Basic Setup

Swift is designed to provide seamless compatibility with Cocoa and Objective-C. You can use Objective-C APIs in Swift, and you can use Swift APIs in Objective-C. This makes Swift an easy, convenient, and powerful tool to integrate into your development workflow.

This guide covers three important aspects of Swift and Objective-C compatibility that you can use to your advantage when developing Cocoa apps:

- **Interoperability** lets you interface between Swift and Objective-C code, allowing you to use Swift classes in Objective-C and to take advantage of familiar Cocoa classes, patterns, and practices when writing Swift code.
- **Mix and match** allows you to create mixed-language apps containing both Swift and Objective-C files that can communicate with each other.
- **Migration** from existing Objective-C code to Swift is made easy with interoperability and mix and match, making it possible to replace parts of your Objective-C apps with the latest Swift features.

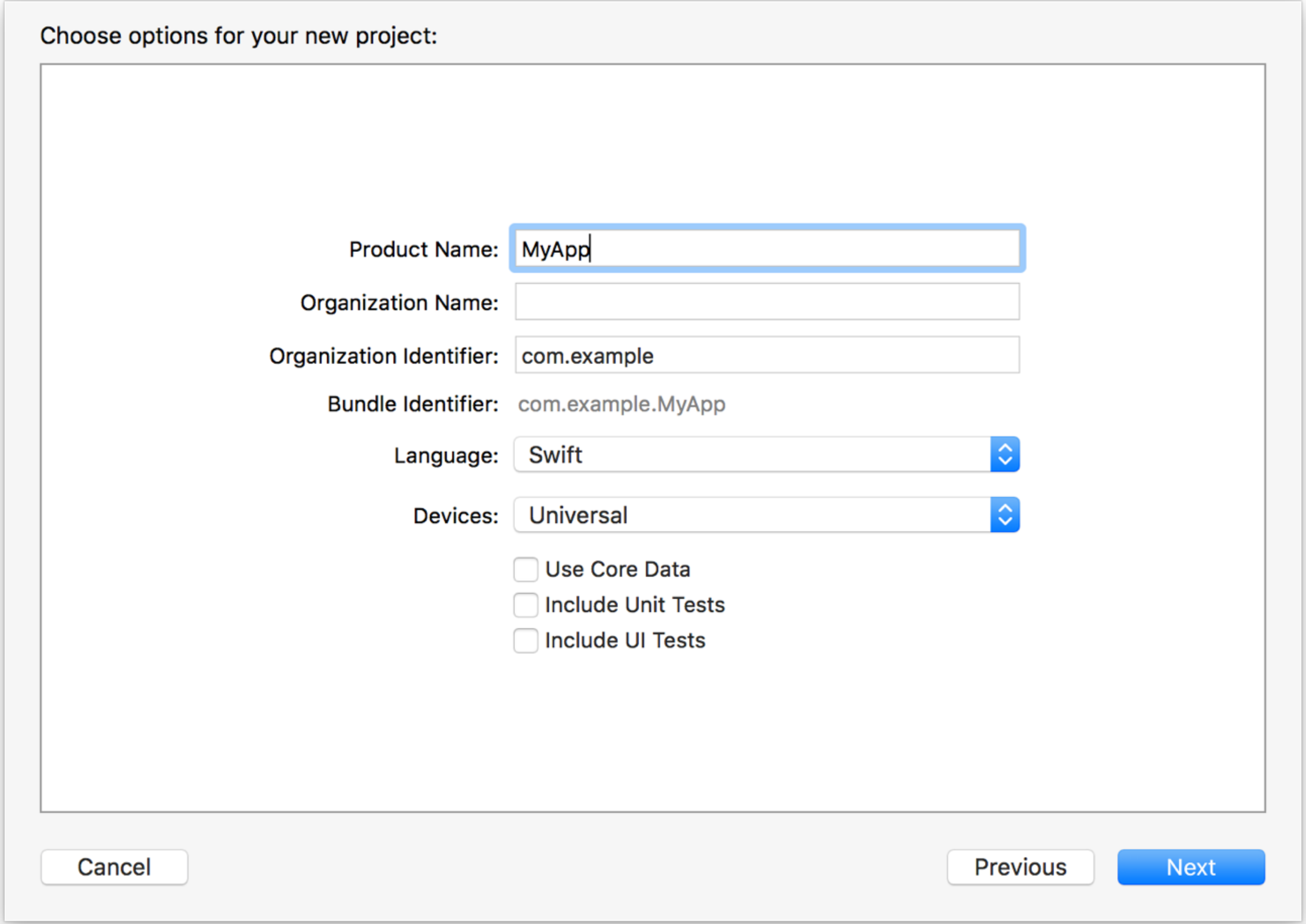
Before you get started learning about these features, you need a basic understanding of how to set up a Swift environment in which you can access Cocoa system frameworks.

