Signing a Mac Product For Distribution

This thread has been locked. Questions are automatically locked after two months of inactivity, or sooner if deemed necessary by a moderator.

I spend a lot of time helping Mac developers with notarisation and Gatekeeper problems, and many of these problems are caused by incorrect code signing. The instructions for how to sign and package a Mac product for distribution are rather scattered, so I've written them all down in one place. And rather than keep that to myself, I'm posting it here for everyone's benefit.

If you have any corrections, feel free to get in touch with me directly (my email address is in my signature). And if have any questions about this, it's probably best to ask them here on DevForums. I've locked this thread, so just start a new thread in the Distribution > Mac Apps topic area. Or, if you want one-on-one help, open a DTS tech support incident and we can pick things up in that context.

IMPORTANT None of the following has been formally reviewed, so it's not official Apple documentation.

Share and Enjoy — Quinn "The Eskimo!" Apple Developer Relations, Developer Technical Support, Core OS/Hardware

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

Signing a Mac Product For Distribution

The best way to sign and package an app is via Xcode. Build a version of your app to distribute using Xcode's Product > Archive command, and then package that archive for your distribution channel via the Organizer. See Distribute your app in Xcode Help for the details.

However, not all Mac products can be distributed this way. For example:

- An app that's distributed outside of the Mac App Store on a disk image
- A product that has to be installed via an installer package
- An app that uses a third-party development environment

In these cases you must manually sign and package your product.

Note If you find this post a little abstract, and would prefer to follow a concrete example, see Manual Code Signing Example.

Consult Resources for Third-Party Development Environments

Many third-party development environments have their own strategies for signing and packaging the products they build. If you're using a third-party development environment, consult its support resources for advice before continuing.

Decide on a Container Format

To get started, decide on your container format. Mac products support two distribution channels:

- An app can be distributed via the Mac App Store
- Apps and non-apps can be distributed outside of the Mac App Store using Developer ID signing

A Mac App Store app must be submitted as an installer package. In contrast, products distributed outside of the Mac App Store can use a variety of different container formats, the most common being:

- Zip archive (
  ```
  .zip
  ```
)
- Disk image (
  ```
  .dmg
  ```
)
- Installer package (
  ```
  .pkg
  ```
)

It's also possible to nest these. For example, you might have an app inside an installer package on a disk image.

Each container format has its own pros and cons, so pick an approach based on the requirements of your product. However, this choice affects how you package your product, something discussed in more detail below.

Structure Your Code Correctly

All code that you distribute must be signed. There's two parts to this:

- Structuring your code to support signing
- Actually signing it

You must structure your code correctly. If you don't, it may be hard (or in some cases impossible) to sign it.

First things first, identify all the code in your product. There are many types of code, including apps, app extensions, frameworks, other bundled code (like XPC Services), shared libraries, and command-like tools. Each type of code has two key attributes

- Is it bundled code? (apps, app extensions, frameworks, other bundled code)
- Is it a main executable? (apps, app extensions, command-line tools)

Both of these attributes affect how you sign the code. In addition, whether the code is bundled is critical to how you structure it. Specifically, bundled code supports the notion of *nested code*. For example, you might have an app extension nested within your app's bundle.

When dealing with nested code, follow these rules:

- Place any nested code in the appropriate nested code location. See the *Nested Code* section of Technote 2206 macOS Code Signing In Depth for that list.
- Do not place non-code items in a nested code location. Rather, place these in the bundle's resources directory (typically
  ```
  Contents/Resources
  ```
).

IMPORTANT Scripts are *not* considered code. If you have scripts — shell, Python, or whatever — place them in the resources directory. These will still be signed, but as a resource rather than as code.

Use Symlinks to Deal with Alien Structures

If you're using a complex third-party library, you may find that the structure required by the library does not match up with the structure required by macOS. In many cases you can resolve this conflict using symlinks. For details, see this DevForums post.

Sign Your Code

Sign code using the

```
codesign
```

tool. Read the following sections to learn about the specific arguments to use, but also keep these general rules in mind:

- **Do not use the**
  ```
  --deep
  ```
  argument. This feature is helpful in some specific circumstances but it will cause problems when signing a complex program. For a detailed explanation as to why, see
  ```
  --deep
  ```
  Considered Harmful.
- Rather, sign every piece of code separately. For a complex app, you should create a script to do this.
- Sign from the inside out. That is, if A depends on B, sign B before you sign A. When you sign A, the code signature encodes information about B, and changing B after the fact can break the seal on that code signature.

Basic Signing

No matter what sort of code you're signing, the basic

```
codesign
```

command looks like this:

```
% codesign -s III /path/to/your/code
```

where

```
III
```

is the name of the code signing identity to use. The specific identity varies depending on your target platform. See the following sections for details.
When signing bundled code (as defined in *Structure Your Code Correctly*) pass in the path to the bundle, not the path to the code.

If you're re-signing code — that is, the code you're signing is already signed — pass the

```
-f
```

option.
If you're signing a main executable (as defined in *Structure Your Code Correctly*) that needs entitlements, add

```
--entitlements ###.entitlements
```

, where

```
###.entitlements
```

is a path to a property list file that contains your entitlements.
If you're signing non-bundled code, set the code signing identifier by adding

```
-i ###
```

, where

```
###
```

is the bundle ID the code would have if it had a bundle ID. For example, if you have an app whose bundle ID is

```
com.example.flying-animals
```

that has a nested command-line tool called

```
pig-jets
```

, the bundle ID for that tool would logically be

```
com.example.flying-animals.pig-jets
```

, and that's a perfectly fine value to use for

```
###
```

Note For bundled code, you don't need to supply a code signing identifier because

```
codesign
```

defaults to using the bundle ID.

