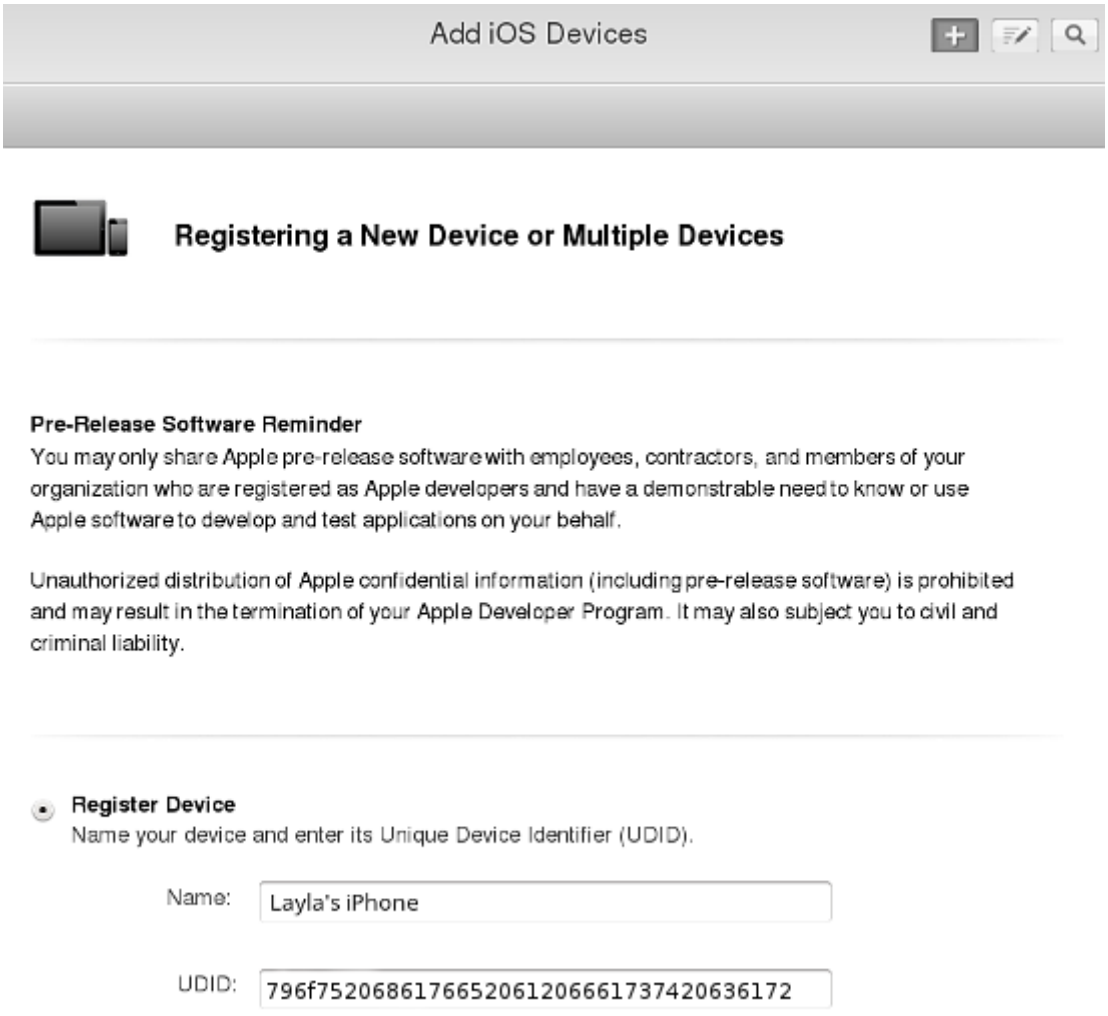
The $99 Apple Developer account allows you to test your iOS apps on a maximum of 100 devices of each type:

```
Apple TV     100
Apple Watch  100
iPad         100
iPhone       100
iPod Touch   100
```

And you must register the UDID of each device in your Apple developer account. If you have the $299 Enterprise account then you can install your app on any device without registering it.

The easiest way to find UDIDs is to connect to your iTunes account. Then connect your iOS device to your Mac computer. Your device will appear on the left sidebar in iTunes. Click on this to display your device information. Then click on the serial number, and you will see your UDID.

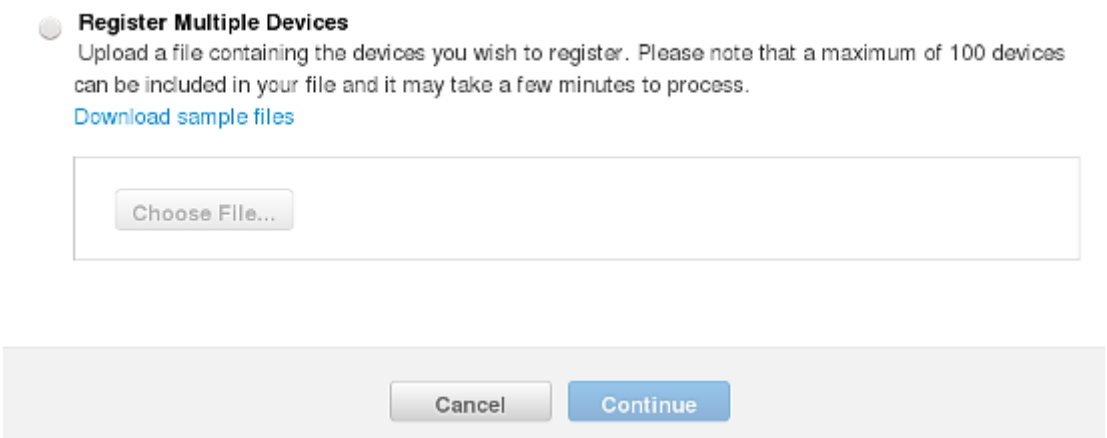
Return to your account on Developer.apple.com, go to **IOS Apps** > **Devices** > **All**, and click the plus button on the top right to register a new device. You can make the name anything you want, and the UDID must be the UDID copied from iTunes.



If you have a large number of devices to register, you may enter them in a text file in this format, and then upload the file:

```
Device ID  Device Name
A1234567890123456789012345678901234567890123456789   NAME1
B1234567890123456789012345678901234567890123456789   NAME2
```

Click **Download sample files** to see examples of plain text and markup files.



When you are finished entering your device IDs click the **Continue** button. Verify, and then click **Done**.

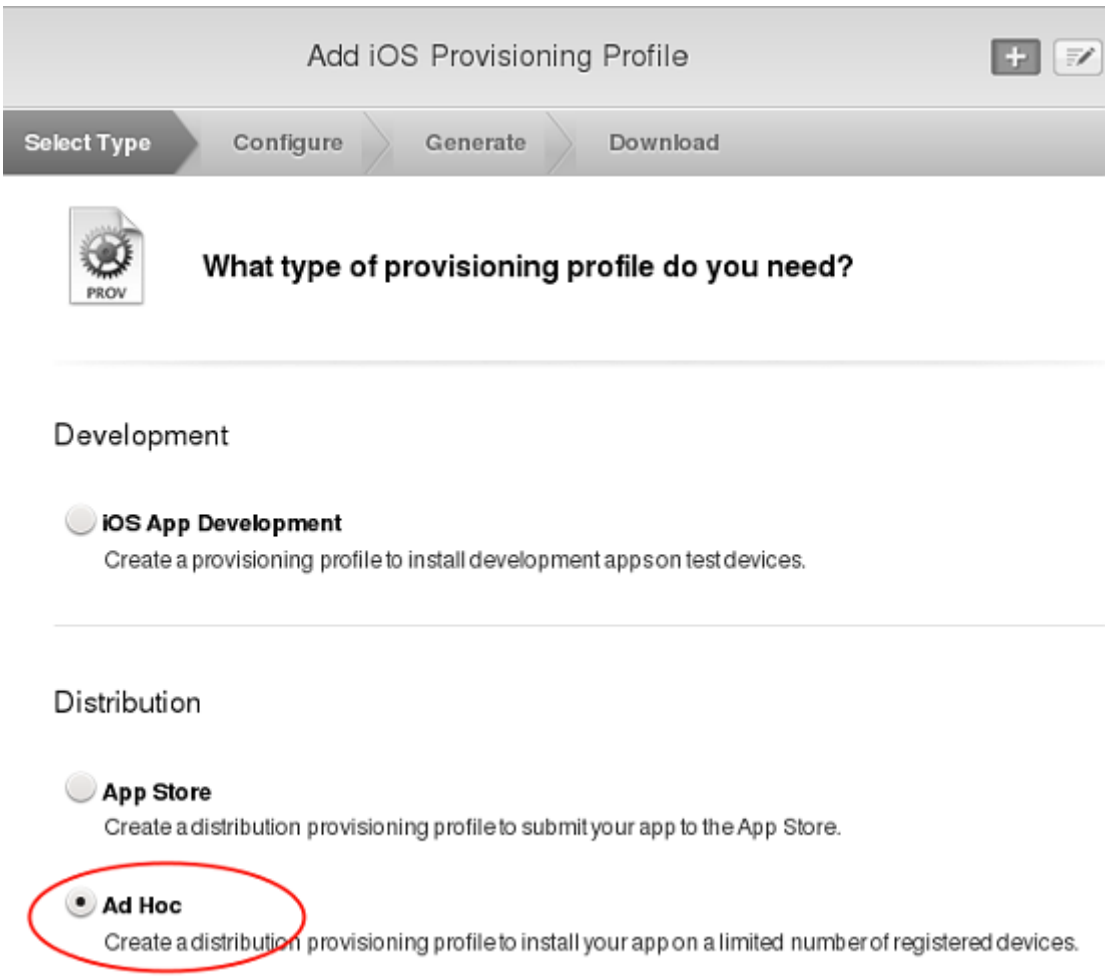
Create Provisioning Profiles

Next Step

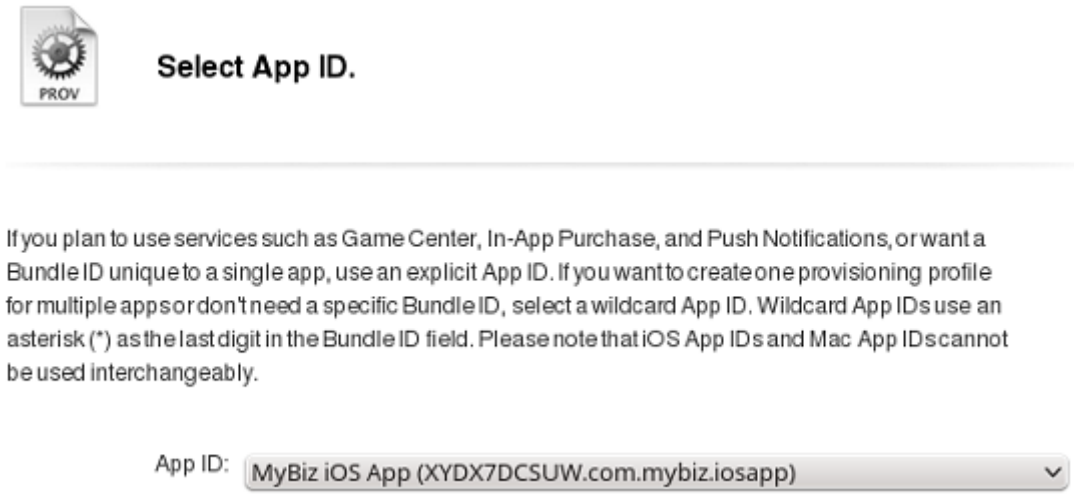
The next phase of this glorious journey is to create eight provisioning profiles: 4 Ad Hoc and 4 App Store <app_store_profiles_label>. You will email the four Ad Hoc profiles, and your P12 certificate <publishing_ios_app_6> (which you will create after your provisioning profiles), to support@owncloud.com after building your branded app with the ownBrander app on customer.owncloud.com. **Do not send us the App Store profiles.** All eight of these profiles must be stored on your Mac PC.

First Ad Hoc Provisioning Profile

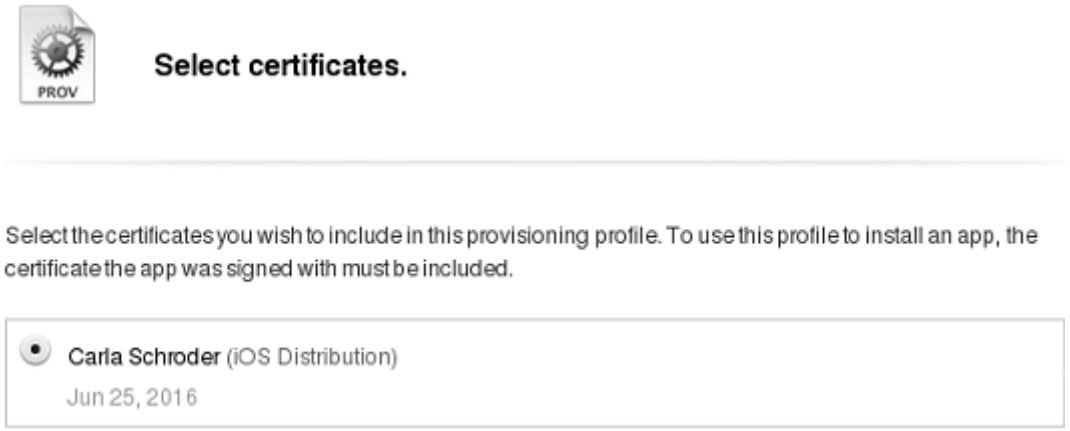
Go to **Provisioning Profiles** > **All**, then click the **[plus button]** (top right) to open the *Add iOS Provisioning Profile* screen. Select **[Ad Hoc]** and click **[Continue]**.



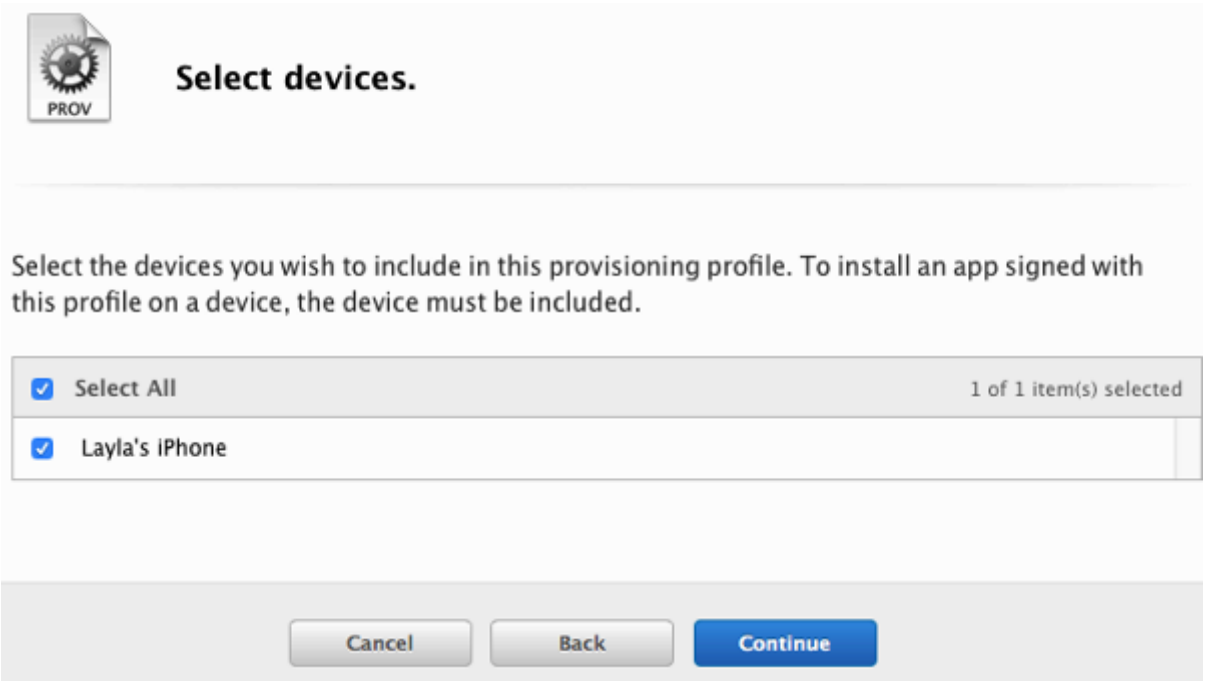
On the **Select App ID** screen select the first of the three App IDs that you created and click **[Continue]**. (The first one has the shortest name, if you followed the naming conventions in this manual.)



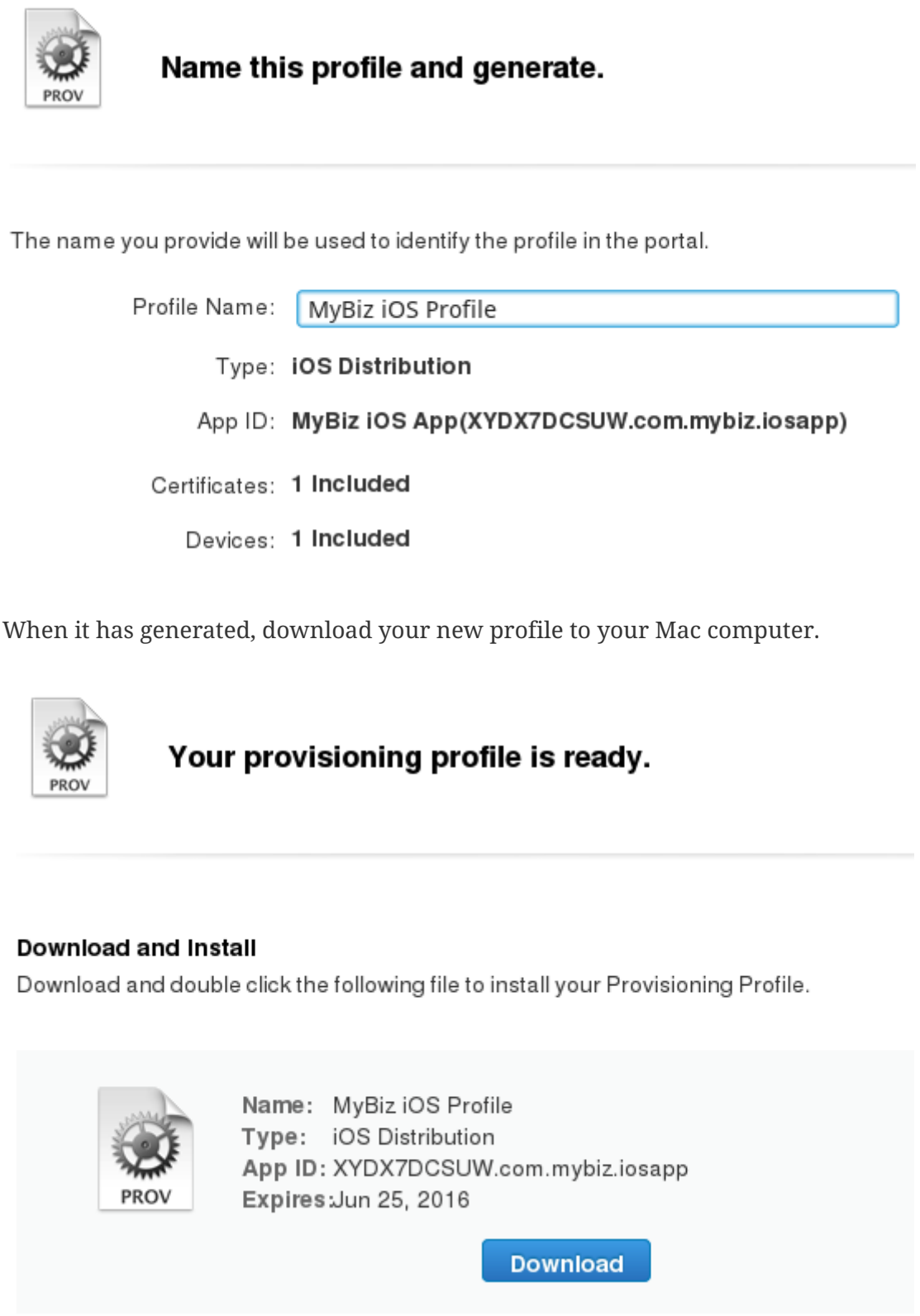
Select the certificate that you created at the beginning of this process and click **[Continue]**.