Setting Up Your Swift Environment

To start experimenting with Cocoa app development using Swift, create a new Swift project from one of the provided Xcode templates.

To create a Swift project in Xcode

1. Choose File > New > Project > (iOS, watchOS, tvOS, *or* macOS) > Application > *your template of choice*.
2. Click the Language pop-up menu and choose Swift.



Choose options for your new project:

Product Name: MyApp

Organization Name:

Organization Identifier: com.example

Bundle Identifier: com.example.MyApp

Language: Swift

Devices: Universal

☐ Use Core Data

☐ Include Unit Tests

☐ Include UI Tests

A Swift project's structure is nearly identical to an Objective-C project, with one important distinction: Swift has no header files. There is no explicit delineation between the implementation and the interface—all of the information about a class, function, or constant resides in a single `.swift` file. This is discussed in more detail in Swift and Objective-C in the Same Project.

From here, you can start experimenting by writing Swift code in the app delegate or a new Swift file you create by choosing File > New > File > (iOS, watchOS, tvOS, *or* macOS) > Source > Swift.

Requirements

Creating an app using Swift 3.0 requires Xcode 8.0 or newer, as well as the following base SDK requirements:

Platform	Base SDK Requirement
macOS	10.12
iOS	10.0
watchOS	3.0
tvOS	10.0

The Swift compiler and Xcode enforce a minimum deployment target of iOS 7 or macOS 10.9. Setting an earlier deployment target results in a build failure.

> NOTE
>
> Executables built from the command line expect to find the Swift libraries in their `@rpath`. If you plan to ship a Swift executable built from the command line, you'll need to ship the Swift dynamic libraries as well. Swift executables built from within Xcode have the runtime statically linked.

Understanding the Swift Import Process

After you have your Xcode project set up, you can import any framework from Cocoa or Cocoa Touch to start working with Objective-C from Swift.

Any Objective-C framework or C library that supports *modules* can be imported directly into Swift. This includes all of the Objective-C system frameworks—such as Foundation, UIKit, and SpriteKit—as well as common C libraries supplied with the system. For example, to use Foundation APIs from a Swift file, add the following import statement to the top of the file:

```
import Foundation
```

With this import statement, that Swift file can now access all of Foundation's classes, protocols, methods, properties, and constants.

The import process is straightforward. Objective-C frameworks vend APIs in header files. In Swift, those header files are compiled down to Objective-C modules, which are then imported into Swift as Swift APIs. The importing process determines how functions, classes, methods, and types declared in Objective-C code appear in Swift. For functions and methods, this process affects the types of their arguments and return values. For types, the process of importing can have the following effects:

- Remap certain Objective-C types to their equivalents in Swift, like `id` to `Any`
- Remap certain Objective-C core types to their alternatives in Swift, like `NSString` to `String`
- Remap certain Objective-C concepts to matching concepts in Swift, like pointers to optionals

For more information on using Objective-C in Swift, see Interacting with Objective-C APIs.

> NOTE
>
> You cannot import C++ code directly into Swift. Instead, create an Objective-C or C wrapper for C++ code.

The model for importing Swift into Objective-C is similar to the one used for importing Objective-C into Swift. Swift vends its APIs—such as from a framework—as Swift modules. Alongside these Swift modules are generated Objective-C headers. These headers vend the APIs that can be mapped back to Objective-C. Some Swift APIs do not map back to Objective-C because they leverage language features that are not available in Objective-C.

For more information on using Swift in Objective-C, see Swift and Objective-C in the Same Project.