Build and release an iOS app

Contents

This guide provides a step-by-step walkthrough of releasing a Flutter app to the App Store and TestFlight.

Preliminaries

Xcode is required to build and release your app. You must use a device running macOS to follow this guide.

Before beginning the process of releasing your app, ensure that it meets Apple's App Review Guidelines.

In order to publish your app to the App Store, you must first enroll in the Apple Developer Program. You can read more about the various membership options in Apple's Choosing a Membership guide.

Register your app on App Store Connect

Manage your app's life cycle on App Store Connect (formerly iTunes Connect). You define your app name and description, add screenshots, set pricing, and manage releases to the App Store and TestFlight.

Registering your app involves two steps: registering a unique Bundle ID, and creating an application record on App Store Connect.

For a detailed overview of App Store Connect, see the App Store Connect guide.

Register a Bundle ID

Every iOS application is associated with a Bundle ID, a unique identifier registered with Apple. To register a Bundle ID for your app, follow these steps:

1. Open the App IDs page of your developer account.
2. Click + to create a new Bundle ID.
3. Enter an app name, select **Explicit App ID**, and enter an ID.
4. Select the services your app uses, then click **Continue**.
5. On the next page, confirm the details and click **Register** to register your Bundle ID.

Create an application record on App Store Connect

Register your app on App Store Connect:

1. Open App Store Connect in your browser.
2. On the App Store Connect landing page, click **My Apps**.
3. Click + in the top–left corner of the My Apps page, then select **New App**.
4. Fill in your app details in the form that appears. In the Platforms section, ensure that iOS is checked. Since Flutter does not currently support tvOS, leave that checkbox unchecked. Click **Create**.
5. Navigate to the application details for your app and select **App Information** from the sidebar.
6. In the General Information section, select the Bundle ID you registered in the preceding step.

For a detailed overview, see Add an app to your account.

Review Xcode project settings

This step covers reviewing the most important settings in the Xcode workspace. For detailed procedures and descriptions, seePrepare for app distribution.

Navigate to your target's settings in Xcode:

1. In Xcode, open `Runner.xcworkspace` in your app's `ios` folder.
2. To view your app's settings, select the **Runner** project in the Xcode project navigator. Then, in the main view sidebar, select the **Runner** target.
3. Select the **General** tab.

Verify the most important settings.

In the **Identity** section:

`Display Name`
The display name of your app.

`Bundle Identifier`
The App ID you registered on App Store Connect.

In the **Signing & Capabilities** section:

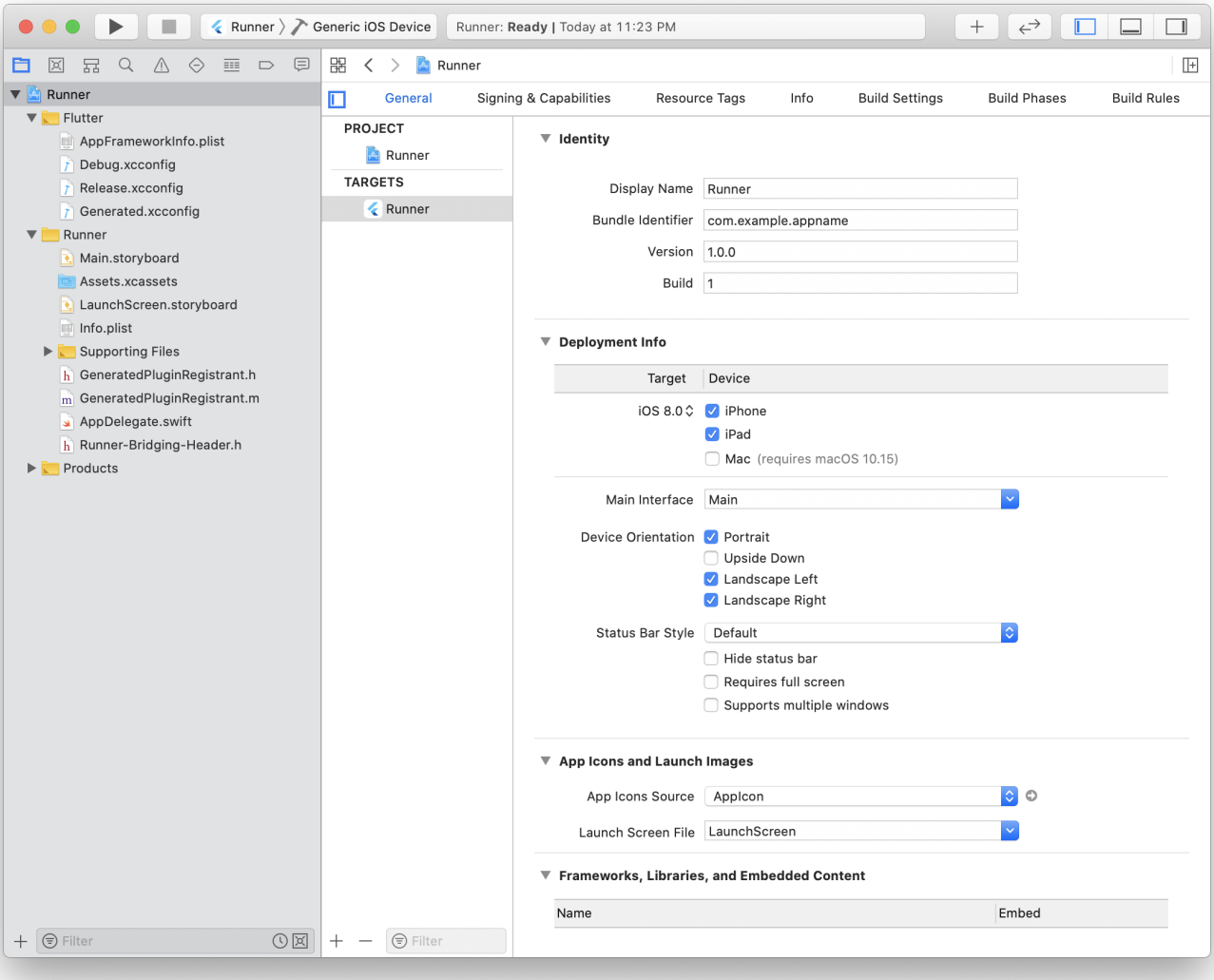
`Automatically manage signing`
Whether Xcode should automatically manage app signing and provisioning. This is set `true` by default, which should be sufficient for most apps. For more complex scenarios, see the Code Signing Guide.