Select the devices that you want to install and test your app on, then click **[Continue]**.

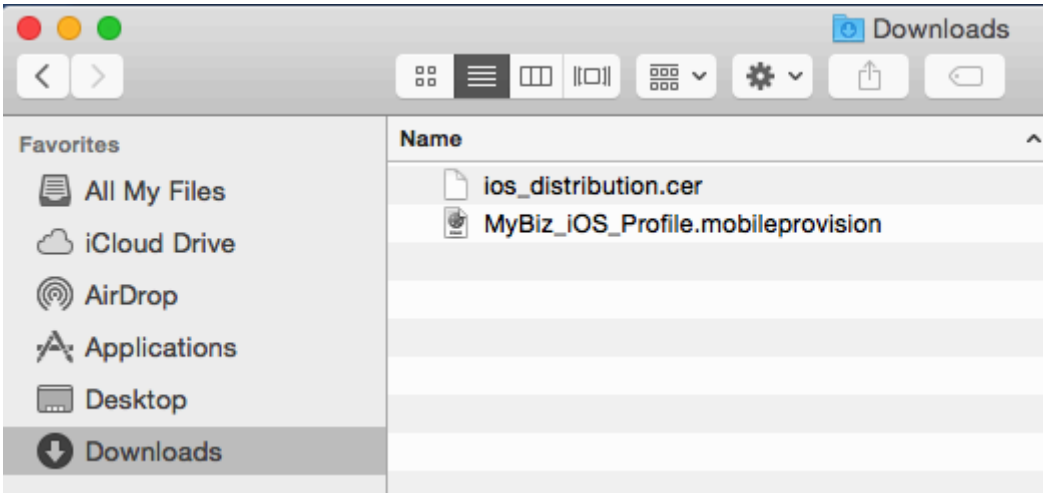


Name your provisioning profile with a descriptive **Profile Name** and click **[Generate]**.



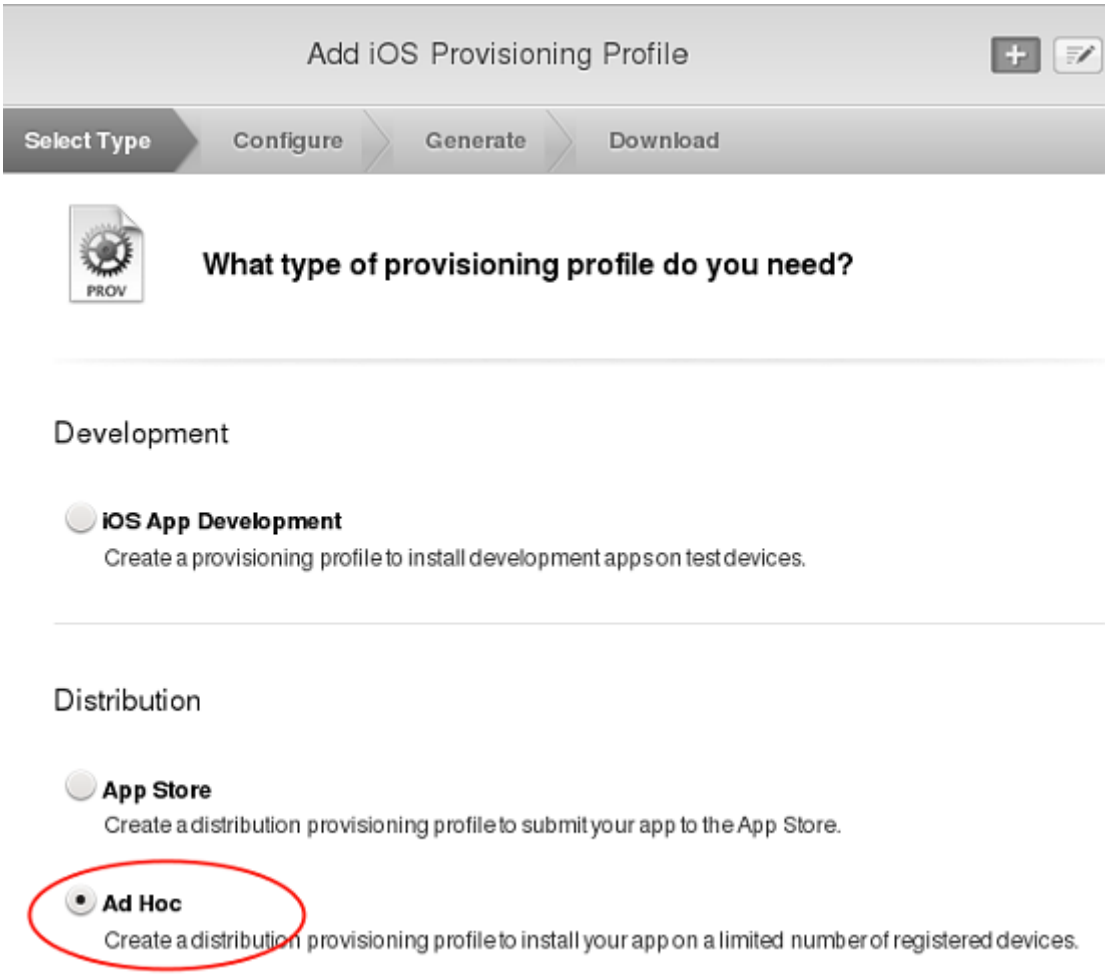
When it has generated, download your new profile to your Mac computer.

Find it on your Mac (usually the Download folder) and double-click to install it in Xcode.

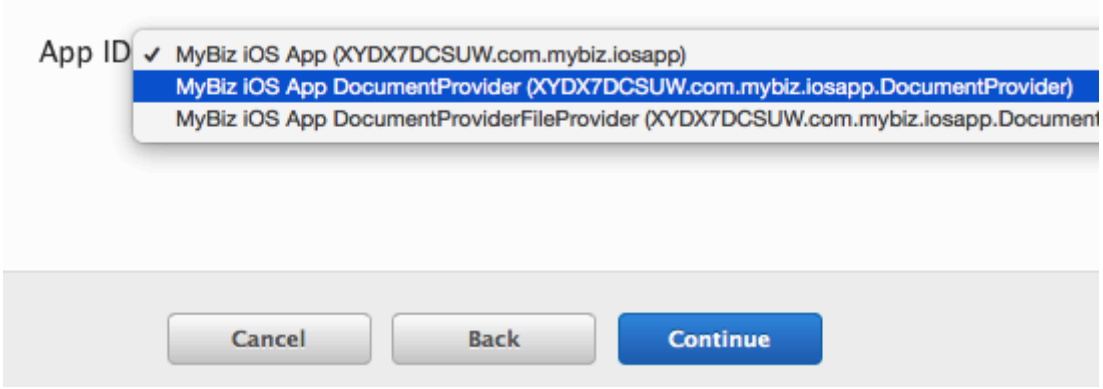


Second Ad Hoc Provisioning Profile

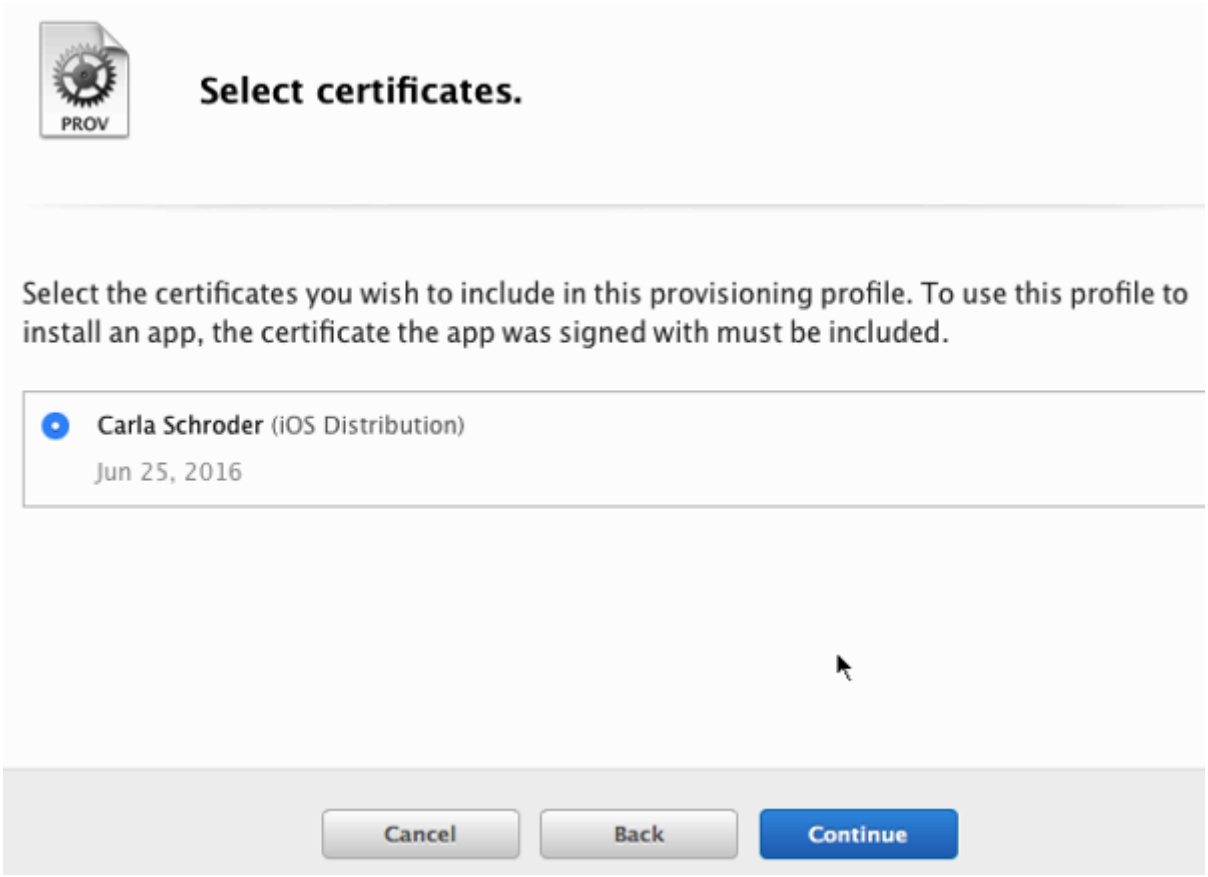
Return to the "Your provision profile is ready" screen, scroll to the bottom and click **[Add Another]**. On the following screen select **[Ad Hoc]** and click **[Continue]**.



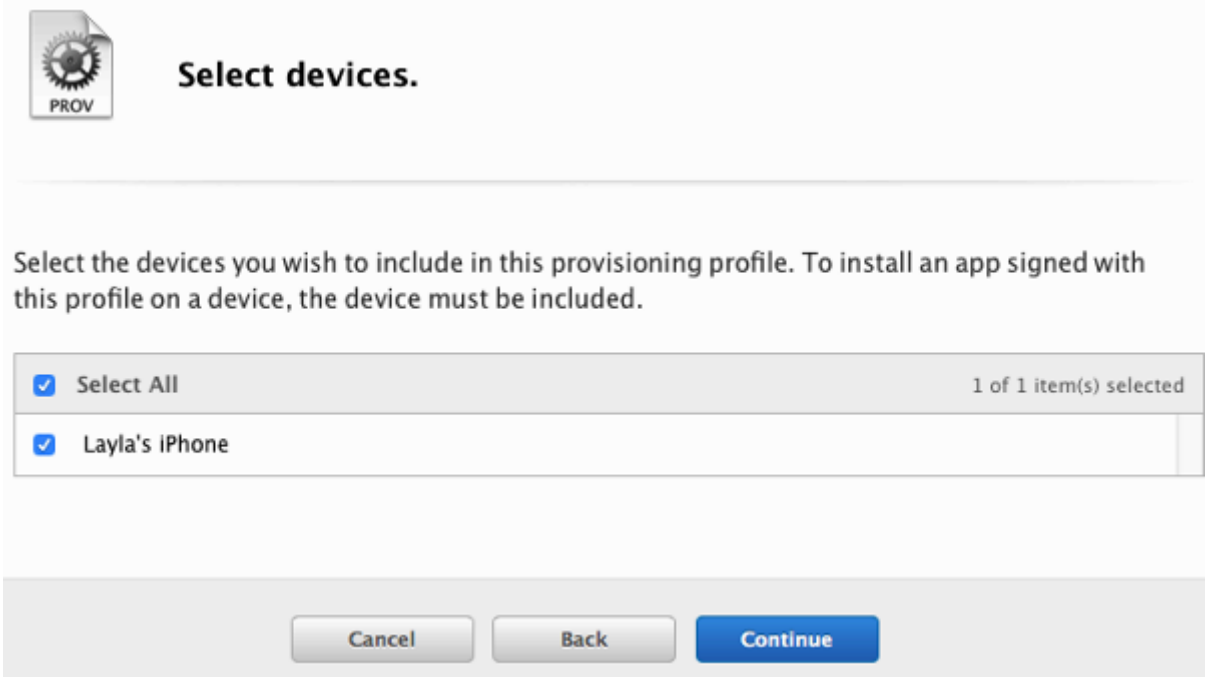
This time select the **.DocumentProvider** app ID and click **[Continue]**.



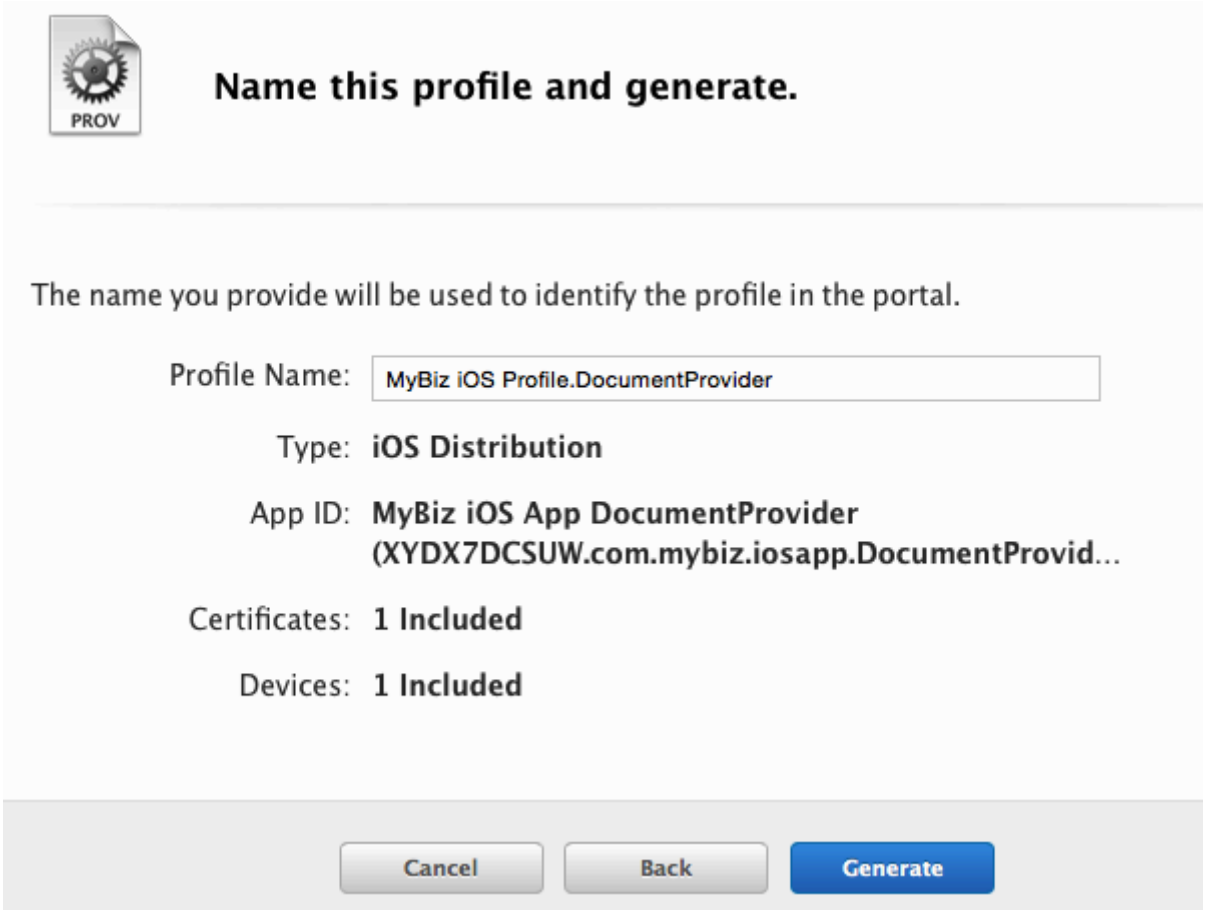
Select the certificate that you created at the beginning of this process and click **[Continue]**.



Select the devices that you want to install and test your app on, then click **[Continue]**. These must be the same devices you selected for the first provisioning profile.



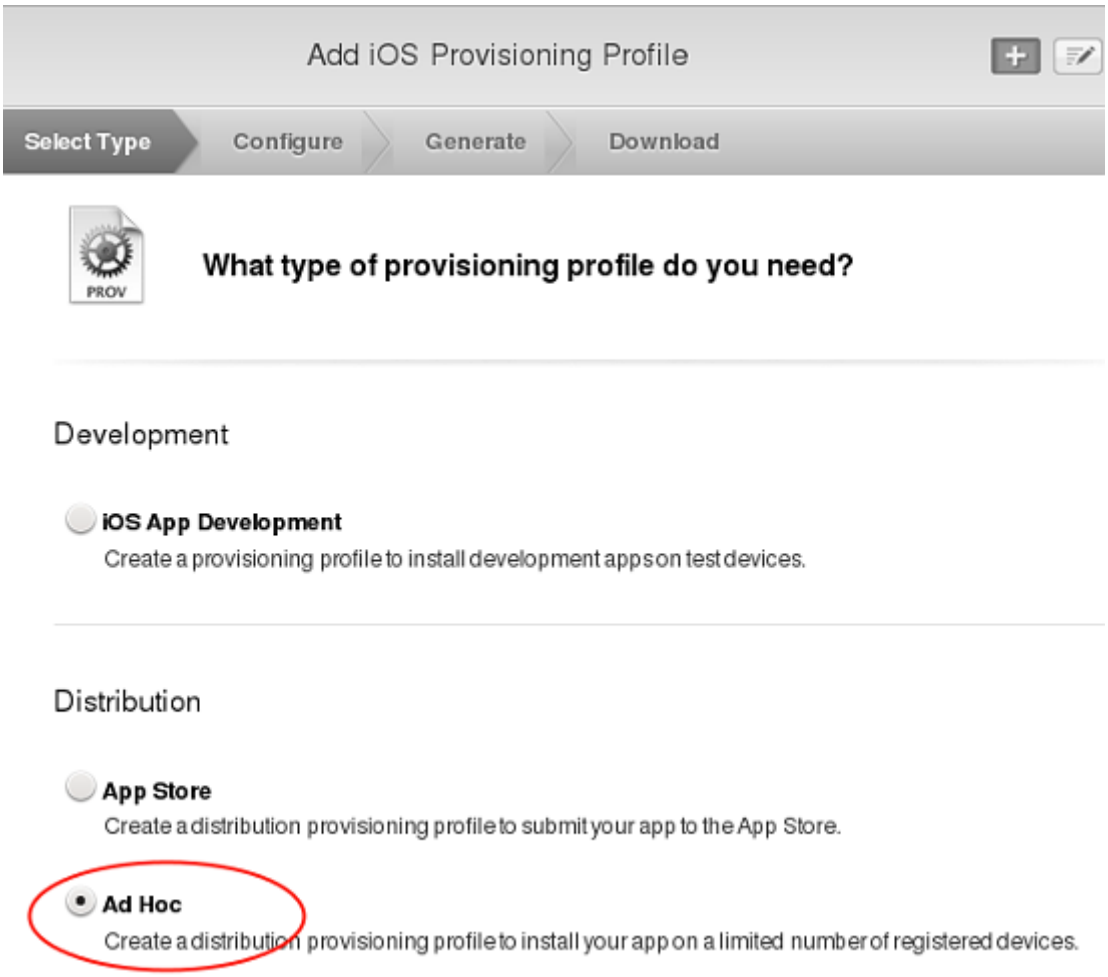
Give this provisioning profile the same name as your first profile, plus **.DocumentProvider** and click **[Generate]**.



