Get started
Samples & tutorials
Development
- User interface
- Data & backend
- Accessibility & internationalization
- Platform integration
- Packages & plugins
- Add Flutter to existing app
  - Introduction
  - Adding to an Android app
  - Adding to an iOS app
    - Project setup
    - Add a single Flutter screen
  - Running, debugging & hot reload
  - Loading sequence and performance
- Tools & techniques
- Migration notes

Testing & debugging
Performance & optimization
Deployment
Resources
Reference
- Widget index
- API reference
- Package site

# Integrate a Flutter module into your iOS project

Docs > Development > Add Flutter to existing app > Adding Flutter to iOS > Integrate Flutter

Contents

Flutter can be incrementaly added into your existing iOS application as embedded frameworks.

## System requirements

Your development environment must meet the macOS system requirements for Flutter with Xcode installed. Flutter supports iOS and later.

## Create a Flutter module

To embed Flutter into your existing application, first create a Flutter module.

From the command line, run:

```
cd some/path/
flutter create --template module my_flutter
```

A Flutter module project is created at `some/path/my_flutter/`. From that directory, you can run the same `flutter` commands yc would in any other Flutter project, like `flutter run --debug` or `flutter build ios`. You can also run the module in Android Studio/IntelliJ or VS Code with the Flutter and Dart plugins. This project contains a single-view example version of your module before it's embedded in your existing application, which is useful for incrementally testing the Flutter-only parts of your code.

### Module organization

The `my_flutter` module directory structure is similar to a normal Flutter application:

```
my_flutter/
├─── .ios/
│    ├─── Runner.xcworkspace
│    └─── Flutter/podhelper.rb
├─── lib/
│    └─── main.dart
├─── test/
└─── pubspec.yaml
```

Add your Dart code to the `lib/` directory.

Add Flutter dependencies to `my_flutter/pubspec.yaml`, including Flutter packages and plugins.

The `.ios/` hidden subfolder contains an Xcode workspace where you can run a standalone version of your module. It is a wrappe project to bootstrap your Flutter code, and contains helper scripts to facilitate building frameworks or embedding the module intc your existing application with CocoaPods.

**Note:** Add custom iOS code to your own existing application's project or to a plugin, not to the module's `.ios/` directory. Changes made in your module's `.ios/` directory do not appear in your existing iOS project using the module, and may be overwritten by Flutter.

Do not source control the `.ios/` directory since it's autogenerated. Before building the module on a new machine, run `flutter pub get` in the `my_flutter` directory first to regenerate the `.ios/` directory before building iOS project using the Flutter module.

## Embed the Flutter module in your existing application

There are two ways to embed Flutter in your existing application.

1. Use the CocoaPods dependency manager and installed Flutter SDK. (Recommended.)
2. Create frameworks for the Flutter engine, your compiled Dart code, and all Flutter plugins. Manually embed the frameworks, update your existing application's build settings in Xcode.

**Note:** Your app does not run on a simulator in Release mode because Flutter does not yet support output x86 ahead-of-time (AOT) binaries for your Dart code. You can run in Debug mode on a simulator or a real device, and Release on a real device.

Using Flutter increases your app size.

### Option A - Embed with CocoaPods and the Flutter SDK

This method requires every developer working on your project to have a locally installed version of the Flutter SDK. Simply build y application in Xcode to automatically run the script to embed your Dart and plugin code. This allows rapid iteration with the most to-date version of your Flutter module without running additional commands outside of Xcode.

The following example assumes that your existing application and the Flutter module are in sibling directories. If you have a diffe directory structure, you may need to adjust the relative paths.

```
some/path/
├── my_flutter/
│   └── .ios/
│       └── Flutter/
│         └── podhelper.rb
└── MyApp/
    └── Podfile
```

If your existing application (`MyApp`) doesn't already have a Podfile, follow the CocoaPods getting started guide to add a `Podfile` to your project.

1. Add the following lines to your `Podfile`:

```
flutter_application_path = '../my_flutter'
load File.join(flutter_application_path, '.ios', 'Flutter', 'podhelper.rb')
```

2. For each Podfile target that needs to embed Flutter, call `install_all_flutter_pods(flutter_application_path)`.

```
target 'MyApp' do
  install_all_flutter_pods(flutter_application_path)
end
```

3. Run `pod install`.

**Note:** When you change the Flutter plugin dependencies in `my_flutter/pubspec.yaml`, run `flutter pub get` in your Flutter module directory to refresh the list of plugins read by the `podhelper.rb` script. Then, run `pod install` again fron your application at`some/path/MyApp`.