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

In the **Build Settings** section:

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

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



For a detailed overview of app signing, see Create, export, and delete signing certificates.

Updating the app's deployment version

If you changed `Deployment Target` in your Xcode project, open `ios/Flutter/AppframeworkInfo.plist` in your Flutter app and update the `MinimumOSVersion` value to match.

Updating the app's version number

The default version number of the app is `1.0.0`. To update it, navigate to the `pubspec.yaml` file and update the following line:

`version: 1.0.0+1`

The version number is three numbers separated by dots, such as `1.0.0` in the example above, followed by an optional build number such as `1` in the example above, separated by a `+`.

Both the version and the build number may be overridden in Flutter's build by specifying `--build-name` and `--build-number`, respectively.

In iOS, `build-name` uses `CFBundleShortVersionString` while `build-number` uses `CFBundleVersion`. Read more about iOS versioning at Core Foundation Keys on the Apple Developer's site.

Add an app icon

When a new Flutter app is created, a placeholder icon set is created. This step covers replacing these placeholder icons with your app's icons:

1. Review the iOS App Icon guidelines.
2. In the Xcode project navigator, select `Assets.xcassets` in the `Runner` folder. Update the placeholder icons with your own app icons.
3. Verify the icon has been replaced by running your app using `flutter run`.

Create a build archive with Xcode

This step covers creating a build archive and uploading your build to App Store Connect.

During development, you've been building, debugging, and testing with *debug* builds. When you're ready to ship your app to users on the App Store or TestFlight, you need to prepare a *release* build. At this point, you might consider obfuscating your Dart code to make it more difficult to reverse engineer. Obfuscating your code involves adding a couple flags to your build command.

In Xcode, configure the app version and build:

1. In Xcode, open `Runner.xcworkspace` in your app's `ios` folder.
2. Select **Runner** in the Xcode project navigator, then select the **Runner** target in the settings view sidebar.
3. In the Identity section, update the **Version** to the user-facing version number you wish to publish.
4. In the Identity section, update the **Build** identifier to a unique build number used to track this build on App Store Connect. Each upload requires a unique build number.

Finally, create a build archive and upload it to App Store Connect:

1. Run `flutter build ipa` to produce a build archive.

Note: On versions of Flutter where `flutter build ipa` is unavailable, open Xcode and select **Product > Archive**. In the sidebar of the Xcode Organizer window, select your iOS app, then select the build archive you just produced.

2. Open `build/ios/archive/MyApp.xcarchive` in Xcode.

3. Click the **Validate App** button. If any issues are reported, address them and produce another build. You can reuse the same build ID until you upload an archive.

4. After the archive has been successfully validated, click **Distribute App**. You can follow the status of your build in the Activities tab of your app's details page on App Store Connect.

Note: When you export your app at the end of **Distribute App**, Xcode will create a directory containing an IPA of your app and an `ExportOptions.plist` file. You can create new IPAs with the same options without launching Xcode by running `flutter build ipa --export-options-plist=path/to/ExportOptions.plist`. See `xcodebuild -h` for details about the keys in this property list.