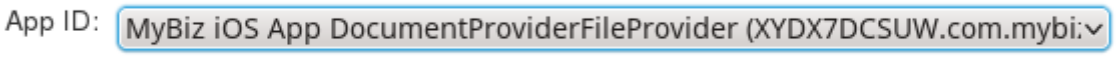
Just like the first provisioning profile, download it to your Mac computer, and then double-click to install it in Xcode.

Third Ad Hoc Provisioning Profile

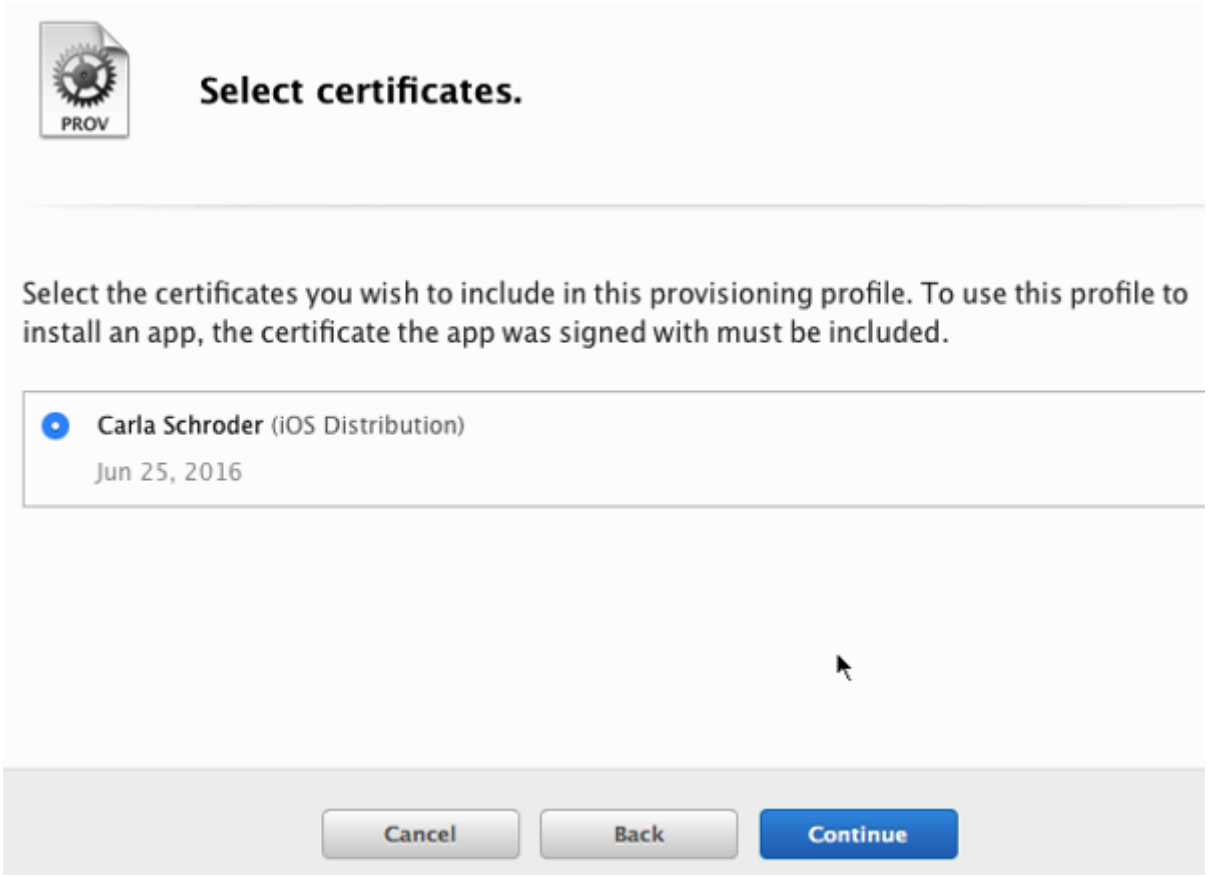
Return to the "Your provision profile is ready" screen, scroll to the bottom and click **[Add Another]**. On the following screen select **[Ad Hoc]** and click **[Continue]**.



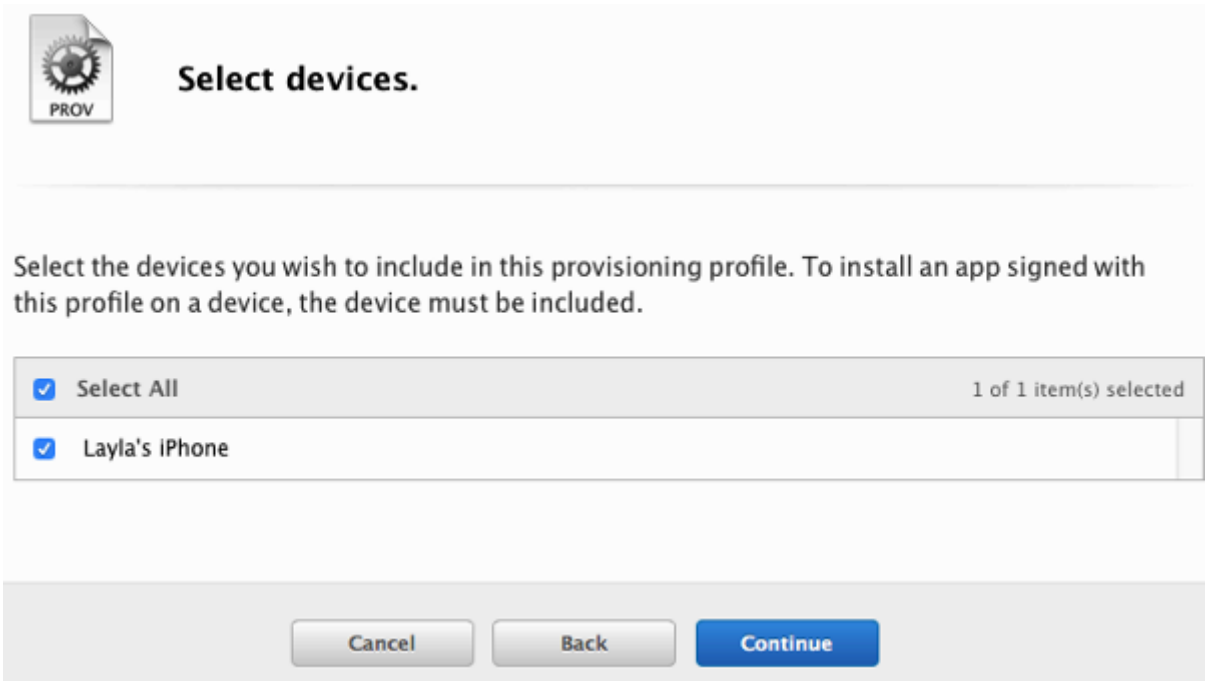
This time select the **.DocumentProviderFileProvider** app ID and click **Continue**.



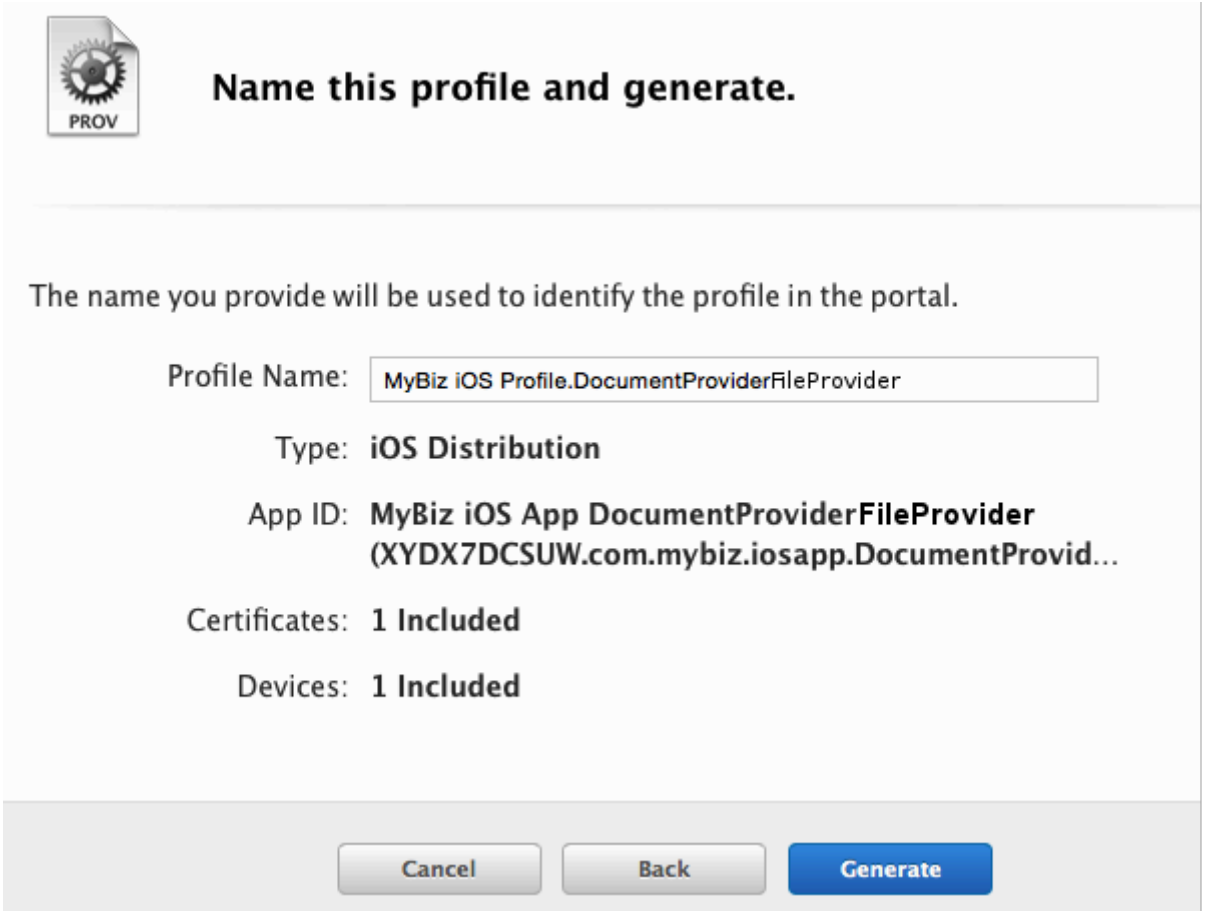
Select the certificate that you created at the beginning of this process and click **[Continue]**.



Select the devices that you want to install and test your app on, then click **[[Continue]**. These must be the same devices you selected for the first provisioning profile.



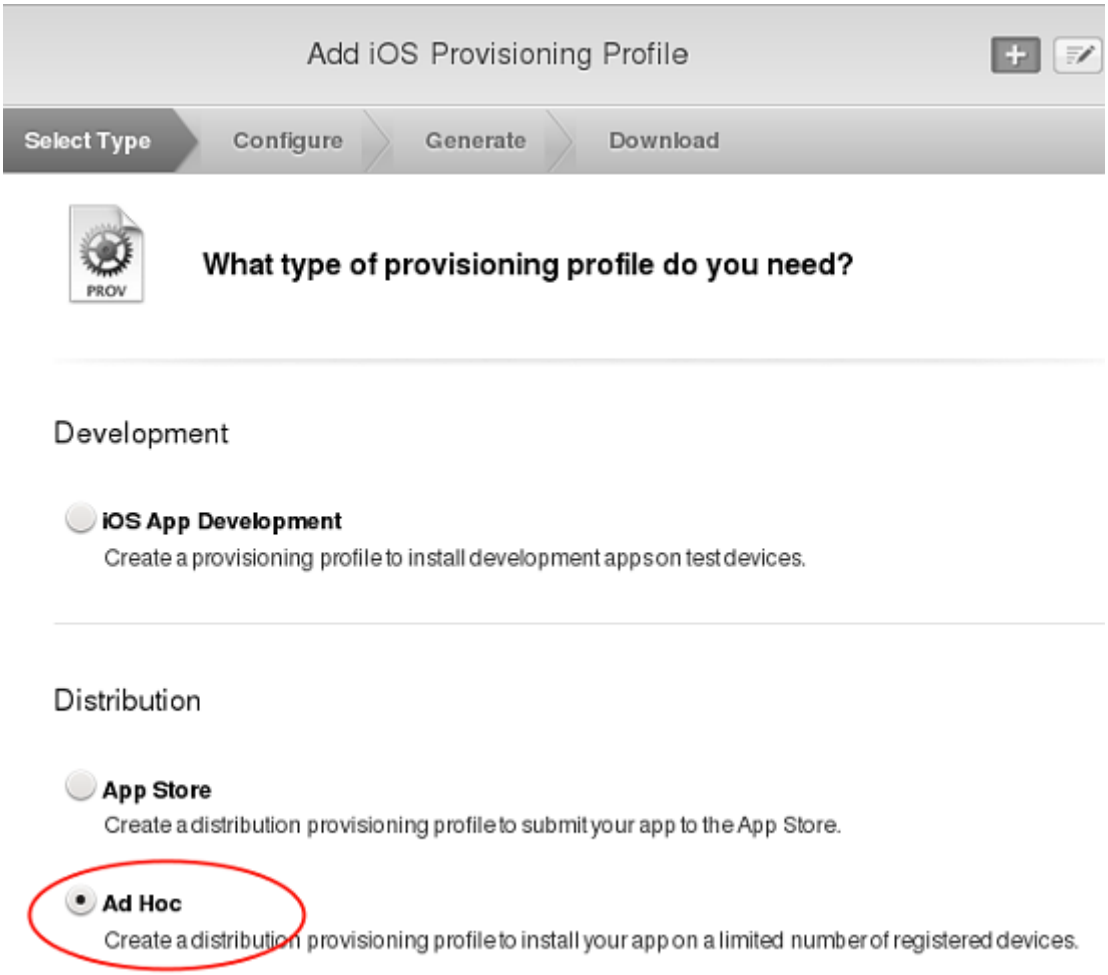
Give this provisioning profile the same name as your first profile plus **.DocumentProviderFileProvider** and click **[Generate]**. There is a 50-character limit, but don't worry about counting characters because it will be automatically truncated if you go over.



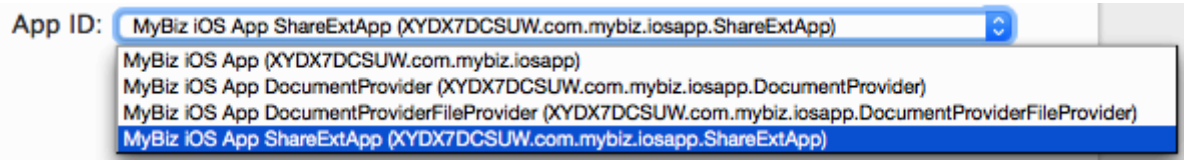
Download it to your Mac computer, and then double-click to install it in Xcode.

Fourth Ad Hoc Provisioning Profile

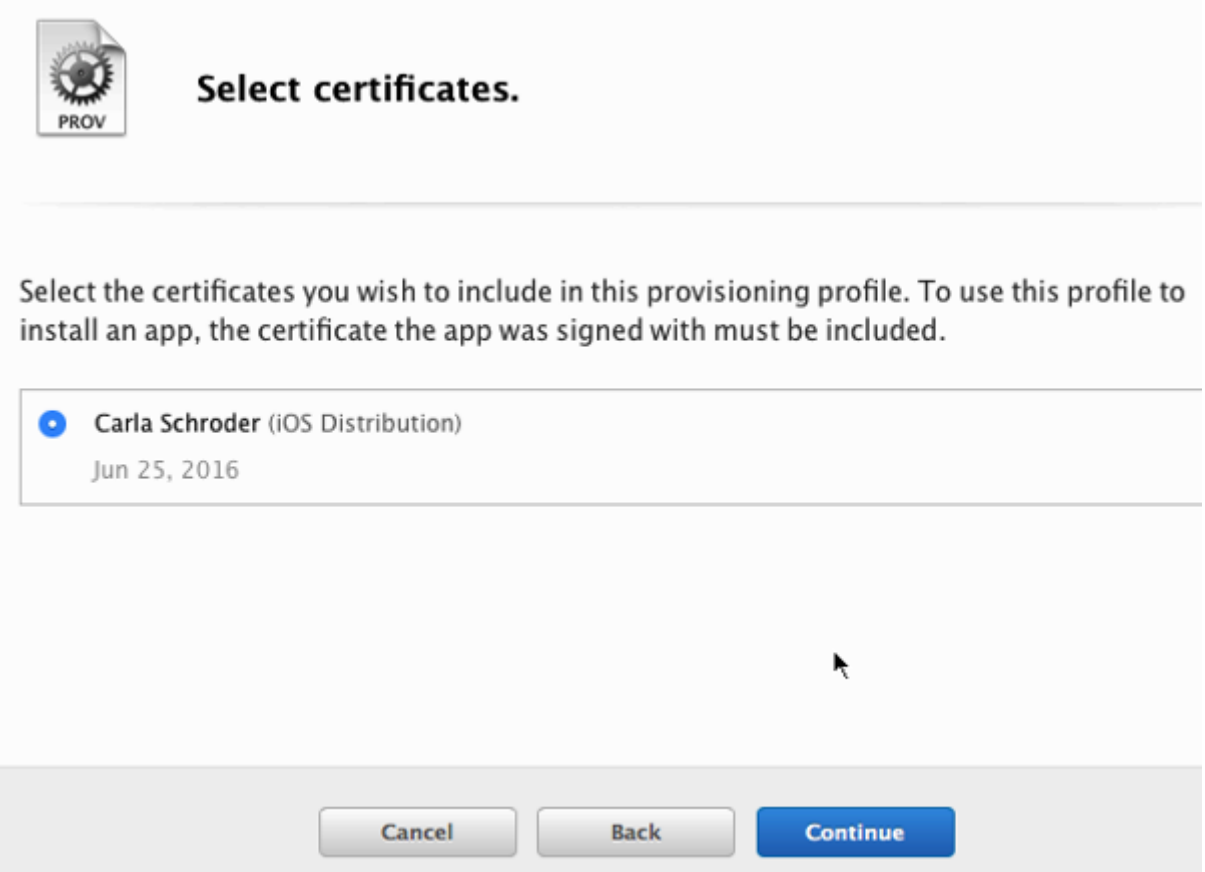
Return to the "Your provision profile is ready" screen, scroll to the bottom and click **[Add Another]**. On the following screen select **[Ad Hoc]** and click **[Continue]**.