The `podhelper.rb` script embeds your plugins, `Flutter.framework`, and `App.framework` into your project.

Your app's Debug and Release build configurations embeds the Debug or Release build modes of Flutter, respectively. Add a Profi build configuration to your app to test in profile mode.

**Tip:** `Flutter.framework` is the bundle for the Flutter engine, and `App.framework` is the compiled Dart code for this project.

Open `MyApp.xcworkspace` in Xcode. You can now build the project using `⌘B`.

### Option B - Embed frameworks in Xcode

Alternatively, you can generate the necessary frameworks and embed them in your application by manually editing your existing Xcode project. You may do this if members of your team can't locally install Flutter SDK and CocoaPods, or if you don't want to us CocoaPods as a dependency manager in your existing applications. You must run `flutter build ios-framework` every time you make code changes in your Flutter module.

If you're using the previous Embed with CocoaPods and Flutter tools method, you can skip these instructions.

The following example assumes that you want to generate the frameworks to `some/path/MyApp/Flutter/`.

```
flutter build ios-framework --output=some/path/MyApp/Flutter/
```

```
some/path/MyApp/
└── Flutter/
    ├── Debug/
    │   ├── Flutter.framework
    │   ├── App.framework
    │   ├── FlutterPluginRegistrant.framework (only if you have plugins with iOS platform code)
    │   └── example_plugin.framework (each plugin is a separate framework)
    ├── Profile/
    │   ├── Flutter.framework
    │   ├── App.framework
    │   ├── FlutterPluginRegistrant.framework
    │   └── example_plugin.framework
    └── Release/
        ├── Flutter.framework
        ├── App.framework
        ├── FlutterPluginRegistrant.framework
        └── example_plugin.framework
```

**Warning:** Always use `Flutter.framework` and `App.framework` from the same directory. Mixing `.framework` imports from different directories (such as `Profile/Flutter.framework` with `Debug/App.framework`) causes runtime crashes.

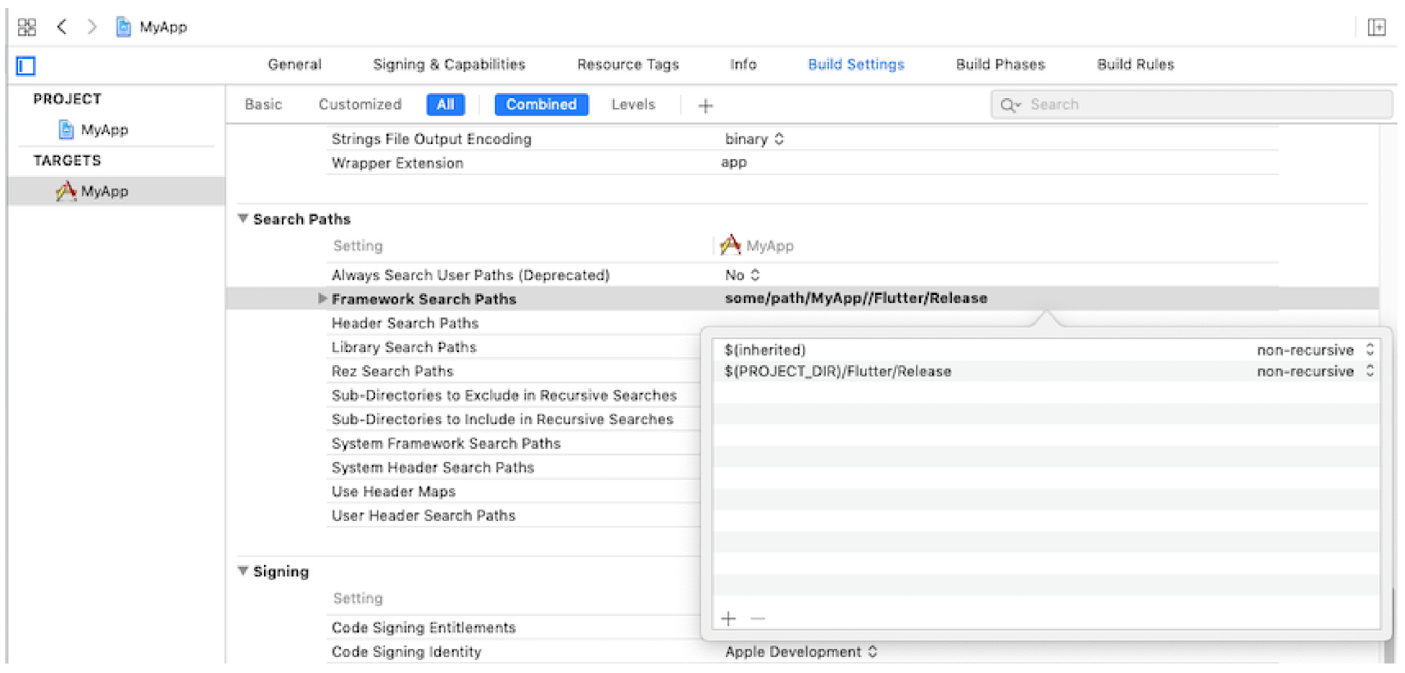
**Tip:** With Xcode 11 installed, you can generate XCFrameworks instead of universal frameworks by adding the flags `--xcframework --no-universal`.