Mac App Store Signing

If you're distributing via the Mac App Store, use your Mac App Distribution signing identity in place of

```
III
```

in the example above. This will typically be named

```
3rd Party Mac Developer Application: ???
```

, where

```
???
```

identifies your team.

Developer ID Signing

If you're distributing outside of the Mac App Store, use your Developer ID Application signing identity in place of

```
III
```

in the example above. This will typically be named

```
Developer ID Application: ???
```

, where

```
???
```

identifies your team.
All Developer ID signed code needs a secure timestamp; enable this by adding the

```
--timestamp
```

option.
If you're signing a main executable (as defined in *Structure Your Code Correctly*), enable the hardened runtime by adding

```
-o runtime
```

option.
The hardened runtime enables additional security checks within your process. You may need to make minor code changes to be compatible with those additional security checks. For some specific examples, watch WWDC 2019 Session 703 All About Notarization. Failing that, you can opt out of these additional security checks using entitlements. See Hardened Runtime Entitlements

Build Your Container

Once you've signed the code in your product, it's time to wrap it in a container for distribution. Follow the advice appropriate for your chosen container format in the following sections. If you're using a nested container format — for example, an app inside an installer package on a disk image — work from the inside out, following the advice for each level of nesting.

Build a Zip Archive

Use the

```
ditto
```

tool to create a zip archive for your product:

1. Create a directory that holds everything you want to distribute.
2. Run
   ```
   ditto
   ```
   as follows:
   ```
   ditto -c -k --keepParent ### ZZZ
   ```
   where
   ```
   ###
   ```
   is the path to the directory from step 1 and
   ```
   ZZZ
   ```
   is the path where
   ```
   ditto
   ```
   creates the zip archive.

Zip archives cannot be signed (although their contents can be).

Build an Installer Package

Use the

```
productbuild
```

tool to create a simple installer package for a single app:

```
% productbuild --sign III --component ### /Applications PPP
```

In this example:

- ```
  III
  ```
  is either your Mac Installer Distribution or Developer ID Installer signing identity, depending on your distribution channel. This will typically be named
  ```
  3rd Party Mac Developer Installer: ???
  ```
  or
  ```
  Developer ID Installer: ???
  ```
  , where
  ```
  ???
  ```
  identifies your team.
- ```
  ###
  ```
  is the path to your app.
- ```
  PPP
  ```
  is the path where
  ```
  productbuild
  ```
  creates the installer package.

IMPORTANT The above is the simplest possible example. There are many different ways to create installer packages. See the man pages for

```
productbuild
```

,

```
productsign
```

,

```
pkgbuild
```

and

```
pkgutil
```

for more details.

Build a Disk Image

Use the

```
hdiutil
```

tool to create a disk image for distribution:

1. Create a directory to act as the source for the root directory of your disk image's volume.
2. Populate that directory with the items you want to distribute.
3. Use
   ```
   hdiutil
   ```
   to create the disk image:
   ```
   % hdiutil create -srcFolder SSS -o DDD
   ```
   where
   ```
   SSS
   ```
   is the directory from step 1 and
   ```
   DDD
   ```
   is the path where
   ```
   hdiutil
   ```
   creates the disk image.
4. Use
   ```
   codesign
   ```
   to sign the disk image:
   ```
   % codesign -s III --timestamp -i ### DDD
   ```
   where
   ```
   III
   ```
   is your Developer ID Application signing identity (typically named
   ```
   Developer ID Application: ???
   ```
   , where
   ```
   ???
   ```
   identifies your team),
   ```
   ###
   ```
   is a pseudo bundle ID as discussed in *Basic Signing*, and
   ```
   DDD
   ```
   is the path to the disk image from step 3.

IMPORTANT There are various third-party tools that can help you create a disk image in exactly the right way. For example, the tool might arrange the icons nicely, set a background image, and add a symlink to

```
/Applications
```

. If you use such a tool, or create your own tool for this, make sure that the resulting disk image:

- Is signed with your Developer ID signing identity
- Is a UDIF-format read-only zip-compressed disk image (type UDZO)

Notarisation

If you're distributing outside of the Mac App Store, you must notarise the file you intend to distribute to your users. For instructions on doing this, see Customizing the Notarization Workflow. Skip the *Export a Package for Notarization* section because you already have the file that you want to submit.

If you're using a nested container format, only notarise the outermost container. For example, if you have an app inside an installer package on a disk image, sign the app, sign the installer package, and sign the disk image, but only notarise the disk image.

The exception to this rule is if you have a custom third-party installer. In that case, see the discussion in Customizing the Notarization Workflow.

Stapler

Once you have notarised your product, you should staple the resulting ticket to the file you intend to distribute. Customizing the Notarization Workflow discusses how to do this for a zip archive. The other common container formats (installer package, disk image) support stapling directly. For example:

```
% xcrun stapler staple FlyingAnimals.dmg
```

Note Stapling is recommended but not mandatory. If you don't staple, a user may have problems if they try to install or run your app for the first time when the Mac is offline.

Change history:

- 20 Jan 2020 — First version.
- 27 Jan 2020 — Minor editorial changes.
- 9 Mar 2020 — Moved the details of
  ```
  --deep
  ```
  into a separate post,
  ```
  --deep
  ```
  Considered Harmful.
- 15 Mar 2020 — Fixed a typo.
- 30 Mar 2020 — Added a link to Manual Code Signing Example.

Gatekeeper Developer ID

Asked 1 year ago by eskimo 4

Reply to this question
  ```