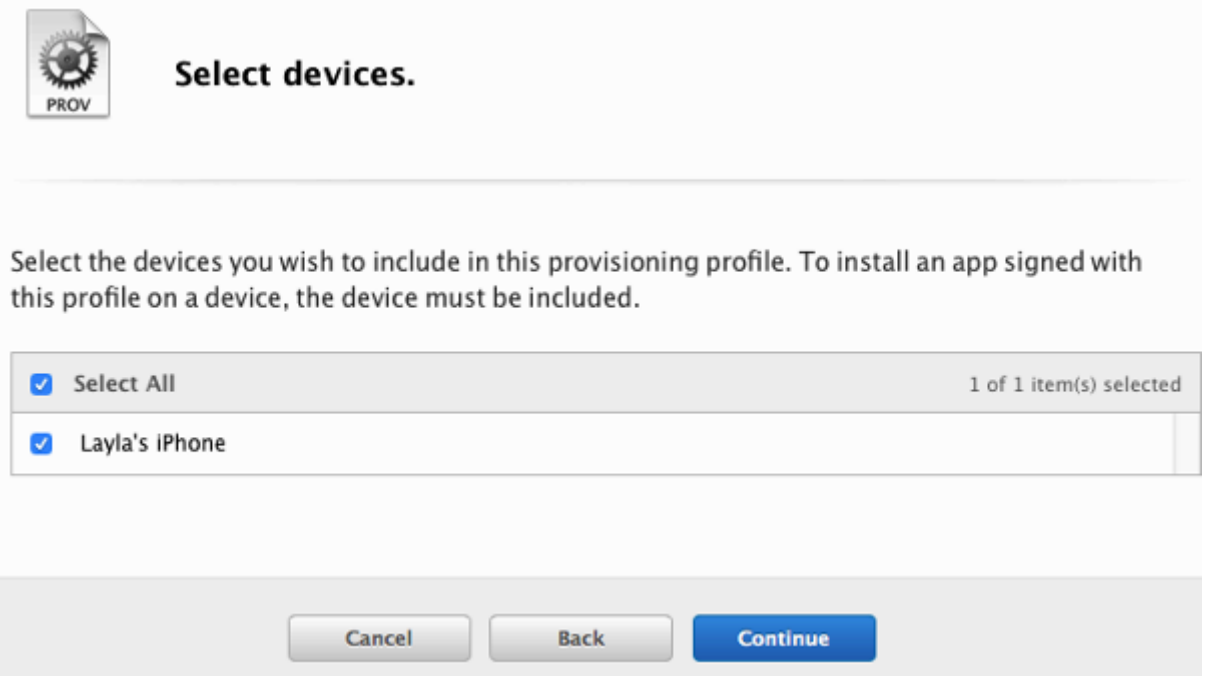
This time select the **.ShareExtApp** app ID and click **[Continue]**.



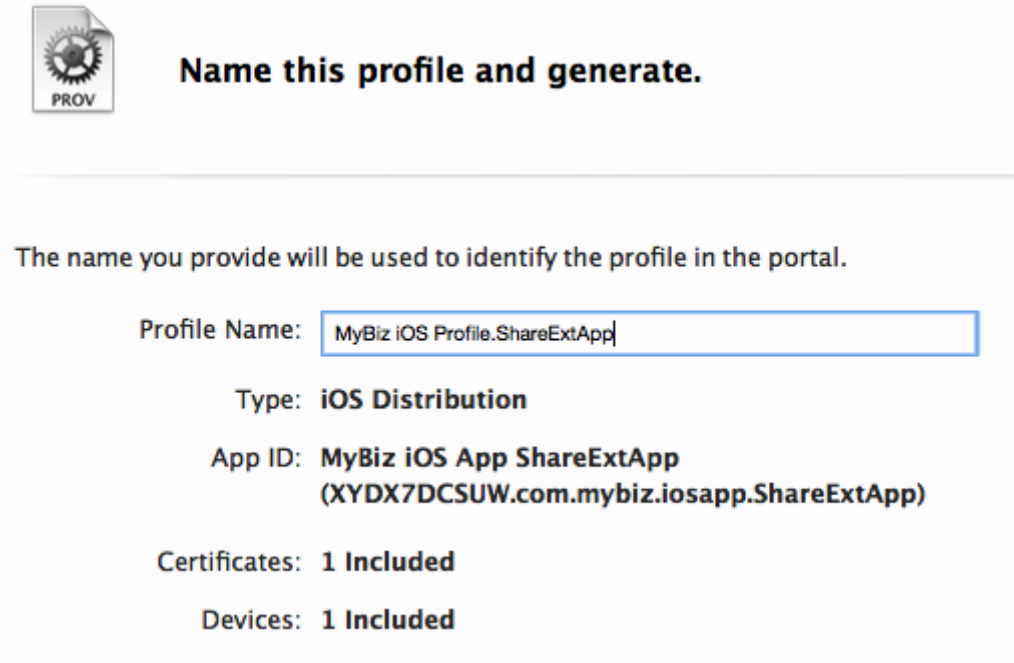
Select the certificate that you created at the beginning of this process and click **[Continue]**.



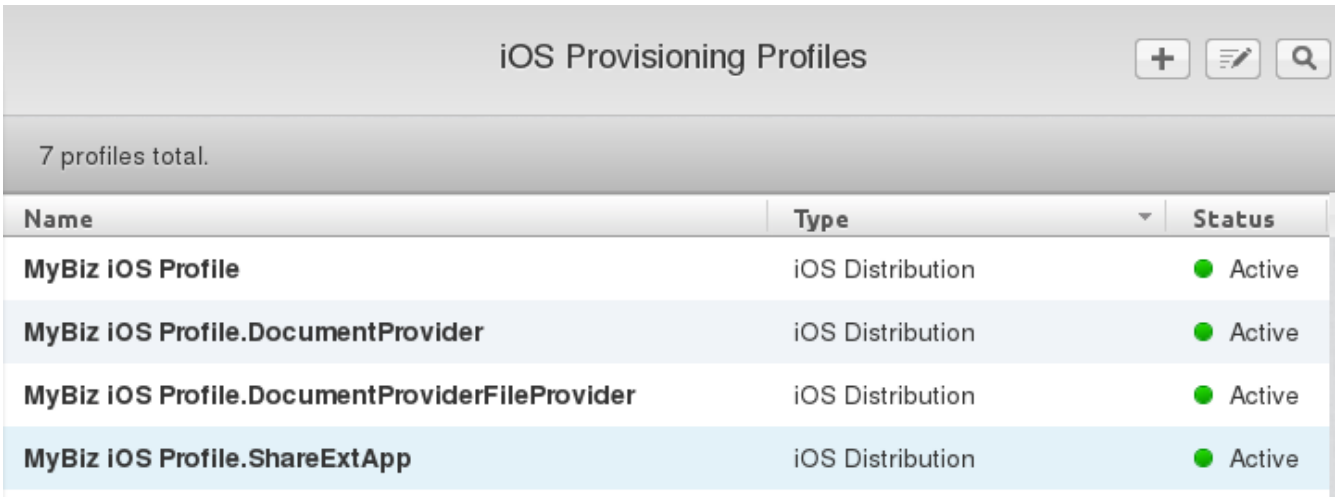
Select the devices that you want to install and test your app on, then click **[Continue]**. These must be the same devices you selected for the first provisioning profile.



Give this provisioning profile the same name as your first profile plus **.ShareExtApp** and click **[Generate]**. There is a 50-character limit, but don't worry about counting characters because it will be automatically truncated if you go over.

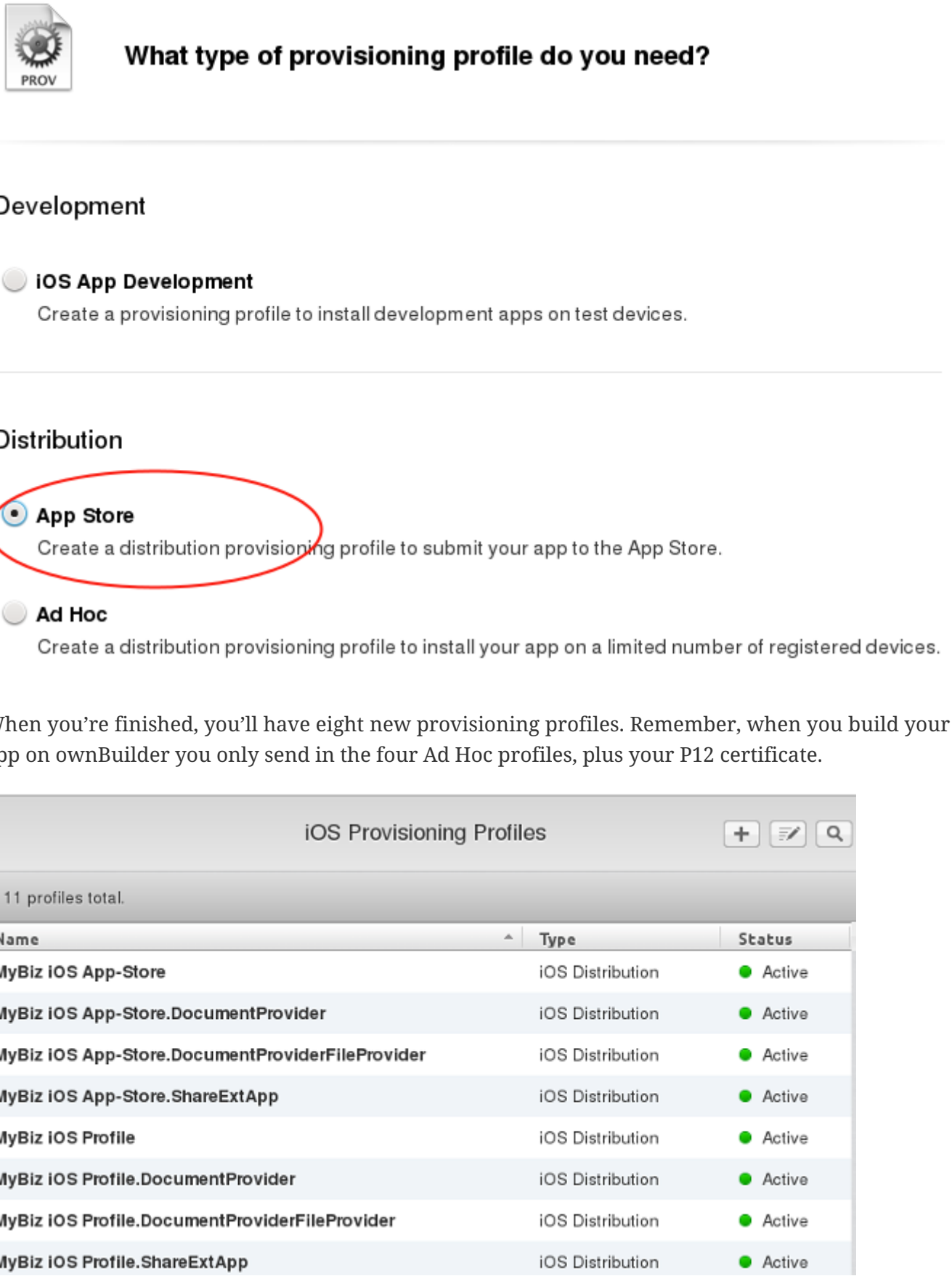


Download it to your Mac computer, and then double-click to install it in Xcode. You should now see all of your Ad Hoc provisioning profiles listed in your "iOS Provisioning Profiles".



Create Four App Store Profiles

Creating your four App Store profiles is the same as creating your Ad Hoc profiles, except that when you start you check the App Store checkbox, and you won't select testing devices.

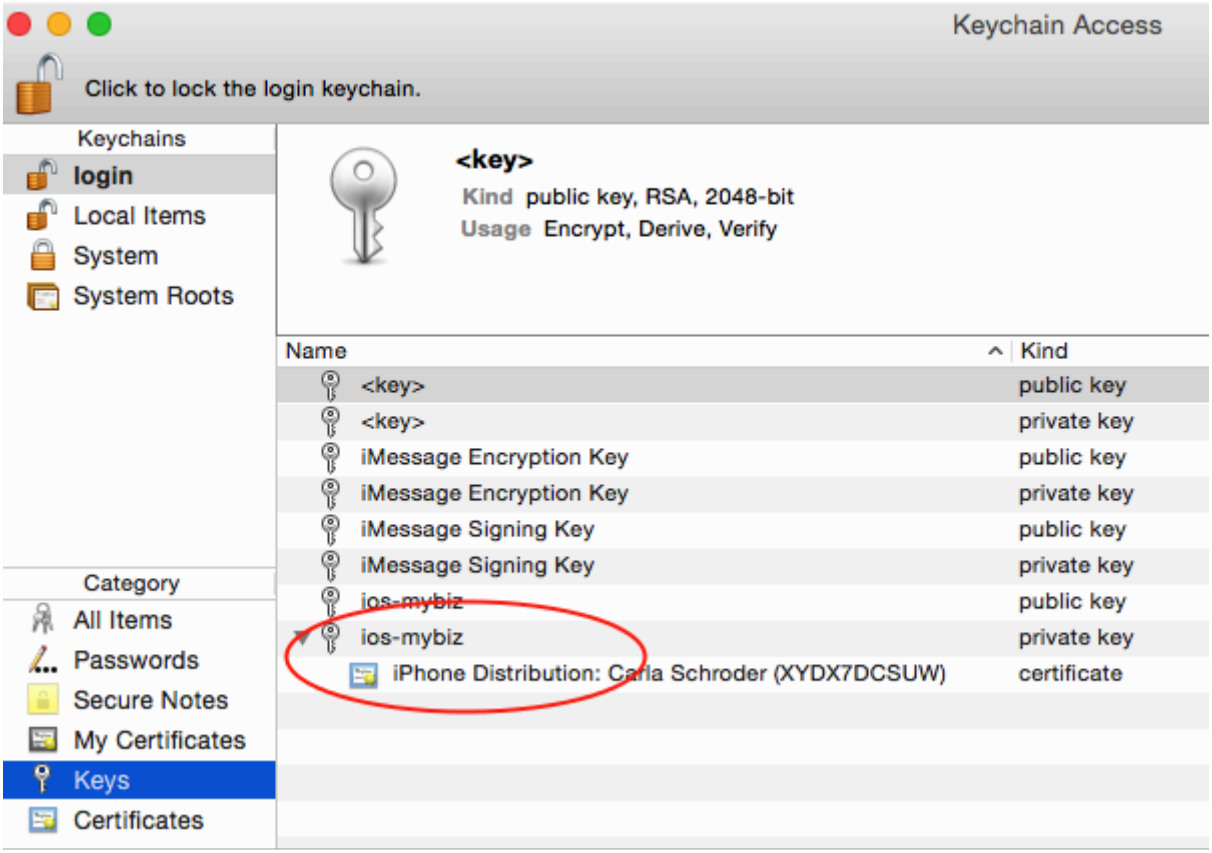


When you're finished, you'll have eight new provisioning profiles. Remember, when you build your app on ownBuilder you only send in the four Ad Hoc profiles, plus your P12 certificate.

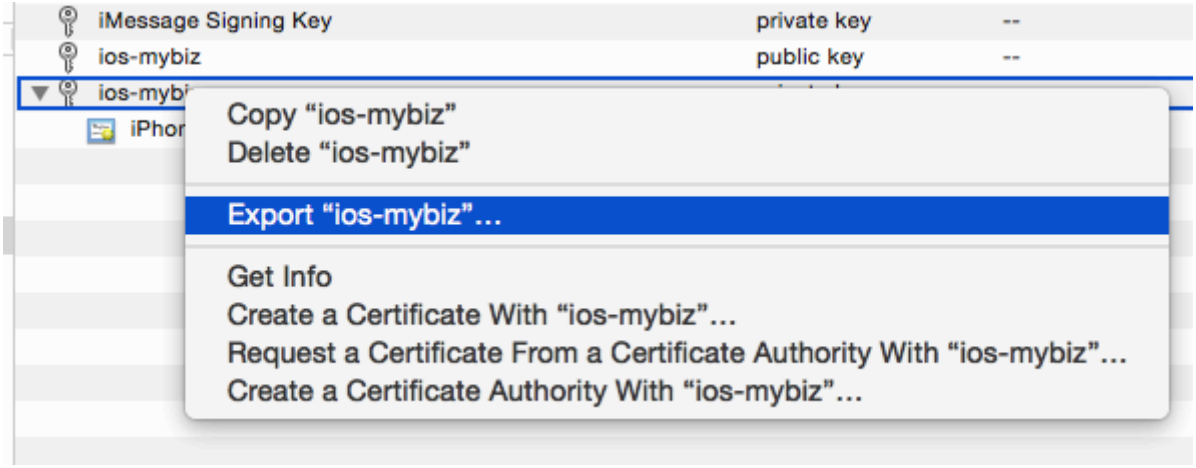
Go to the next page to learn how to create your P12 certificate <publishing_ios_app_6>.

Creating a P12 Certificate

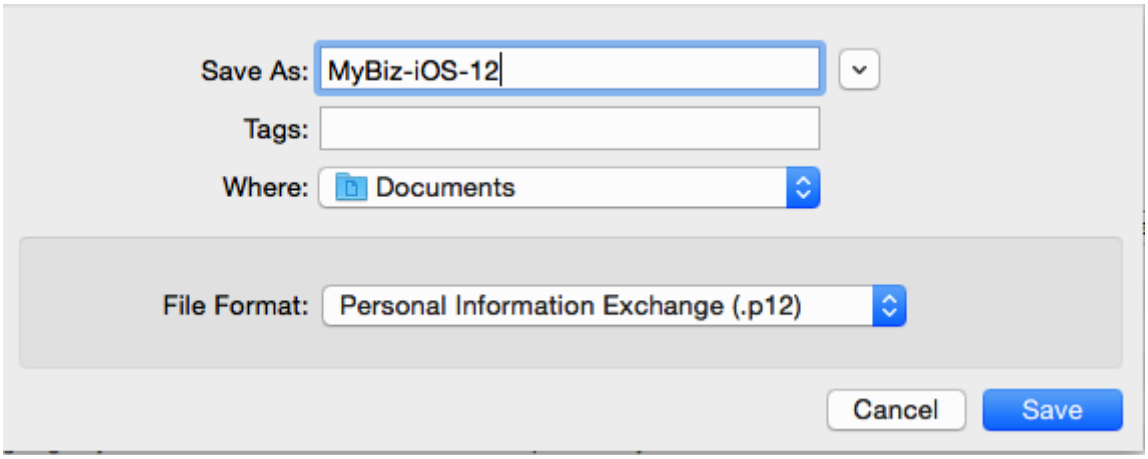
In addition to emailing your four Ad Hoc provisioning profiles to support@owncloud.com, you must also include your P12 certificate. To create this, return to Keychain Access on your Mac computer and find your private key that you created at the beginning (see Create Certificate Signing Request).