You should receive an email within 30 minutes notifying you that your build has been validated and is available to release to testers on TestFlight. At this point you can choose whether to release on TestFlight, or go ahead and release your app to the App Store.

For more details, see Upload an app to App Store Connect.

Create a build archive with Codemagic CLI tools

This step covers creating a build archive and uploading your build to App Store Connect using Flutter build commands and Codemagic CLI Tools executed in a terminal in the Flutter project directory. This allows you to create a build archive with full control of distribution certificates in a temporary keychain isolated from your login keychain.

1. Install the Codemagic CLI tools:

```
pip3 install codemagic-cli-tools
```
content_copy

2. You'll need to generate an App Store Connect API Key with App Manager access to automate operations with App Store Connect. To make subsequent commands more concise, set the following environment variables from the new key: issuer id, key id, and API key file.

```
export APP_STORE_CONNECT_ISSUER_ID=aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
export APP_STORE_CONNECT_KEY_IDENTIFIER=ABC1234567
export APP_STORE_CONNECT_PRIVATE_KEY=`cat
/path/to/api/key/AuthKey_XXXYYYZZZ.p8`
```
content_copy

3. You need to export or create an iOS Distribution certificate to code sign and package a build archive.

   If you have existing certificates, you can export the private keys by executing the following command for each certificate:

```
openssl pkcs12 -in <certificate_name>.p12 -nodes -nocerts -legacy -
out cert_key
```
content_copy

   Or you can create a new private key by executing the following command:

```
ssh-keygen -t rsa -b 2048 -m PEM -f cert_key -q -N ""
```
content_copy

   Later, you can have CLI tools automatically create a new iOS Distribution from the private key.

4. Set up a new temporary keychain to be used for code signing:

```
keychain initialize
```
content_copy

Restore Login Keychain! After running `keychain initialize` you **must** run the following:

```
keychain use-login
```

This sets your login keychain as the default to avoid potential authentication issues with apps on your machine.

5. Fetch the code signing files from App Store Connect:

```
app-store-connect fetch-signing-files YOUR.APP.BUNDLE_ID \
    --platform IOS_APP_STORE \
    --certificate-key=@file:/path/to/cert_key \
    --create
```

Where `cert_key` is either your exported iOS Distribution certificate private key or a new private key which automatically generates a new certificate. The certificate will be created from the private key if it doesn't exist in App Store Connect.

6. Now add the fetched certificates to your keychain:

```
keychain add-certificates
```

7. Update the Xcode project settings to use fetched code signing profiles:

```
xcode-project use-profiles
```

8. Install Flutter dependencies:

```
flutter packages pub get
```

9. Install CocoaPods dependencies:

```
find . -name "Podfile" -execdir pod install \;
```

10. Build the Flutter the iOS project:

```
flutter build ipa --release \
    --export-options-plist=/Users/USERNAME/export_options.plist
```

Note that `export_options.plist` is the output of the `xcode-project use-profiles` command.

11. Publish the app to App Store Connect:

```
APP_FILE=$(find $(pwd) -name "*.ipa")
app-store-connect publish \
    --path "$APP_FILE"
```
content_copy

12. As mentioned earlier, don't forget to set your login keychain as the default to avoid authentication issues with apps on your machine:

```
keychain use-login
```
content_copy

You should receive an email within 30 minutes notifying you that your build has been validated and is available to release to testers on TestFlight. At this point you can choose whether to release on TestFlight, or go ahead and release your app to the App Store.

Release your app on TestFlight

TestFlight allows developers to push their apps to internal and external testers. This optional step covers releasing your build on TestFlight.

1. Navigate to the TestFlight tab of your app's application details page on App Store Connect.
2. Select **Internal Testing** in the sidebar.
3. Select the build to publish to testers, then click **Save**.
4. Add the email addresses of any internal testers. You can add additional internal users in the **Users and Roles** page of App Store Connect, available from the dropdown menu at the top of the page.

For more details, see Distribute an app using TestFlight.

Release your app to the App Store

When you're ready to release your app to the world, follow these steps to submit your app for review and release to the App Store:

1. Select **Pricing and Availability** from the sidebar of your app's application details page on App Store Connect and complete the required information.
2. Select the status from the sidebar. If this is the first release of this app, its status is **1.0 Prepare for Submission**. Complete all required fields.
3. Click **Submit for Review**.

Apple notifies you when their app review process is complete. Your app is released according to the instructions you specified in the **Version Release** section.

For more details, see Distribute an app through the App Store.

Troubleshooting

The Distribute your app guide provides a detailed overview of the process of releasing an app to the App Store.