Embed and link the generated frameworks into your existing application in Xcode. There are multiple ways to do this—use the method that is best for your project.

## Link on the frameworks

For example, you can drag the frameworks from `some/path/MyApp/Flutter/Release/` in Finder into your targets' build settings > Build Phases > Link Binary With Libraries.

In the target's build settings, add `$(PROJECT_DIR)/Flutter/Release/` to the **Framework Search Paths** (`FRAMEWORK_SEARCH_PATH`



Strings File Output Encoding: binary
Wrapper Extension: app

Search Paths
Setting — MyApp
Always Search User Paths (Deprecated): No
Framework Search Paths: some/path/MyApp//Flutter/Release
Header Search Paths
Library Search Paths
Rez Search Paths
Sub-Directories to Exclude in Recursive Searches
Sub-Directories to Include in Recursive Searches
System Framework Search Paths
System Header Search Paths
Use Header Maps
User Header Search Paths

$(inherited) non-recursive
$(PROJECT_DIR)/Flutter/Release non-recursive

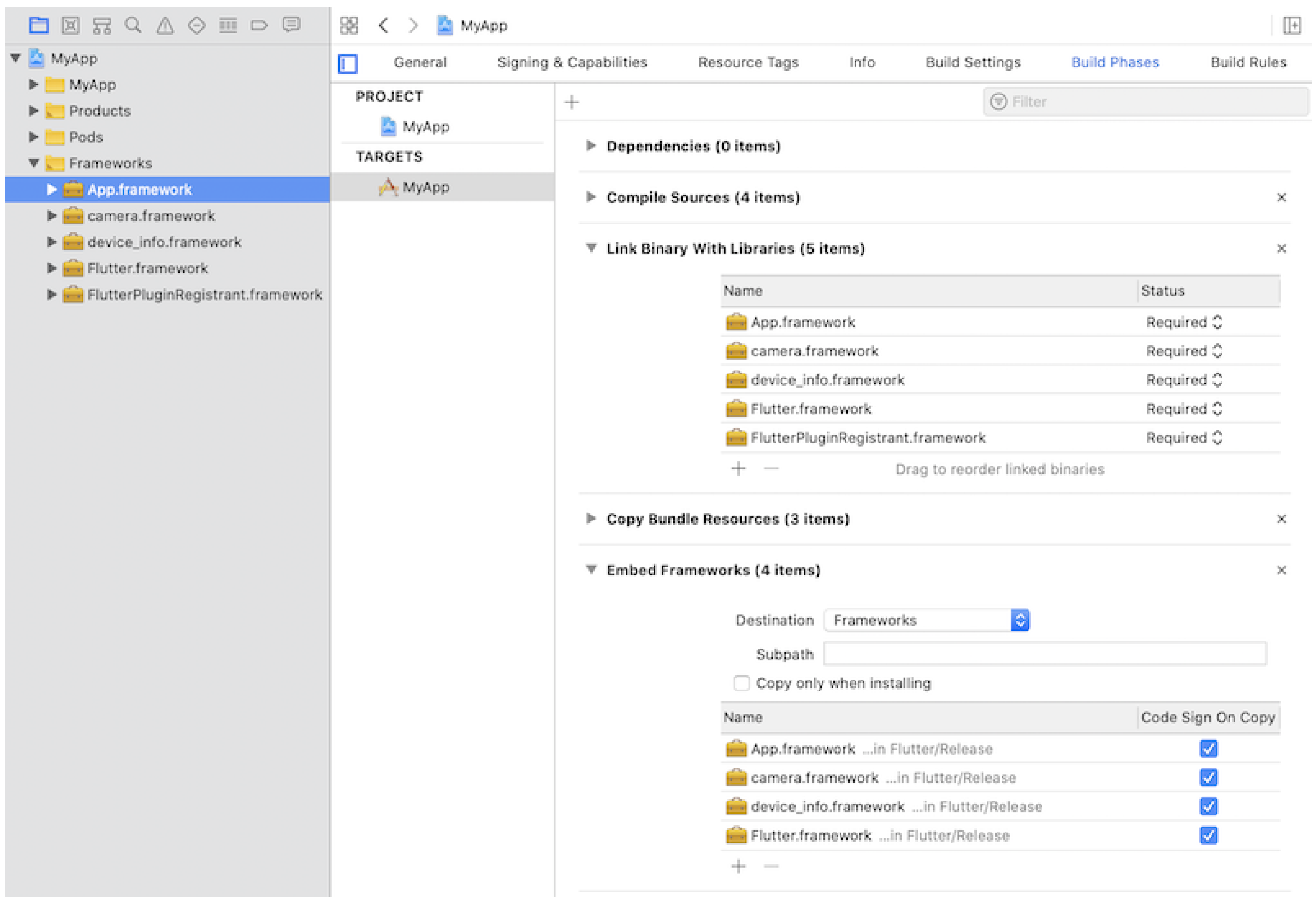
Signing
Setting
Code Signing Entitlements
Code Signing Identity: Apple Development