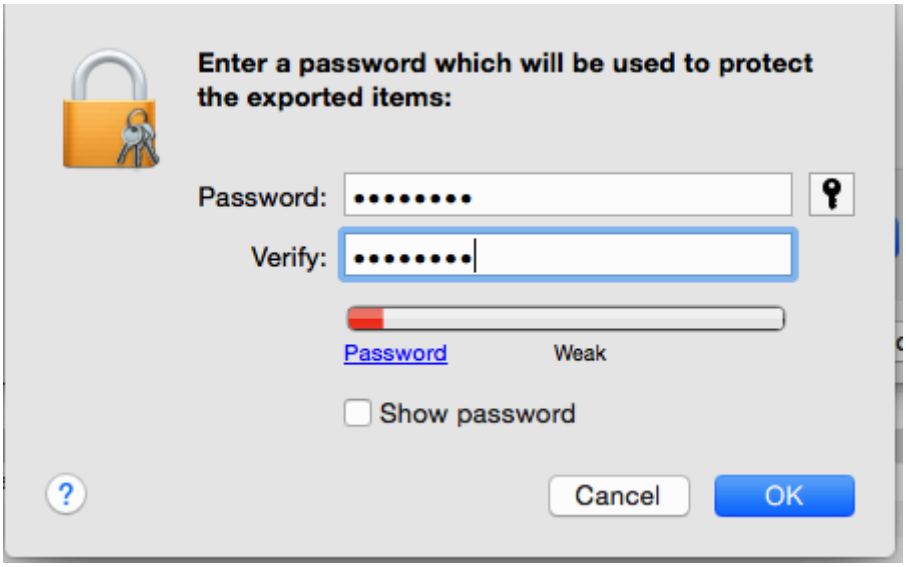
Right-click on your private key and left-click **Export [your key name]**.



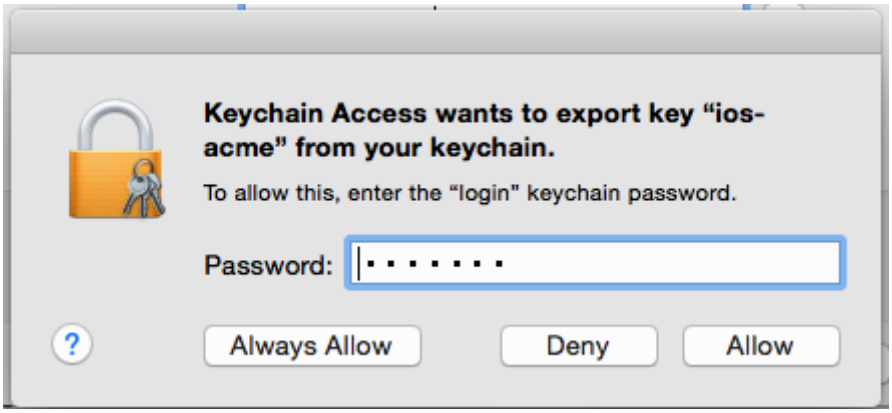
Enter any name you want, the location you want to save it to, and click **[Save]**.



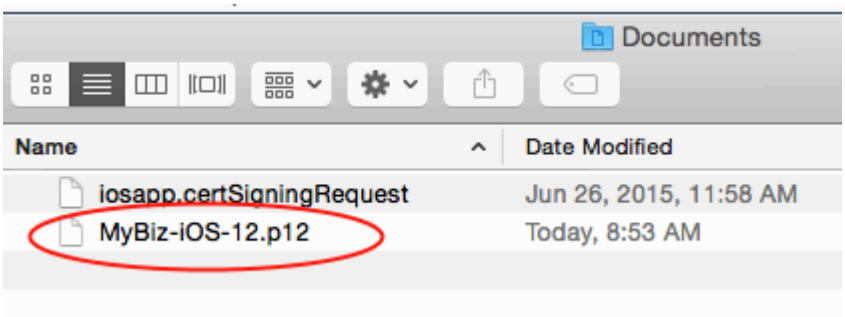
In the next screen you have the option to enter a password. If you put a password on your P12 certificate you will have to include it when you send your certificate and provisioning profiles to support@owncloud.com. Click **[OK]**.



On the next screen you must enter your login keychain password, which is your Mac login password, and click **[Allow]**.



Now your new P12 certificate should be in the directory you saved it in.

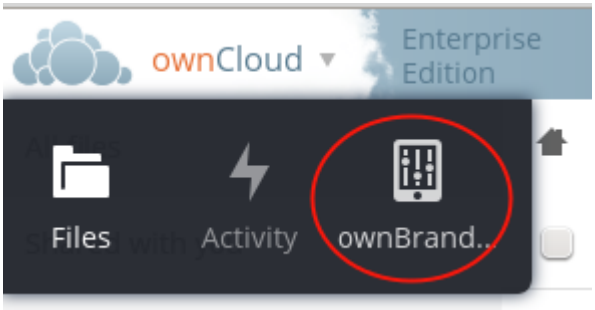


You have now completed all the necessary steps for signing your branded iOS app. The next step is to build your app with the ownBrander app on https://customer.owncloud.com.

Building Your iOS App With ownBrander

Build Your Branded iOS App

At long last you have arrived at the point where you can actually build your branded iOS app. Log into your account on customer.owncloud.com/owncloud and open the ownBrander app.



If you don't see the ownBrander app, open a support request with the **[Open Case]** button.

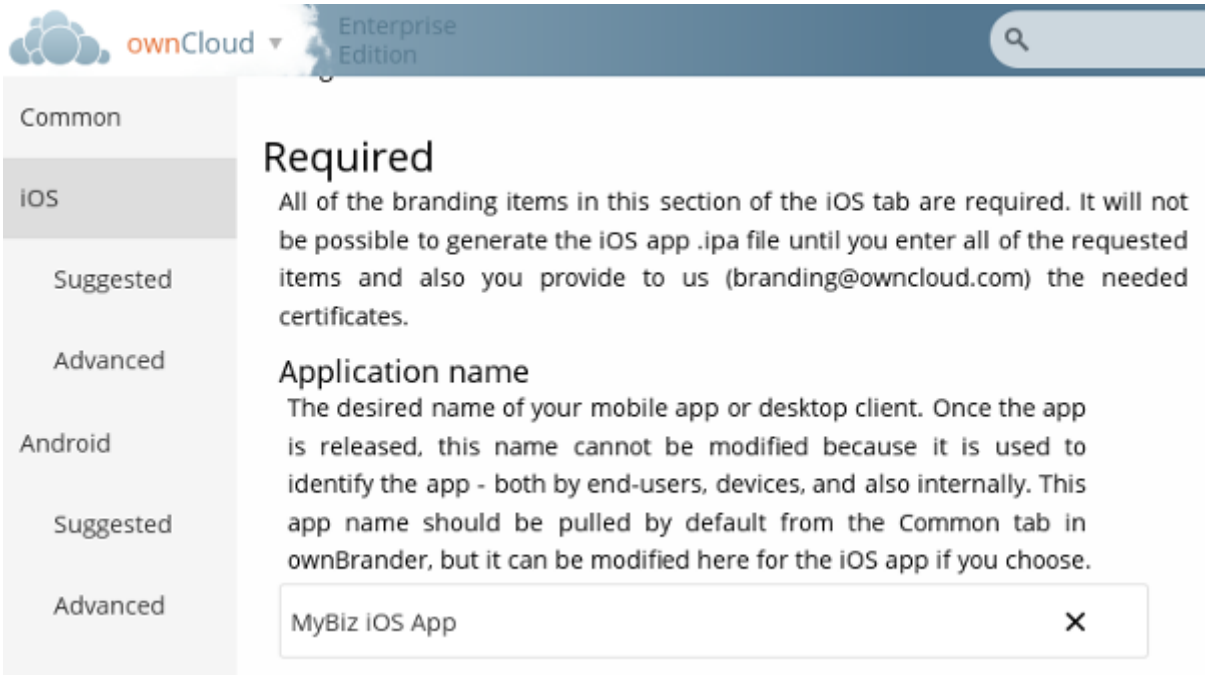
Your first ownBrander task is to review the iOS page on ownBrander for your image requirements. You will need a lot of them, in specific sizes and formats, and they're all listed on the ownBrander page.

There are three sections: Required, Suggested, and Advanced. The Required sections contains all of the required elements that you must configure. Suggested and Advanced allow additional customizations.

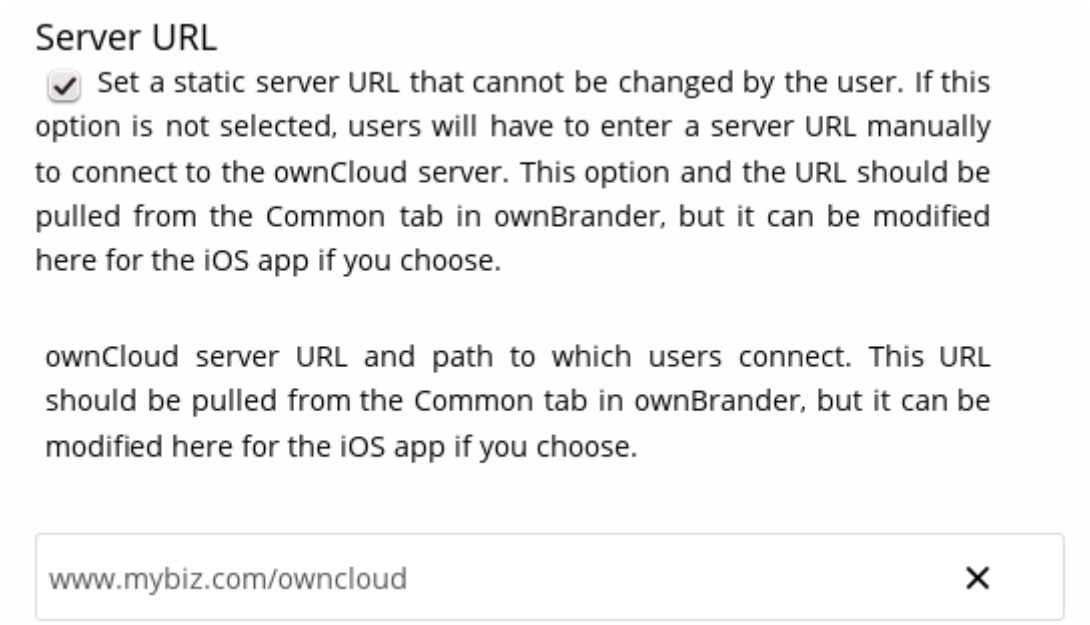
When you have completed and submitted your app, email your three provisioning profiles and P12 certificate to support@owncloud.com.

Required Section

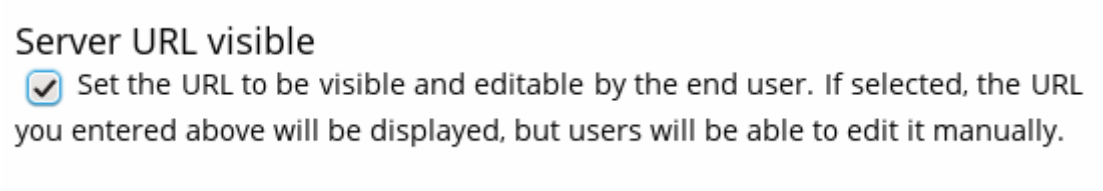
Enter your application name. This can be anything; in this example it is the same name used in our signing certificate examples.



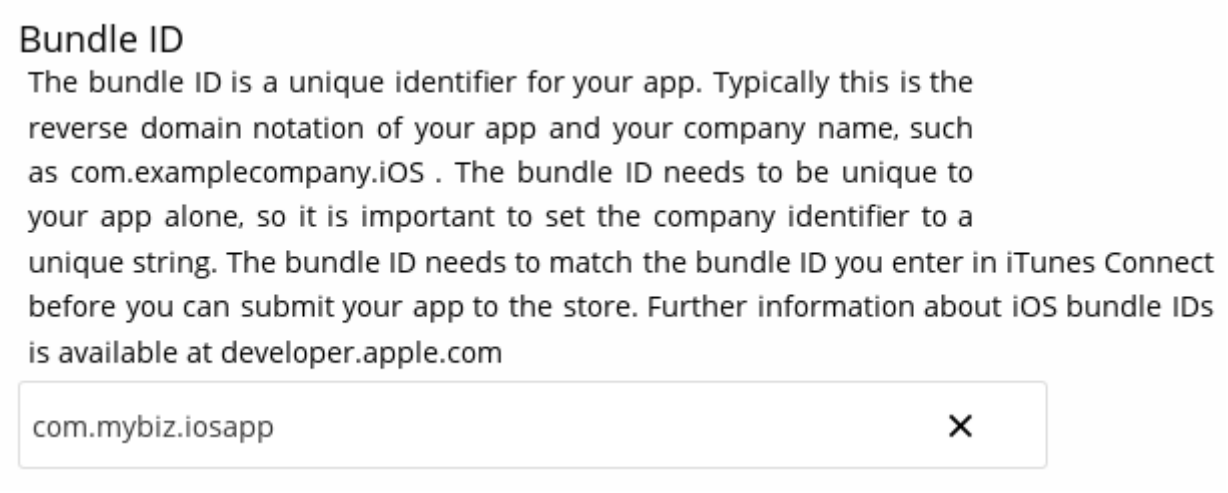
Next, enter your ownCloud server URL. This hard-codes it into your app. If you leave this blank then your users will have to enter it every time they use the app.



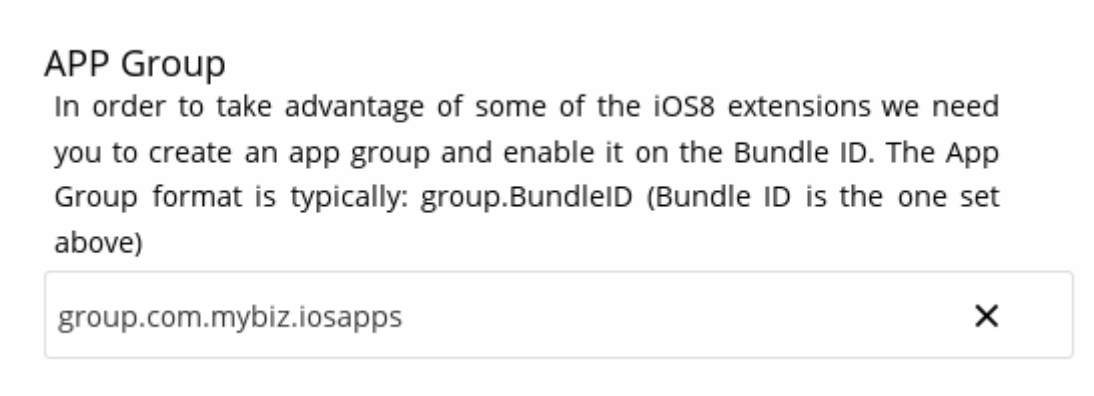
Check **Server URL Visible** to make your ownCloud server URL the default, and to allow your users to enter a different URL.



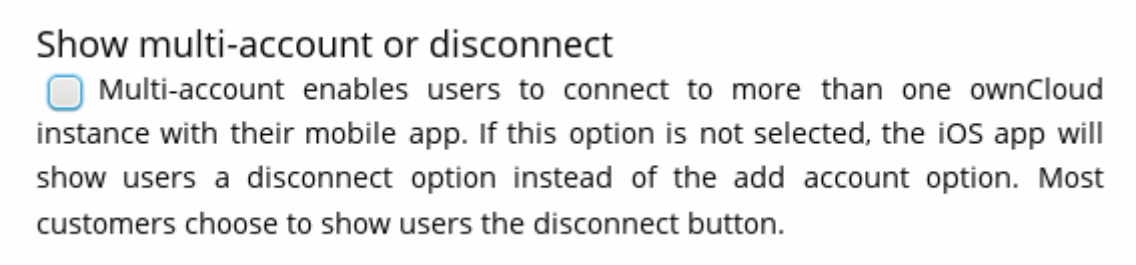
And now, the all-important **Bundle ID**. Make sure that this is exactly the same as the **Bundle ID** you created on developer.apple.com (see Bundle ID).



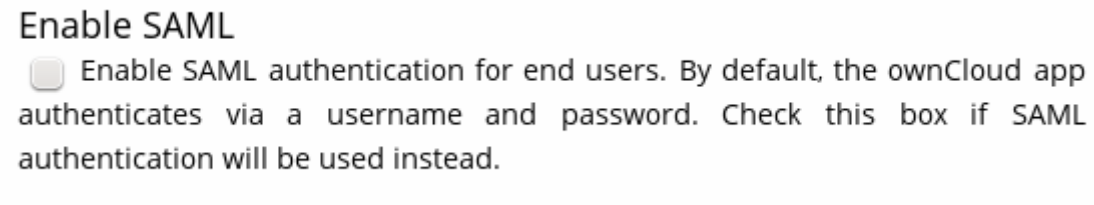
You must also enter the **App Group** you created.



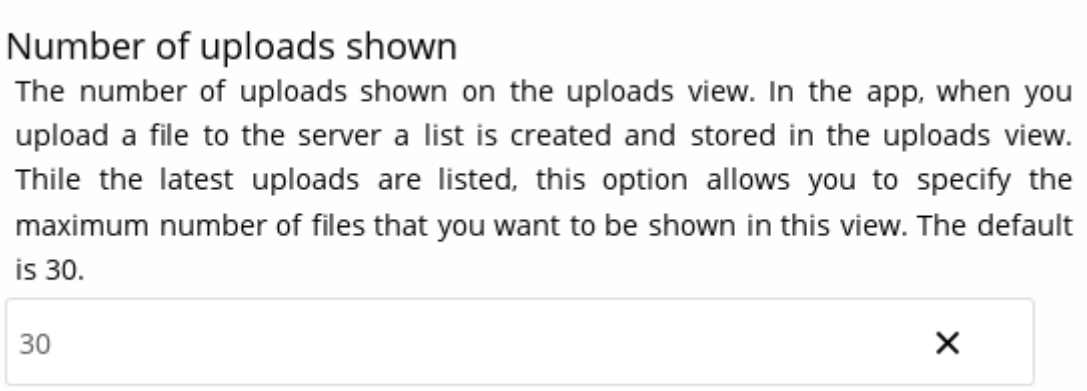
Check **Show multi-account or disconnect** if you plan to allow your users to have more than one ownCloud account.



