

An Apple Library Primer

This thread has been locked by a moderator.



1k

Apple's library technology has a long and glorious history, dating all the way back to the origins of Unix. This does, however, mean that it can be a bit confusing to newcomers. This is my attempt to clarify some terminology.

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

Share and Enjoy

Quinn "The Eskimo!" @ Developer Technical Support @ Apple
let myEmail = "eskimo" + "1" + "@" + "apple.com"

An Apple Library Primer

Apple's tools support two related concepts:

- **Platform** — This is the platform itself; macOS, iOS, iOS Simulator, and Mac Catalyst are all platforms.
- **Architecture** — This is a specific CPU architecture used by a platform. `arm64` and `x86_64` are both architectures.

A given architecture might be used by multiple platforms. The most obvious example of this `arm64`, which is used by all of the platforms listed above.

Code built for one platform will not work on another platform, even if both platforms use the same architecture.

Code is usually packaged in either a Mach-O file or a static library. **Mach-O** is used for executables, dynamic libraries, bundles, and object files. These can have a variety of different extensions; the only constant is that `.o` is always used for a Mach-O containing an object file. Use `otool` and `nm` to examine a Mach-O file. Use `vtool` to quickly determine the platform for which a Mach-O was built. Use `dyld_info` to get more details about a dynamic library.

IMPORTANT All the tools mentioned here are documented in man pages; for information on how to access that documentation, see [Reading UNIX Manual Pages](#).

A **dynamic library** has the extension `.dylib`. You may also see this called a shared library.

A **framework** is a bundle structure with the `.framework` extension that has both compile-time and run-time roles:

- At compile time, the framework combines the library's headers and its stub library (stub libraries are explained below).
- At run time, the framework combines the library's code, as a Mach-O dynamic library, and its associated resources.

The exact structure of a framework varies by platform. For the details, see [Placing Content in a Bundle](#).

macOS supports both frameworks and standalone dynamic libraries. Other Apple platforms support frameworks but not standalone dynamic libraries.

Historically these two roles were combined, that is, the framework included the headers, the dynamic library, and its resources. These days Apple ships different frameworks for each role. That is, the macOS SDK includes the compile-time framework and macOS itself includes the run-time one. Most third-party frameworks continue to combine these roles.

A **static library** is an archive of one or more object files. It has the extension `.a`. Use `ar`, `libtool`, and `ranlib` to inspect and manipulate these archives.

There is no such thing as a **static framework**. Well, you might hear this term used by non-Apple people, but it's not something that Apple has ever supported. DTS spends a lot of time explaining this to folks who are having mysterious build problems.

A **universal binary** is a file that contains multiple architectures for the same platform. Universal binaries always use the **universal binary format**. Use the `file` command to learn what architectures are within a universal binary. Use the `lipo` command to manipulate universal binaries.

A universal binary's architectures are either all in Mach-O format or all in the static library archive format. The latter is called a **universal static library**.

A universal binary has the same extension as its non-universal equivalent. That means a `.a` file might be a static library or a universal static library.

Most tools work on a single architecture within a universal binary. They default to the architecture of the current machine. To override this, pass the architecture in using a command-line option, typically `-arch` or `--arch`.

Apple recently introduced the **XCFramework** format, a single document package that includes libraries for any combination of platfoms and architectures. It has the extension `.xcframework`. An XCFramework holds either a framework, a dynamic library, or a static library. All the elements must be the same type. Use `xcodebuild` to create an XCFramework. For specific instructions, see [Xcode Help > Distribute binary frameworks > Create an XCFramework](#).

A **stub library** is a compact description of the contents of a dynamic library. It has the extension `.tbd`, which stands for *text-based description* (TBD). Apple's SDKs include stub libraries to minimise their size; for the backstory, read [this post](#). Stub libraries currently use YAML format, a fact that's relevant when you try to [interpret linker errors](#). Use the `tapi` tool to create and manipulate these files. In this context *TAPI* stands for a *text-based API*, an alternative name for TBD.

Apple platforms use **DWARF**. When you compile a file, the compiler puts the debug info into the resulting object file. When you link a set of object files into a executable, dynamic library, or bundle for distribution, the linker does not include this debug info. Rather, debug info is stored in a separate **debug symbols** document package. This has the extension `.dSYM` and is created using `dsymutil`. Use `symbols` to learn about the symbols in a file. Use `dwarfdump` to get detailed information about DWARF debug info. Use `atos` to map an address to its corresponding symbol name.

Over the years there have been some *really* good talks about linking and libraries at WWDC, including:

- WWDC 2022 Session 110362 [Link fast: Improve build and launch times](#)
- WWDC 2022 Session 110370 [Debug Swift debugging with LLDB](#)
- WWDC 2021 Session 10211 [Symbolication- Beyond the basics](#)
- WWDC 2019 Session 416 [Binary Frameworks in Swift](#) — Despite the name, this covers XCFrameworks in depth.
- WWDC 2018 Session 415 [Behind the Scenes of the Xcode Build Process](#)
- WWDC 2017 Session 413 *App Startup Time: Past, Present, and Future*
- WWDC 2016 Session 406 *Optimizing App Startup Time*

Note The older talks are no longer available from Apple, but you may be able to find transcripts out there on the 'net.

Historically Apple published a document, *Mac OS X ABI Mach-O File Format Reference*, or some variant thereof, that acted as the definitive reference to the Mach-O file format. This document is no longer available from Apple. If you're doing serious work with Mach-O, I recommend that you find an old copy. It's definitely out of date, but there's no better place to get a high-level introduction to the concepts. The [Mach-O Wikipedia page](#) has a link to an archived version of the document.

For the most up-to-date information about Mach-O, see the declarations and doc comments in `<mach-o/loader.h>`.

Revision History

- **2023-01-23** Explained the compile-time and run-time roles of a framework. Made other minor editorial changes.
- **2022-11-17** Added an explanation of TAPI.
- **2022-10-12** Added links to Mach-O documentation.
- **2022-09-29** Added info about `.dSYM` files. Added a few more links to WWDC sessions.
- **2022-09-21** First posted.

Linker

Reply

Posted 8 months ago by eskimo

Add a Comment

This site contains user submitted content, comments and opinions and is for informational purposes only. Apple disclaims any and all liability for the acts, omissions and conduct of any third parties in connection with or related to your use of the site. All postings and use of the content on this site are subject to the [Apple Developer Forums Participation Agreement](#).

>
Developer
>
Forums

Platforms

iOS
iPadOS
macOS
tvOS
watchOS

Tools

Swift
SwiftUI
SF Symbols
Swift Playgrounds
TestFlight
Xcode
Xcode Cloud

Topics & Technologies

Accessibility
Accessories
App Extensions
App Store
Audio & Video
Augmented Reality
Business
Design
Distribution
Education
Fonts
Games
Health & Fitness
In-App Purchase
Localization
Maps & Location
Machine Learning
Security
Safari & Web

Resources

Documentation
Curriculum
Downloads
Forums
Videos

Support
Support Articles
Contact Us
Bug Reporting
System Status

Account
Apple Developer
App Store Connect
Certificates, IDs, & Profiles
Feedback Assistant

Programs

Apple Developer Program
Apple Developer Enterprise Program
App Store Small Business Program
MFi Program
News Partner Program
Video Partner Program
Security Bounty Program
Security Research Device Program

Events
App Accelerators
App Store Awards
Apple Design Awards
Apple Developer Academies
Entrepreneur Camp
Tech Talks
WWDC