TestFlight, Provisioning Profiles, and the Mac App Store

This thread has been locked by a moderator.

604

I regularly come across Mac developers who have an app in the Mac App Store but are unable to submit it to TestFlight. This post explains a common cause of that problem.

If you have any questions or comments about this, start a new thread and tag it with *Provisioning Profiles* and *TestFlight* so that I see it.

Share and Enjoy

—

Quinn “The Eskimo!” @ Developer Technical Support @ Apple

```
let myEmail = "eskimo" + "1" + "@" + "apple.com"
```

TestFlight, Provisioning Profiles, and the Mac App Store

A provisioning profile authorises a device to run your app. For historical reasons, not all Mac apps need a provisioning profile. A Mac app only needs a profile if it uses a **restricted entitlement**, that is, an entitlement that must be authorised by a profile. For more background on this, see TN3125 Inside Code Signing: Provisioning Profiles and, specifically, its *Entitlements on macOS* section.

IMPORTANT Your Mac App Store apps must be signed with the App Sandbox Entitlement, but that entitlement is unrestricted.

This means that many Mac App Store apps ship without a provisioning profile, and that’s absolutely fine. However, these apps run into problems with TestFlight. To submit an app to TestFlight, it must have a provisioning profile.

If you attempt to submit an app without a profile to TestFlight, it’ll fail with an error like this:

ITMS-90889: Cannot be used with TestFlight because the bundle at ‘MyApp.app’ is missing a provisioning profile. Main bundles are expected to have provisioning profiles in order to be eligible for TestFlight.

The fix is to give your app a profile. How you do this depends on how you build your app.

Fix an app built with Xcode

If you build your app with Xcode, the fix is relatively straightforward: Sign your app with a restricted entitlement. This causes Xcode’s code signing machinery to kick in. If you have automatic code signing enable, Xcode will sort this all out for you. If you use manual signing, Xcode will highlight the problems you need to solve.

A good restricted entitlement to use is the Keychain Access Groups Entitlement. Enable this by adding Keychain Sharing to the Signing & Capabilities editor for your app. You have two options here:

- Leave the Keychain Groups list empty. This will fix this problem while having no effect on any keychain code in your app.
- Use this as an opportunity to switch to the data protection keychain. In this case you might want to add one or more keychain access groups.

For an explanation as to why you might want to switch to using the data protection keychain, see TN3137 On Mac keychain APIs and implementations. For more information about keychain access groups, see Sharing Access to Keychain Items Among a Collection of Apps

Fix an app built outside of Xcode

If you don’t use Xcode to build your app:

1. Use Developer > Account > Identifiers to create an App ID for your app. Remember that your App ID is the combination of an App ID prefix and your app’s bundle ID. For new App IDs, use your Team ID as the App ID prefix.
2. Use Developer > Account > Profiles to create a macOS App Development provisioning profile for that App ID.
3. Use Developer > Account > Profiles to create a Mac App Store distribution provisioning profile for that App ID.
4. Update your build system to embed a provisioning profile into your app. Use the profile from step 2 for development-signed builds and the one from step 3 for distribution-signed builds. For information about where to place the profile, see Placing Content in a Bundle.
5. Add the following to your `.entitlements`:
   - A `com.apple.application-identifier` property whose value is your App ID
   - A `com.apple.developer.team-identifier` property whose value is your Team ID
6. Build your app and check your work by dumping the entitlements claimed by your app and the entitlements authorised by your provisioning profile. For the specific commands to use, see TN3125 Inside Code Signing: Provisioning Profiles.

WARNING In step 1, if your team has any unique App ID prefixes registered, the Developer website might default to using one of those legacy values rather than your Team ID (r. 70571514). If the App ID Prefix value is a popup, select your Team ID from the list. If the App ID value is a read-only copy of your Team ID, your team has no unique App ID prefixes, and so the Developer website always uses your Team ID as the App ID prefix.

IMPORTANT In step 5, make sure that your `.entitlements` file is only applied to the app itself, not to any nested code. For more on this, see the *Entitlements and Nested Code* section below.

Historically you might have been able to get away with using single `.entitlements` file for all your code. Once you start adding restricted entitlements, like `com.apple.application-identifier`, this bad practice will cause problems.

For general information about how to sign and package apps outside of Xcode, see Creating Distribution-Signed Code for Mac and Packaging Mac Software for Distribution.

Entitlements and Nested Code

An App Store app might contain the following code;

- The app itself
- Nested libraries, like a framework or a dynamic library
- Nested executables, like a helper tool or an app extension

Step 5 in the previous section specifically refers to the entitlements of the main app. When it comes to nested code, the first case is easy: **Never add entitlements to nested libraries.** It doesn’t do anything useful and can prevent your code from running.

The story with nested executables is more nuanced. To start, every nested executable must be signed with at least one entitlement because:

- All App Store executables must be sandboxed.
- You enable the App Sandbox with the `com.apple.security.app-sandbox` entitlement.

In many cases a nested executable only needs unrestricted entitlements, like `com.apple.security.app-sandbox` and `com.apple.security.inherit`. In that case the nested code doesn’t need a provisioning profile.

If a nested executable uses restricted entitlements, it needs a provisioning profile to authorise the use of those entitlements, and its own unique App ID to tie the executable to the profile. Place this profile in the nested executable’s bundle, according to the rules in Placing Content in a Bundle.

IMPORTANT The nested code can’t ‘piggyback’ off the app’s provisioning profile. It needs its own profile with its own unique App ID.

Revision History

- **2023-08-17** Added the *Entitlements and Nested Code* section. Made other minor editorial changes.
- **2023-07-17** First posted.

Provisioning Profiles TestFlight

Reply

Posted 3 months ago by eskimo

Add a Comment