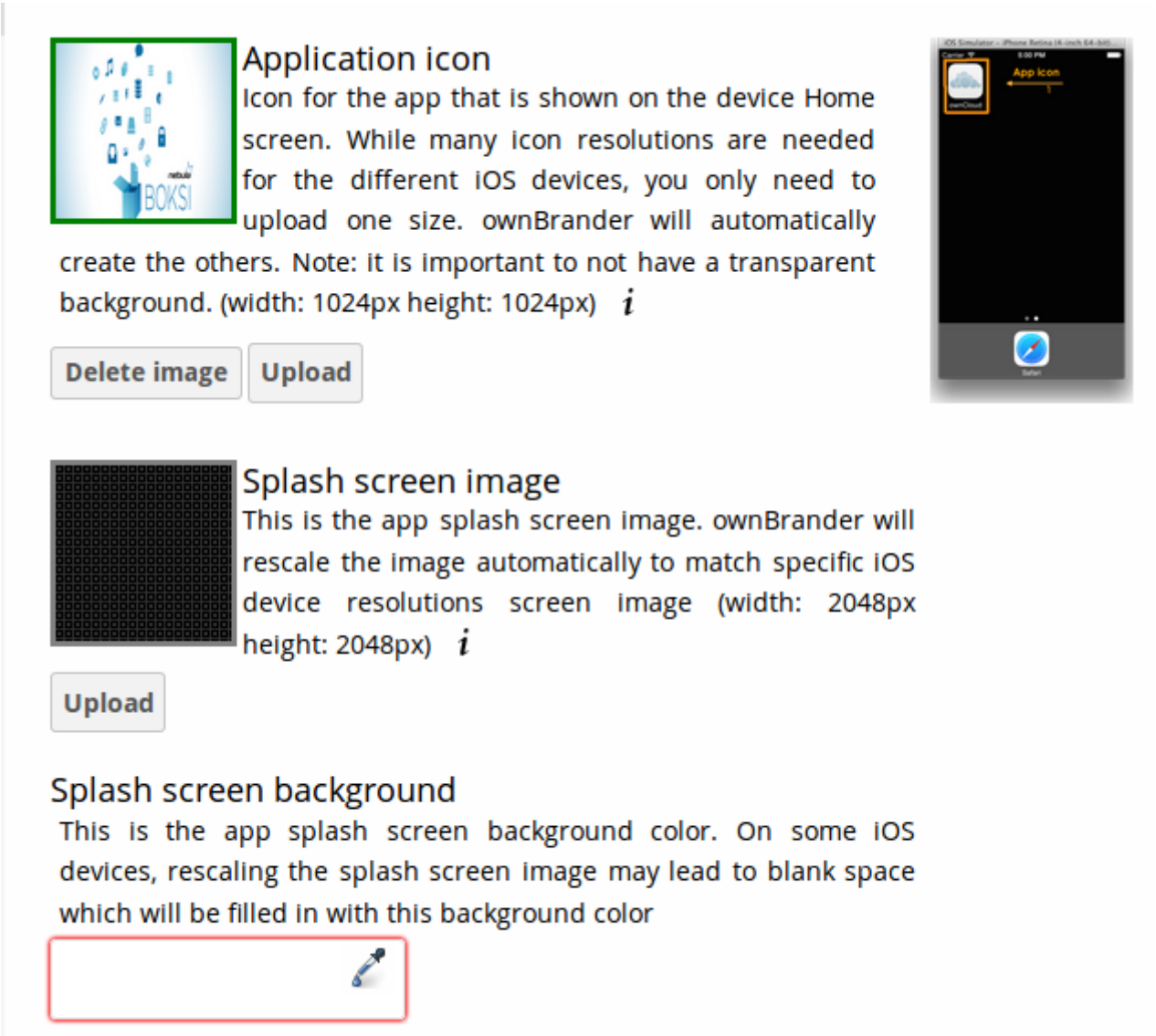
Check **Enable SAML** authentication if that is what you use on your ownCloud server. Otherwise leave it blank.



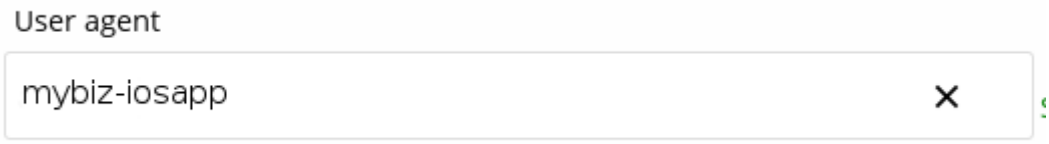
Number of uploads shown controls the length of the most recent uploads list on the app. The default is 30.



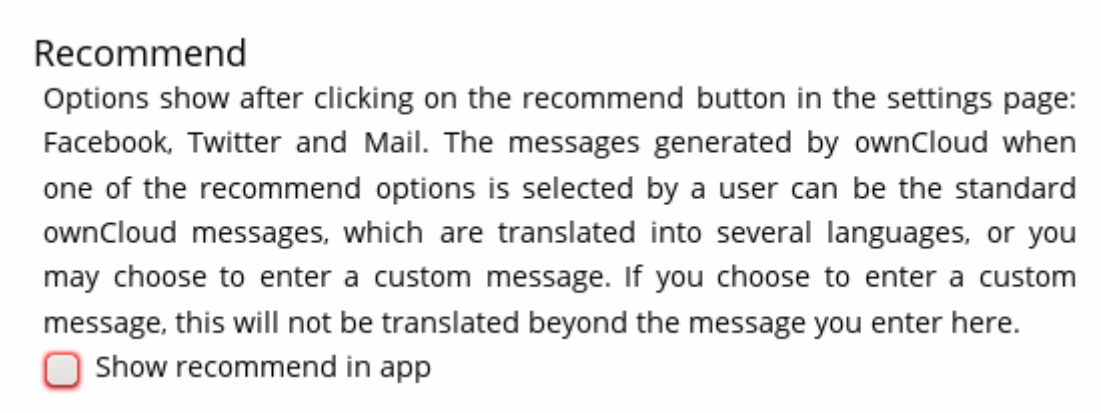
The next section is for uploading your custom artwork to be built into the app. The ownBuilder app tells you exactly which images you need, and their required size. You only need one Splash Screen image, and ownBrander will automatically resize and crop it for different-sized screens. You must also select a background color, which ensures that the splash screen image is always at the correct size ratio. (Click the example images on the right to enlarge them.)



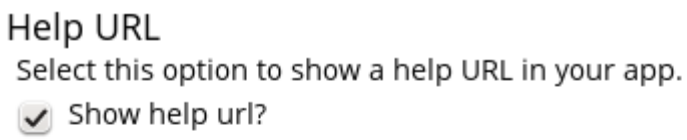
You may enter a custom **User agent**, which is useful for traffic analysis and whitelisting your app.

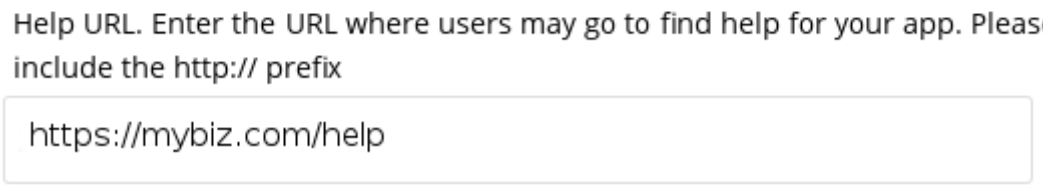


Check **Recommend** to open a Twitter, Facebook, and Email recommendation configurator.

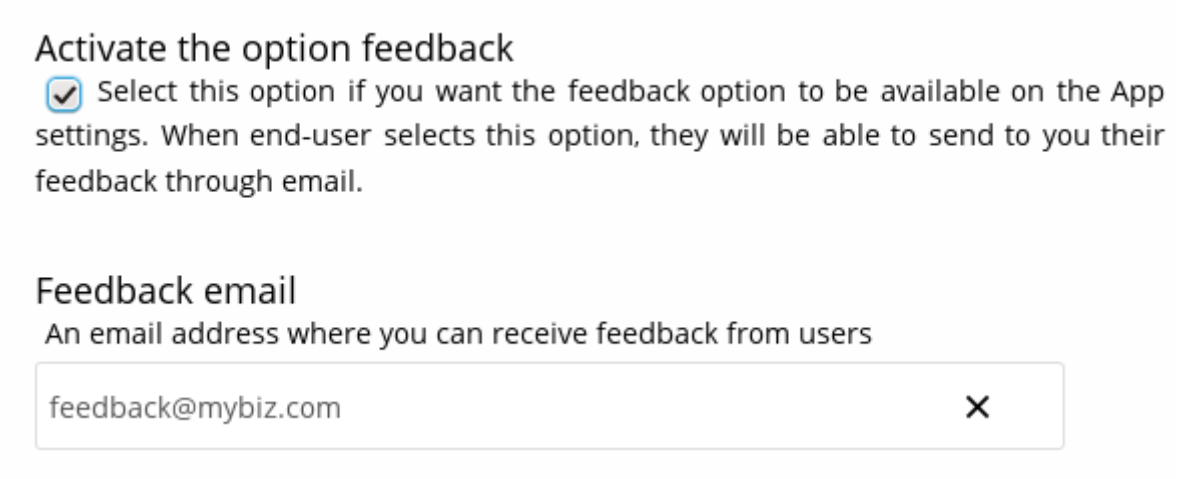


If you have online help, enter the URL here.

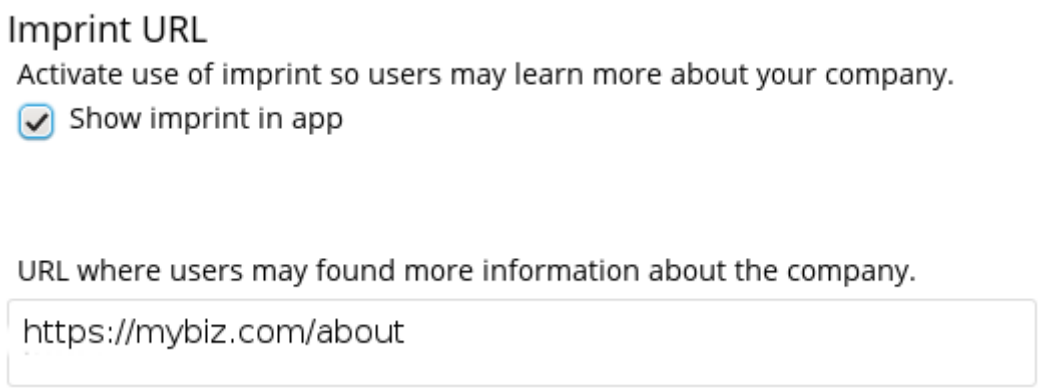




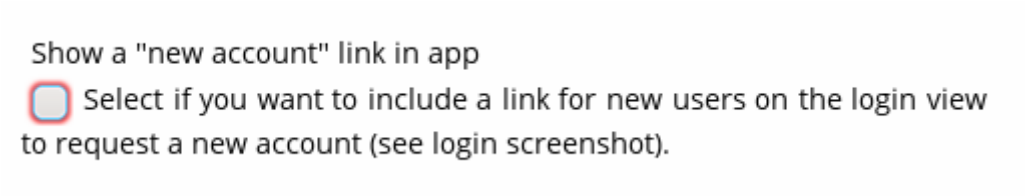
Activate the option feedback creates an option for your users to either enable or not enable the feedback option on their devices. If you enable this, enter your **Feedback email** address.



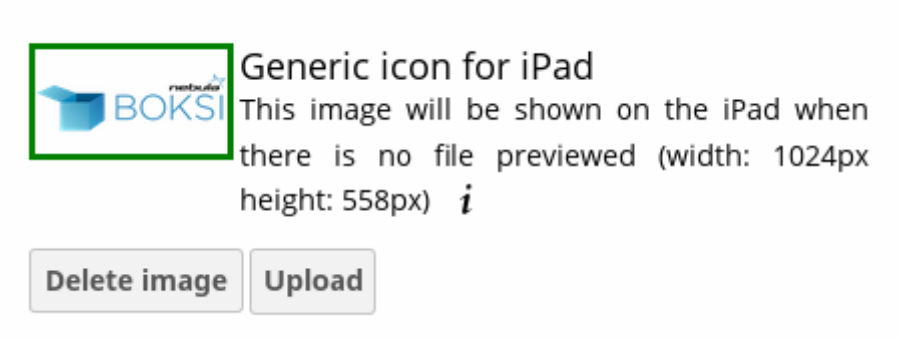
Enter your **Imprint URL** (your "about" page)



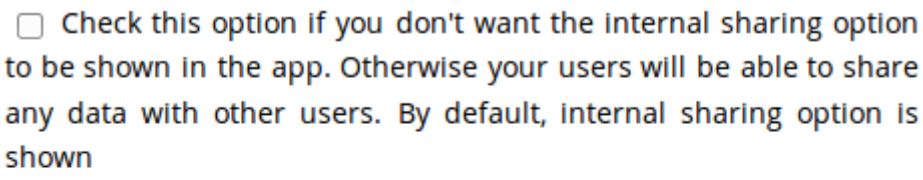
Check **Show a "new account" link in app** to allow new users to request a new account.

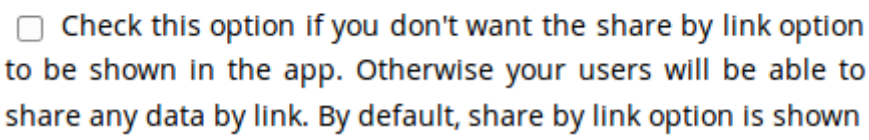


Upload an icon that will be displayed by default when there is no file preview to display.

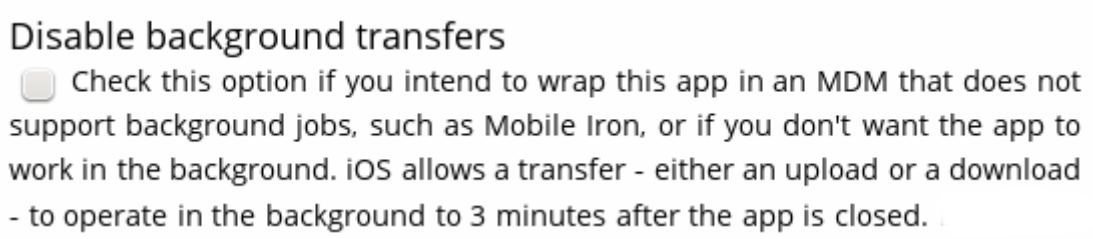


By default, both internal sharing and sharing by link are enabled. You have the options to disable one or both of these.

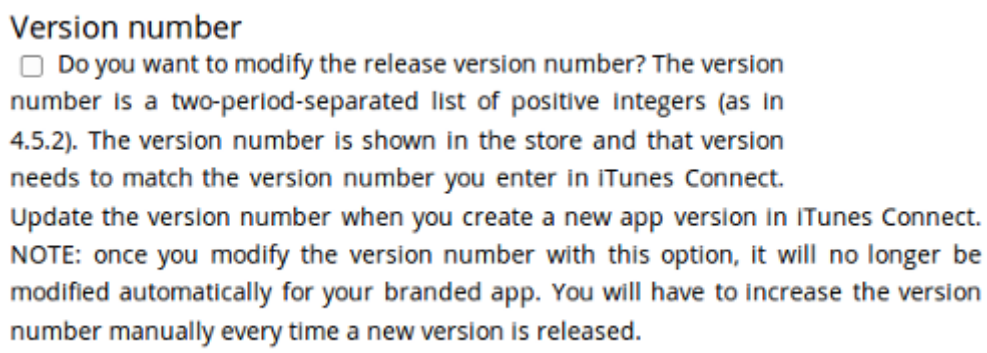




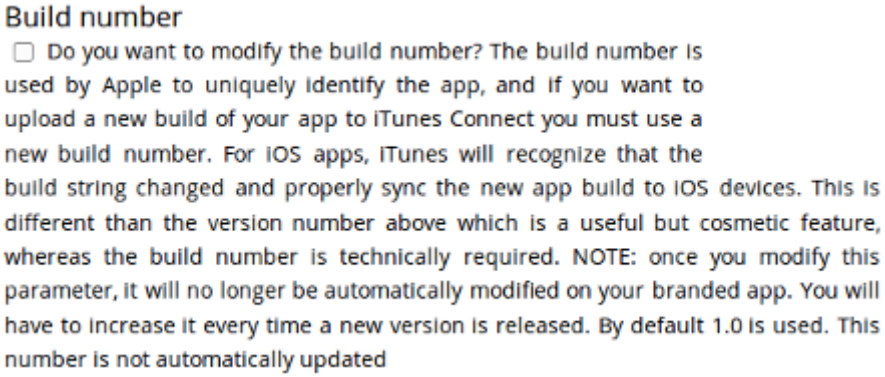
You may disable background transfers if you are using mobile device management (MDM), such as Mobile Iron, that does not support background jobs, or if you simply do not want to allow the app to work in the background. By default, the ownCloud iOS app supports background file transfers by taking advantage of Background Execution.



The default version number of your branded app is the same as the official ownCloud app. You have the option to customize your version number. Once you do this, you will have to update it manually for new releases. This must be the same as the version number that you enter in iTunes. Your version number is visible to your users.



You may also customize the build number, which defaults to 1.0.0. This must also be manually updated when you customize it. Your build number is used by iTunes to uniquely identify your app. When the build number changes, iTunes automatically syncs the updates for your users. The build number is not visible to your users.



That completes the required elements of your branded iOS app.

Suggested Section

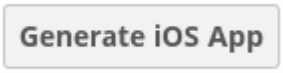
The Suggested section allows you to customize additional elements such as text and background colors, and icons. The Suggested items are all optional.

Advanced Section

The Advanced section allows you to optionally customize the color of messages such as connection status, error messages, letter separators, buttons, and additional icons.

Generate iOS App

When you have uploaded all of your images and completed your customizations, click the **Generate iOS App** button and take a well-deserved break. Remember to email your four Ad Hoc provisioning profiles and P12 certificate to support@owncloud.com.



You may go back and make changes, and when you click the **Generate iOS App** button the build system will use your latest changes.

Check your account on customer.owncloud.com in 48 hours to see your new branded ownCloud app.

Testing Your New Branded iOS App

Distribute the File

You'll distribute the file with the .ipa extension, like our example MyBiz iOS App-3.4.211.ipa, from your https://customer.owncloud.com/owncloud account to your beta testers. To do this you'll need a Mac computer, an iPhone or iPad registered in your Apple developer account, and the iTunes account associated with your Apple developer account.

1. Connect your registered iPhone or iPad to a Mac running iTunes.

2. Double-click your iOS .ipa file.
3. You should see your device in the upper left corner of your iTunes windows. Click on it.
4. Click the **[Apps]** button. Now you should see your app in the iTune apps list, with an Install button. Click it.
5. The Install button changes to Will Install.
6. Click the **[sync]** button in the lower-right corner to sync your device. This installs your app on your device.

Your other testers can now install and test your app on their registered iPhones and iPads just like any other app from iTunes.

If you have the Enterprise Apple developer account, there is no limit on the number of testing devices, and they do not have to be registered.

Getting Crash Reports From Testers

iOS automatically records crash logs when apps crash. Your testers can retrieve and send these logs to you. They must follow these steps:

1. Connect the testing device to a Mac computer running iTunes.
2. The crash logs are automatically downloaded to `~/Library/Logs/CrashReporter/MobileDevice`
3. Attach the relevant log files to email and send them to you.