## Embed the frameworks

The generated dynamic frameworks must be embedded into your app to be loaded at runtime.

**Important:** Plugins might produce static or dynamic frameworks. Static frameworks should be linked on, but never embedded. If you embed a static framework into your application, your application is not publishable to the App Store and fails with a **Found an unexpected Mach-O header code** archive error.

For example, you can drag the framework (except for `FlutterPluginRegistrant` and any other static frameworks) from your application's Frameworks group into your targets' build settings > Build Phases > Embed Frameworks. Then, select **Embed & Sign** from the drop-down list.



Dependencies (0 items)

Compile Sources (4 items)

Link Binary With Libraries (5 items)

| Name | Status |
|---|---|
| App.framework | Required |
| camera.framework | Required |
| device_info.framework | Required |
| Flutter.framework | Required |
| FlutterPluginRegistrant.framework | Required |

Drag to reorder linked binaries

Copy Bundle Resources (3 items)

Embed Frameworks (4 items)

Destination: Frameworks
Subpath:
Copy only when installing

| Name | Code Sign On Copy |
|---|---|
| App.framework ...in Flutter/Release | ✓ |
| camera.framework ...in Flutter/Release | ✓ |
| device_info.framework ...in Flutter/Release | ✓ |
| Flutter.framework ...in Flutter/Release | ✓ |

You should now be able to build the project in Xcode using `⌘B`.

**Tip:** To embed the Debug version of the Flutter frameworks in your Debug build configuration and the Release version in your Release configuration, in your `MyApp.xcodeproj/project.pbxproj`, try replacing `path = Flutter/Release/example.framework;` with `path = "Flutter/$(CONFIGURATION)/example.framework";` for all added frameworks. (Note the added `"`.)

You must also add `$(PROJECT_DIR)/Flutter/$(CONFIGURATION)` to the **Framework Search Paths** (`FRAMEWORK_SEARCH_PATHS` build setting.

## Option C - Embed application and plugin frameworks in Xcode and Flutter framework with CocoaPods

Alternatively, instead of distributing the large Flutter.framework to other developers, machines, or continuous integration systems you can instead generate Flutter as CocoaPods podspec by adding the flag `--cocoapods`. This produces a `Flutter.podspec` inst of an engine Flutter.framework. The App.framework and plugin frameworks will be generated as described in Option B.

**Important:** The `--cocoapods` flag is available in Flutter v1.13.6.

```
flutter build ios-framework --cocoapods --output=some/path/MyApp/Flutter/
```

```
some/path/MyApp/
└── Flutter/
    ├── Debug/
    │   ├── Flutter.podspec
    │   ├── App.framework
    │   ├── FlutterPluginRegistrant.framework
    │   └── example_plugin.framework (each plugin with iOS platform code is a separate framework)
    ├── Profile/
    │   ├── Flutter.podspec
    │   ├── App.framework
    │   ├── FlutterPluginRegistrant.framework
    │   └── example_plugin.framework
    └── Release/
        ├── Flutter.podspec
        ├── App.framework
        ├── FlutterPluginRegistrant.framework
        └── example_plugin.framework
```

Host apps using CocoaPods can add Flutter to their Podfile:

```
pod 'Flutter', :podspec => 'some/path/MyApp/Flutter/{build_mode}/Flutter.podspec'
```

Embed and link the generated App.framework, FlutterPluginRegistrant.framework, and any plugin frameworks into your existing application as described in Option B.

## Development

You can now add a Flutter screen to your existing application.