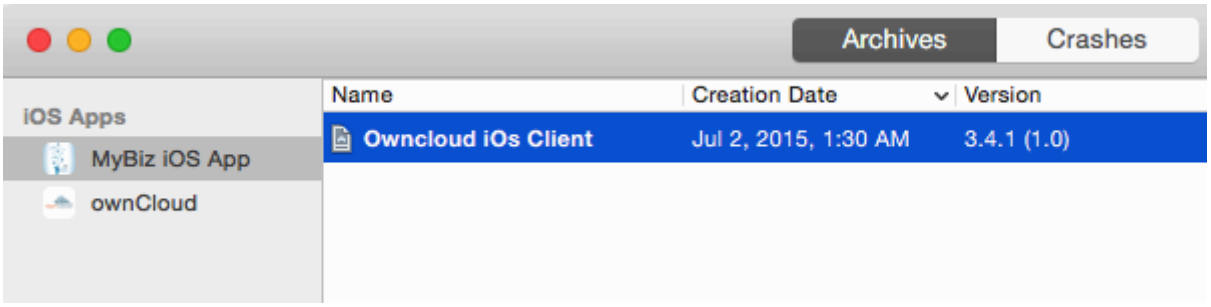
Publishing Your New Branded iOS App

Publish for General Distribution on iTunes

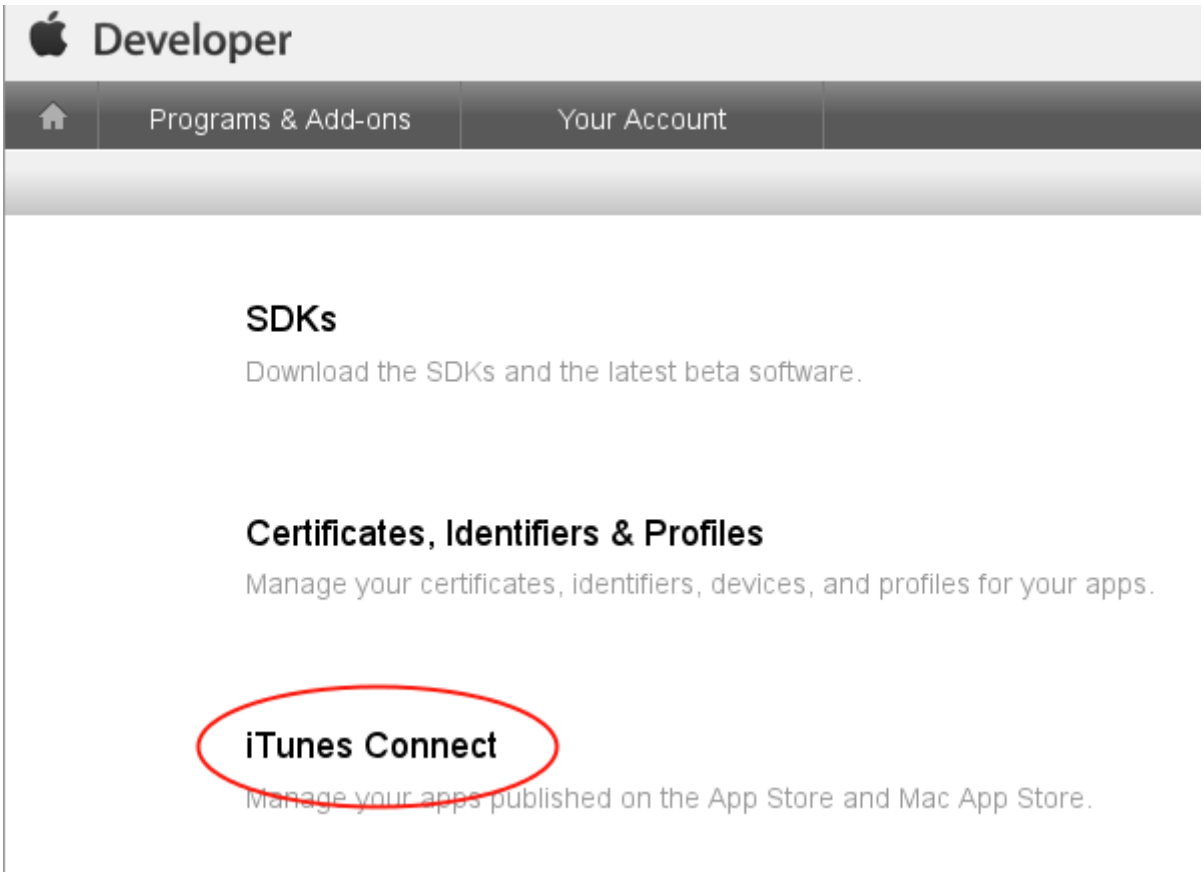
At last, after following all the previous steps and passing beta testing, your branded iOS app is ready to publish for general distribution on iTunes. You need a Mac computer with Xcode installed (Xcode is a free download), and you need the eight provisioning profiles (4 Ad Hoc and 4 Apple Store) and p12 file that you created copied to the same computer that you are using to upload your app to iTunes. You will also need a number of screenshots of your app in specific sizes and resolutions, which are detailed in your iTunes Connect setup screen.

Apple must review and approve your app, and the approval process can take several days to several weeks.

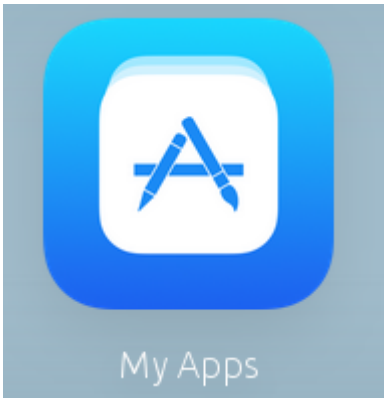
Download the `xcarchive.zip` file from your account. Your friendly macOS computer will automatically unpack it and change the name to something like `ownCloud iOS Client 02-07-15 10.30.xcarchive`. Double-click on this file to automatically install it into Xcode. Go to Xcode and you will see it in the Archives listing under **Window › Organizer**.



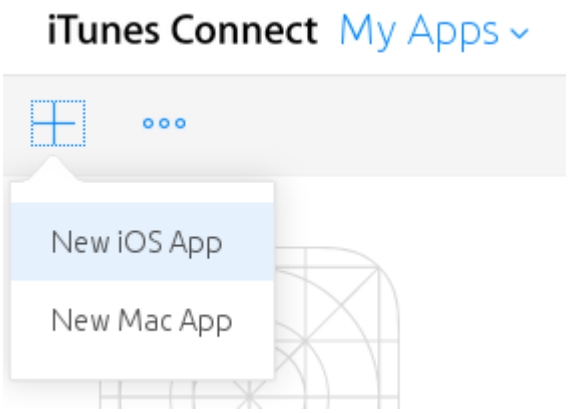
Next, go back to the Apple Developer Member Center to log into iTunes Connect to set up your app storefront.



After logging in click the blue **[My Apps]** button. This takes you to the main screen for managing your apps on iTunes.



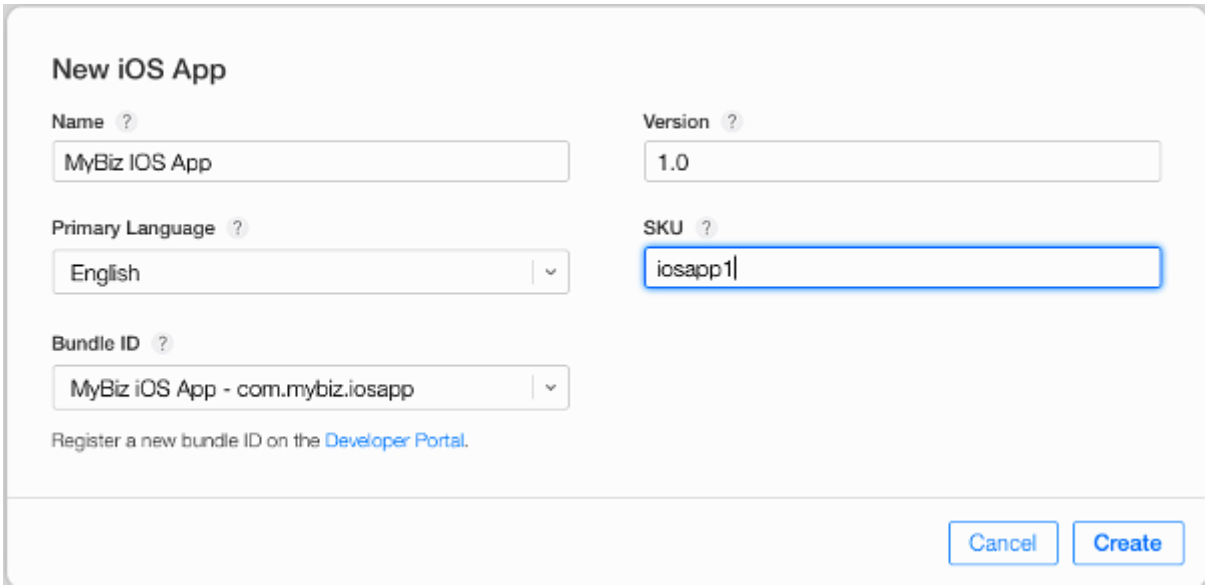
Click the plus button on the top left to setup your new branded iOS app.



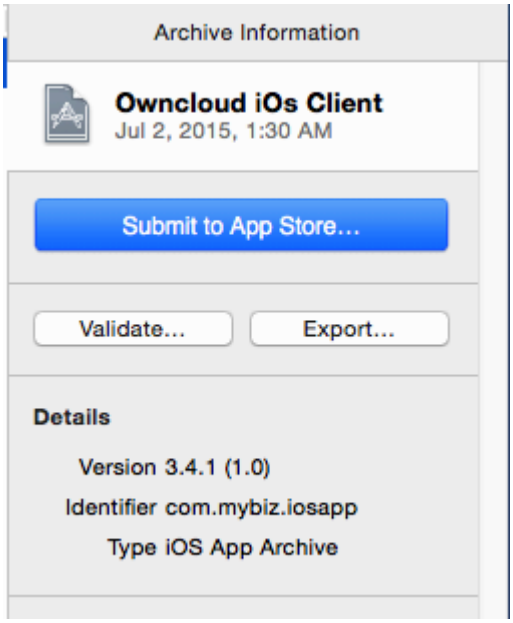
This opens a screen where you will enter your app information. Make sure you get it right the first time, because it is difficult to delete apps, and Apple will not let you re-use your app name or SKU.

- Enter any name you want for your app. This is the name that will appear in your App Store listing.
- Choose your primary language.
- Select the bundle ID from the drop-down selector.
- Enter your app version number, which should match the version number as it appears in your Xcode organizer.
- The SKU is a unique ID for your app, and is anything you want.

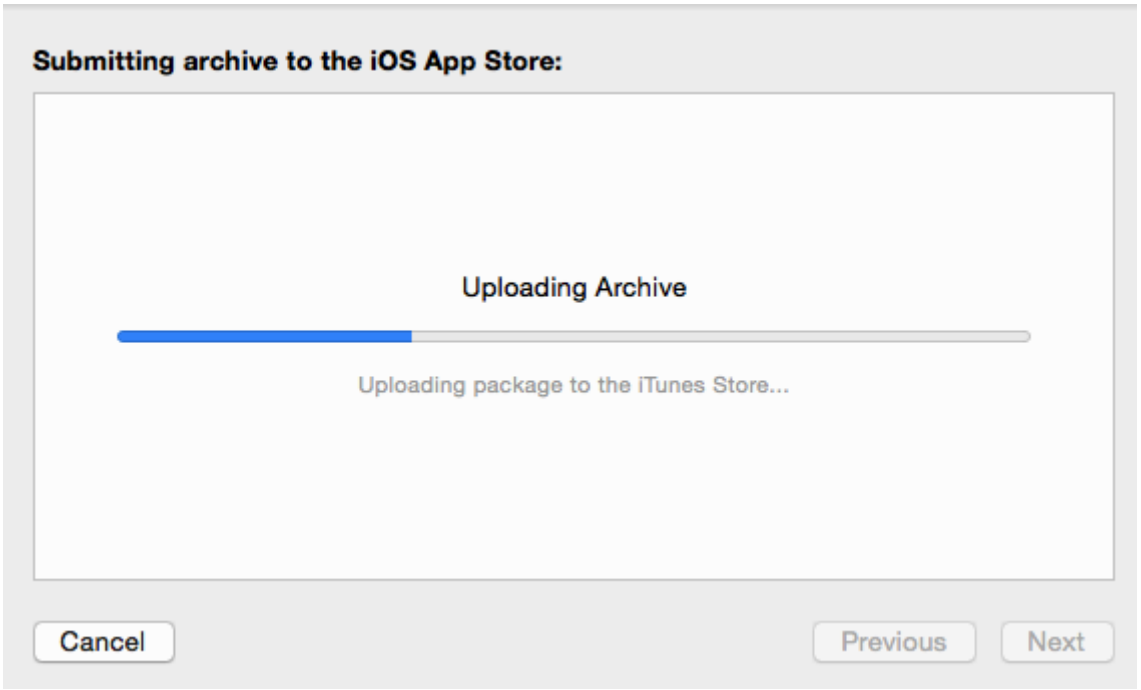
Then click the **[Create]** button.



Now go back to your Xcode organizer to upload your app; click the blue **[Submit to App Store]** button.



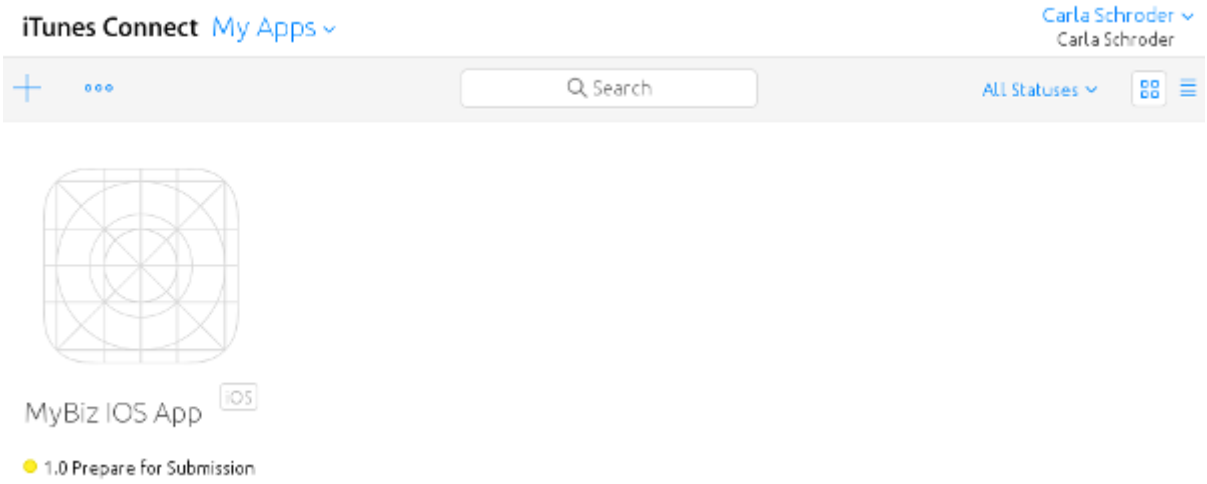
This takes a few minutes as it verifies your bundle ID and certificates, and then you will see an upload status.



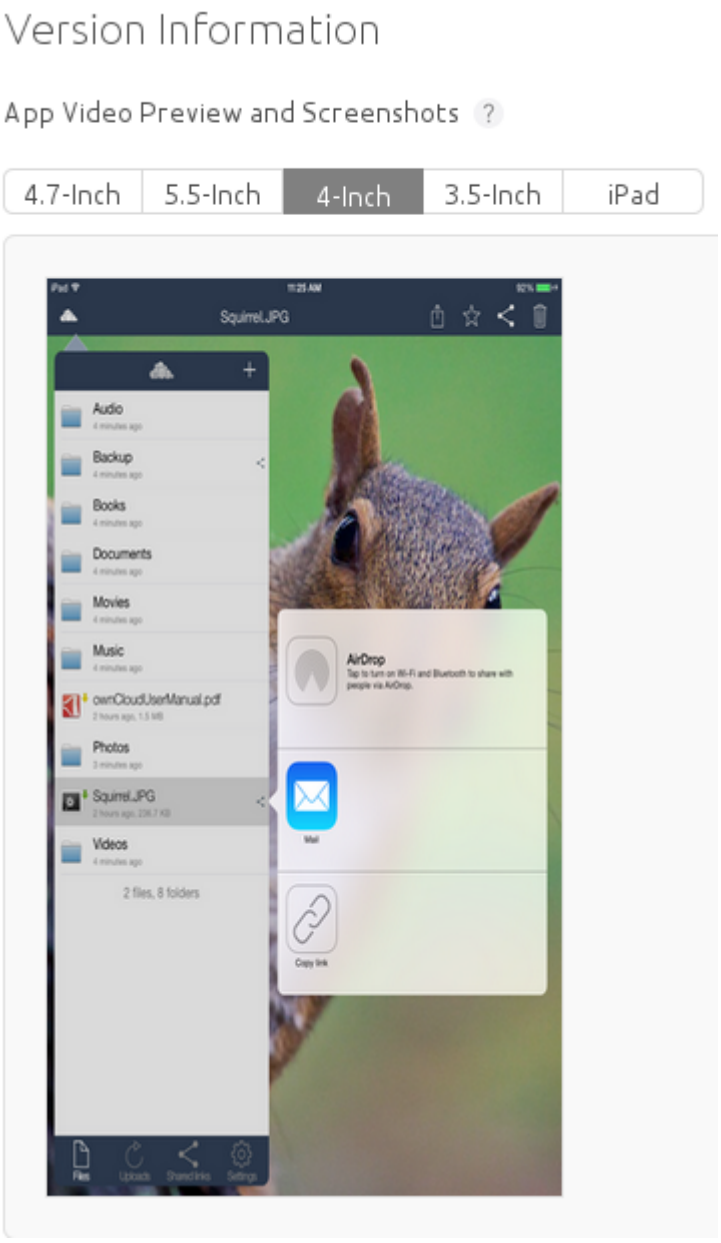
At long last, after working through this long complex process, you are almost ready to publish your app on iTunes.

Setting up Your iTunes Storefront

There are just a few steps remaining. Now that you have uploaded your branded iOS app, you need to upload some screenshots, an optional demo video, and fill in some information for your app listing on your iTunes storefront. You should see something like this on your main screen (figure 8). You should click the **[Save]** button at the top right periodically to preserve your changes.



This screen displays all of your apps and their submission status. Click **[Prepare for Submission]** to get started on the submission process. The first screen is for entering screenshots of your app for various devices, and optionally a demonstration video. Click the little question marks to learn the required image specifications.



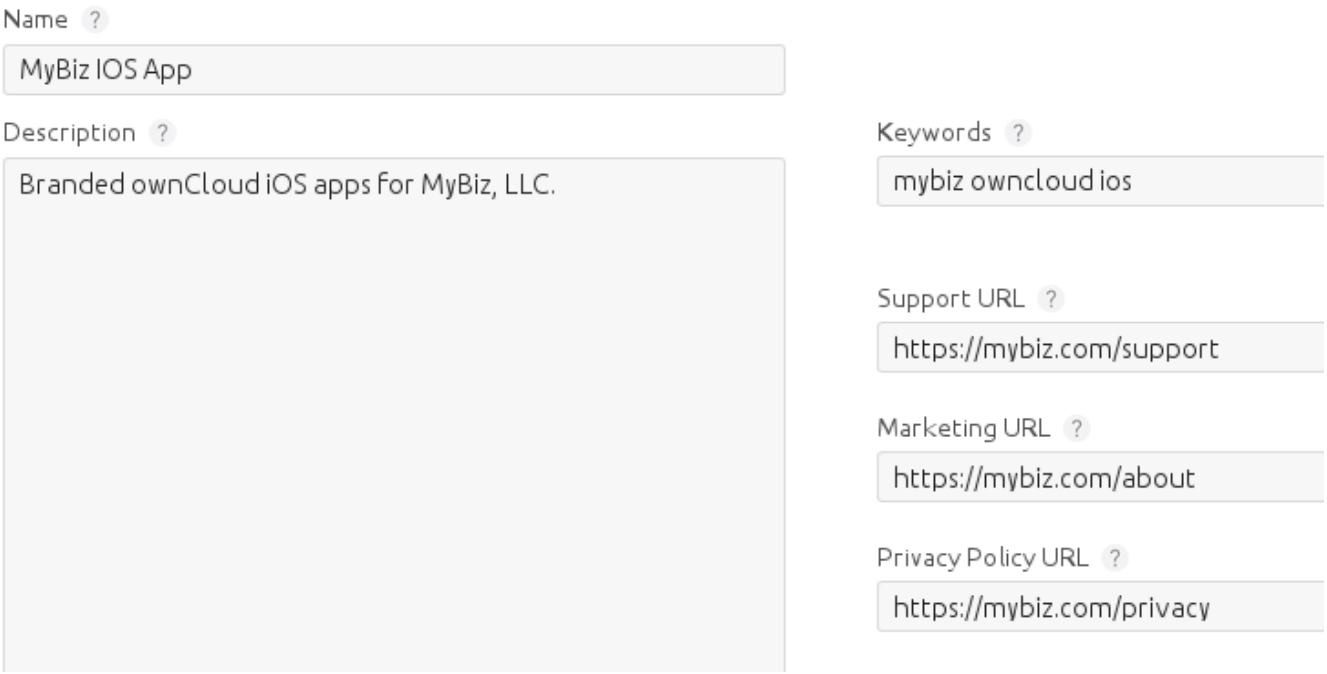
Apple simplified the screenshot submission process. Please check this Video (in Safari) for details.

For the ownCloud client, we also don't use real screenshots, we use frames in different sizes instead. You can find templates to generate those assets. Here are examples for the Sketch app:

- https://github.com/LaunchKit/SketchToAppStore
- https://github.com/MengTo/AppStoreSketch

Then you must enter:

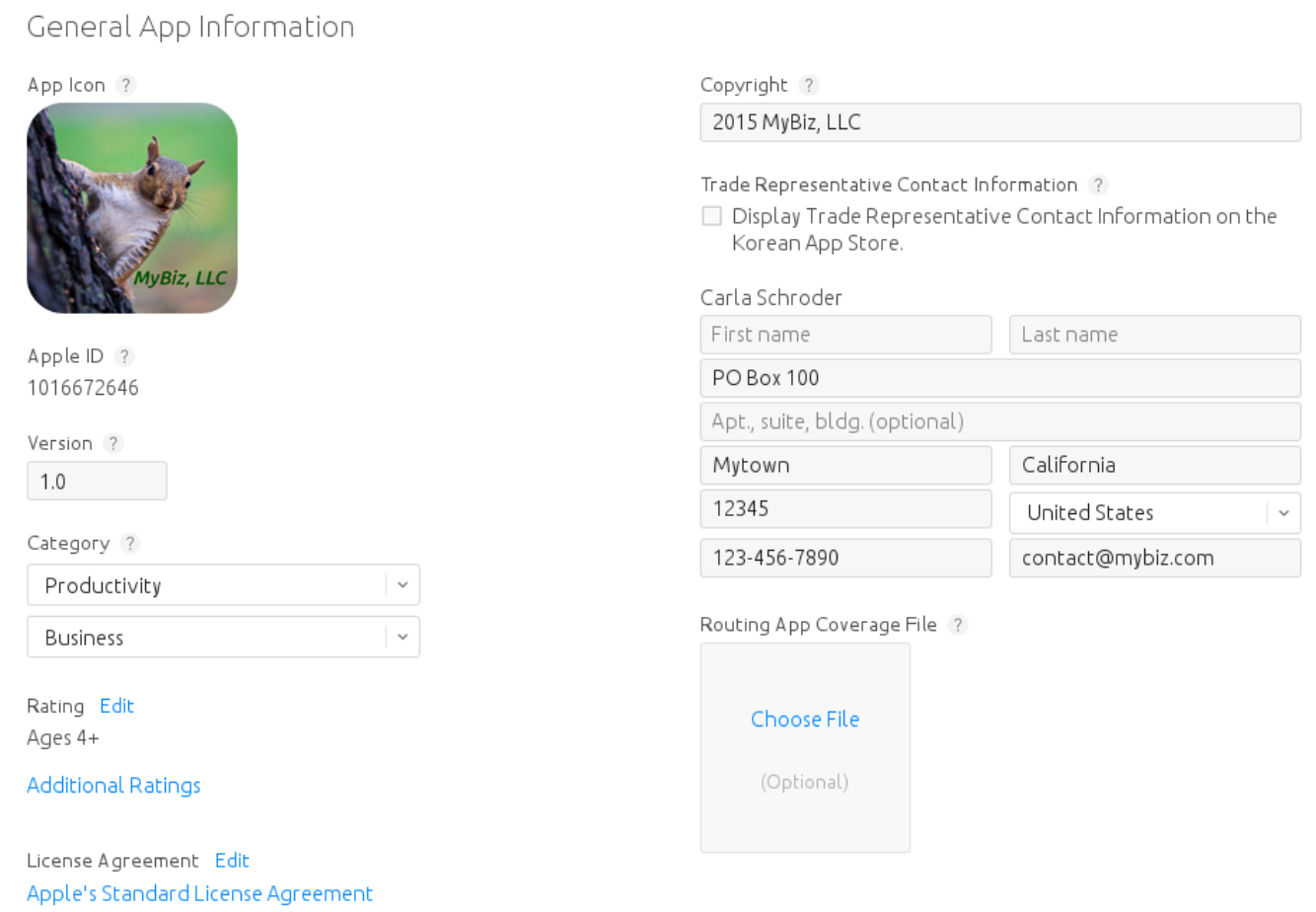
- Your app name
- A description
- Some keywords for iTunes searches; and
- Some optional URLs



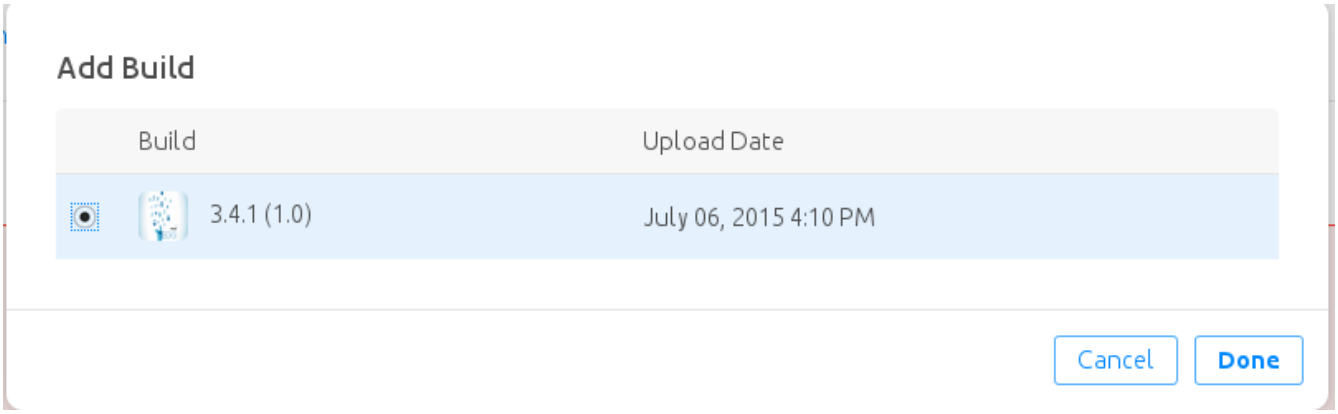
The next section is for Apple Watch. If you don't support Apple Watch you can skip this.

The **General App Information** section requires a:

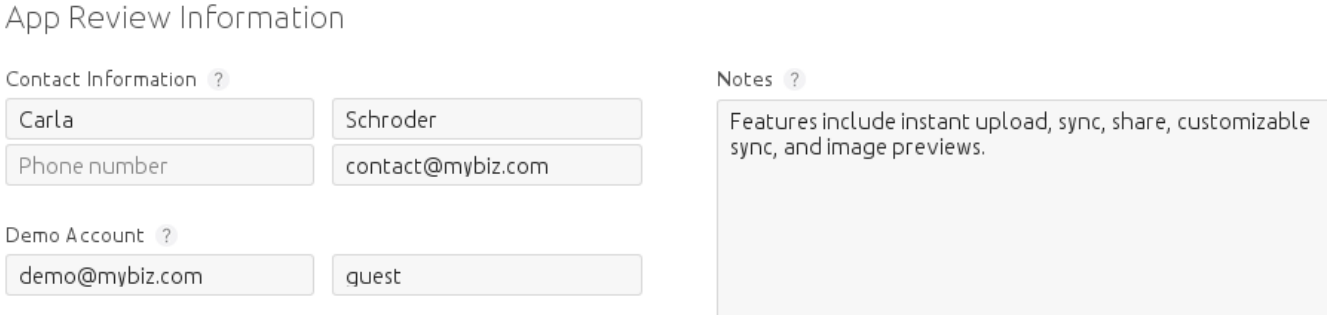
- 1024 x 1024 logo
- Version
- Rating
- Category
- License
- Copyright, and
- Contact information



In the **Build** section, click the plus button and select your app.



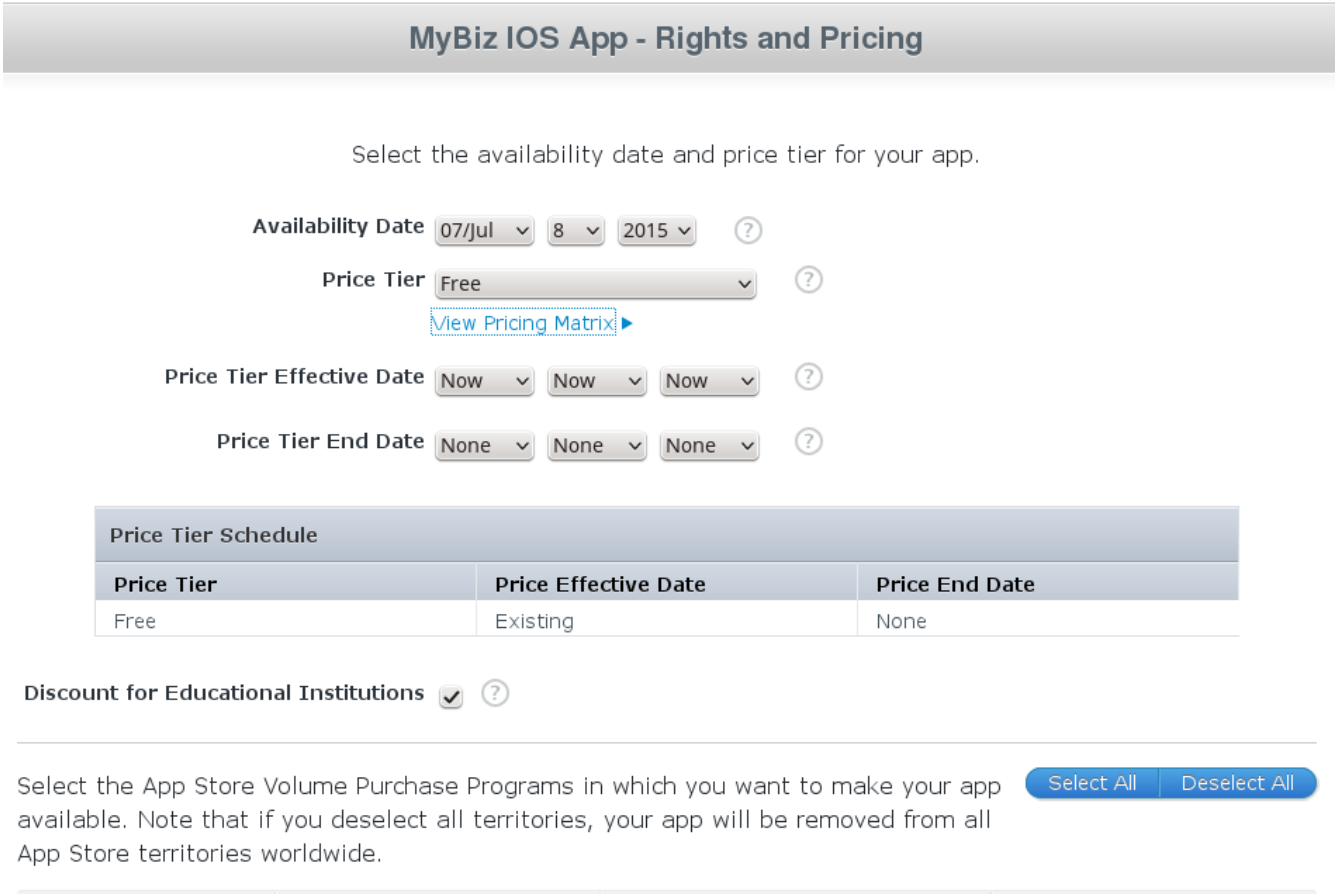
The **App Review Information** requires contact information, and some information about your app to guide reviewers. Remember, everyone on iTunes can review your app, so it's in your best interest to be helpful. You may optionally provide a login for a demo account.



The **Version Release** section allows you to choose between automatic release, which means your app will be published upon approval, or manual release, where you must release your app after it is approved.

Pricing

Next, you must go to the **Pricing** page to set your price, and to select the territories you want your app to be available in.



Submit For Review

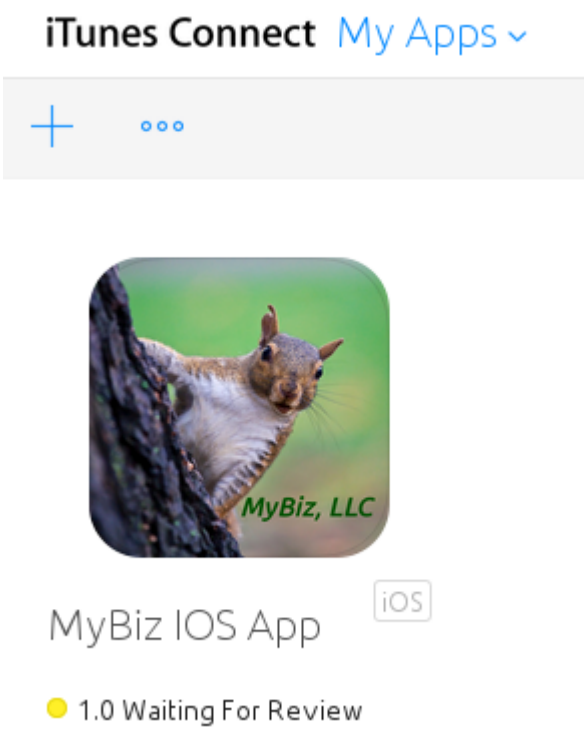
When you have filled in all the required forms and provided the required screenshots, click **Save** and then **Submit for Review**. If anything needs to be corrected you will see messages telling you exactly what must be fixed.

The next screen is legalese; click the appropriate Yes or No boxes, and then click the **Submit** button.

You are now finished. No really, you are. When you return to your **My Apps** page you'll see that the status of your app has changed to "Waiting for review". In a few days, or perhaps many days, your app will either be approved or rejected. If it is rejected Apple will tell you what you need to do to get it approved.

FAQ

Here are the most common answers to questions from the iOS App Review Team.



When, at last, it is published on iTunes you may distribute the URL so that your users may install and use your app.

Additional Server Configuration

Add Support for Apple Universal Links

What is Universal Links?

When you support universal links, iOS users can tap a link to your website and be seamlessly redirected to your installed app, without going through Safari. If your app isn't installed, tapping a link to your website opens your website in Safari. For more details, see Support Universal Links.

There's some special changes that need to be made. Quoting from Apple's official documentation on Universal Link Support:

> Adding support for universal links is easy. There are three steps you need to take:
>
> 1. Create an apple-app-site-association file that contains JSON data about the URLs that your app can handle.
> 2. Upload the apple-app-site-association file to your HTTPS web server. You can place the file at the root of your server or in the `.well-known` subdirectory.
> 3. Prepare your app to handle universal links.

The `apple-app-site-association` data is generated by ownBrander and must be served statically over HTTPS.

You can safely place it in the root folder of your ownCloud installation (e.g., `/var/www/owncloud`).

What is apple-app-site-association?

The `apple-app-site-association` directory is either a subdirectory of your ownCloud URL or of the `/.well-known/` directory, and **must** be served over HTTPS. Data generated by ownBrander is accessed via this subdirectory.

The name "apple-app-site-association" is mandatory.

The file which gets accessed when using this subdirectory is also named `apple-app-site-association` without any extension. When this subdirectory is accessed, the web server must set the content type to `application/json`. The physical path used when accessing this directory must be defined in your web server config. In the examples below, the path for the file is `/var/www/owncloud/`.

When using a physical path to the file inside your ownCloud directory, this file must be present when you upgrade ownCloud, or you chose a different path outside the ownCloud root.

Apache Configuration

To achieve the second requirement, some changes will also need to be made to your Apache configuration. If you configured your installation with the official Admin Manual, your Apache `owncloud.conf` file must include the following:

```
# Create an alias for the file (for compatibility reasons):
AliasMatch "^/(\.well-known/)?apple-app-site-association$" "/var/www/owncloud/apple-
app-site-association"

<Directory /var/www/owncloud/>
    Options +FollowSymlinks
    AllowOverride All

    # Set the right mime-type for the file:
    <Files apple-app-site-association>
        Header set Content-type "application/json"
    </Files>

    [...]
</Directory>
```

See the AliasMatch documentation for more details

Also, a new RewriteCond directive, included in the code block below, needs to be included in your `.htaccess` (or VirtualHost configuration), so that no other redirections will apply to any of these two paths.

```
RewriteCond %\{REQUEST_URI} !^/(.well-known/)?apple-app-site-association$
```

NGINX Configuration

For NGINX, the directives to be added are:

```
location ~* ^/(\.well-known/)?apple-app-site-association {
    # uncomment and update the root configuration line below,
    # in case the path to your ownCloud installation is in a different location.
    # root '/var/www/owncloud';
    default_type 'application/json';
}
```

If you're behind a firewall, additional access rules will be required to whitelist the URLs.

FAQ iOS App Review Team

Information from Apple: https://developer.apple.com/support/app-review/

The product contains cryptography, and whether it classifies for export exemptions.

No, the product does not contain cryptography. Although the app is ready to connect via SSL, this does not imply that the app includes any cryptography

How does the app utilize Document Picker and File Provider extensions?

The ownCloud app takes advantage of the Document Provider extensions so that those apps that act as Document Picker may access to the ownCloud data, edit it and then changes are automatically uploaded back to the ownCloud server.

Background Audio

Questions:

- What is the purpose of declaring Audio background mode? Please explain the need for this background mode and where the usage can be found in your binary.
- Your app declares support for audio in the UIBackgroundModes key in your Info.plist but did not include features that require persistent audio. The audio key is intended for use by applications that provide audible content to the user while in the background, such as music player or streaming audio applications. Please revise your app to provide audible content to the

user while the app is in the background or remove the "audio" setting from the UIBackgroundModes key.

Answer:

Sometimes, usually, the first time the ownCloud app is submitted, it is rejected because it is included the background mode, Apple rejected it because in the past some apps used this trick to avoid the app to be fully closed. Howerver, the ownCloud app used it only when music is played. This may be checked by Apple reviewers, what we suggest is to be proactive, instead of waiting for the app to be rejected because of that, adding an explanation line, something such as: You may notice that the app is ready to play music not only in foreground but also in background, for you to test it we have uploaded to the test account the file XXX

Content Rights - Does your app contain, display, or access third-party content?

If the branded app has the help option enable, the answer is yes. Within the help, we are having access to an external web Otherwise, no

Does this app use the Advertising Identifier (IDFA)?

No, no ads at all

IPv6 Connectivity

Question:

> We discovered one or more bugs in your app when reviewed on the iPad and the iPhone running iOS 10.2 on Wi-Fi connected to an IPv6 network - Specifically, the app does not connect to the server.

Information from Apple: https://developer.apple.com/library/content/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/UnderstandingandPreparingfortheIPv6Transition/UnderstandingandPreparingfortheIPv6Transition.html

Here you can check your server for IPv6 connectivity: http://ipv6-test.com/validate.php

Business questions from Apple

- Does your app access any paid content or services?
- What are the paid content or services, and what are the costs?
- Who pays for the content or services?
- Where do they pay, and what's the payment method?
- If users create an account to use your app, are there fees involved?
- How do users obtain an account?

This is a standard question Apple has to avoid iTunes circumvention as for some stuff they want the 30% revenue share. (see In-App Purchase: https://developer.apple.com/in-app-purchase/)

[1] https://developer.android.com/guide/app-bundle

[2] https://android-developers.googleblog.com/2021/06/the-future-of-android-app-bundles-is.html