Manual Code Signing Example

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

687

In Signing a Mac Product For Distribution, I explained the rules for how to sign a Mac product for distribution. This post is a concrete example of that process.

Share and Enjoy
—
Quinn “The Eskimo!” @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

The Scenario

For this example I’m assuming you have a sandboxed app, QShare, that contains a shared extension (appex). You’ve factored the common code out into a framework, QCore. That framework contains a helper tool, `QCoreTool`. Thus, the overall structure of the app looks something like this:

```
QShare.app
QShare.app/Contents/
QShare.app/Contents/Info.plist
QShare.app/Contents/MacOS/
QShare.app/Contents/MacOS/QShare
QShare.app/Contents/Resources/
QShare.app/Contents/Resources/MainMenu.nib
QShare.app/Contents/Frameworks/
QShare.app/Contents/Frameworks/QCore.framework/
QShare.app/Contents/Frameworks/QCore.framework/QCore -> Versions/Current/QCore
QShare.app/Contents/Frameworks/QCore.framework/Resources -> Versions/Current/Resources
QShare.app/Contents/Frameworks/QCore.framework/Versions/
QShare.app/Contents/Frameworks/QCore.framework/Versions/Current -> A
QShare.app/Contents/Frameworks/QCore.framework/Versions/A/
QShare.app/Contents/Frameworks/QCore.framework/Versions/A/QCore
QShare.app/Contents/Frameworks/QCore.framework/Versions/A/Resources/
QShare.app/Contents/Frameworks/QCore.framework/Versions/A/Resources/Info.plist
QShare.app/Contents/Frameworks/QCore.framework/Versions/A/Helpers/
QShare.app/Contents/Frameworks/QCore.framework/Versions/A/Helpers/QCoreTool
QShare.app/Contents/PlugIns/
QShare.app/Contents/PlugIns/QShareExtension.appex/
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/Info.plist
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/MacOS
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/MacOS/QShareExtension
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/Resources
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/Resources/icon.icns
QShare.app/Contents/PlugIns/QShareExtension.appex/Contents/Resources/ShareViewController.nib
```

You can actually build such an app in Xcode, and then use Product > Archive to create an Xcode archive (`.xcarchive`) for it. In that case you could use Organizer to upload the app in this archive to either the Mac App Store or the notary service. However, in this case we’re assuming that you want to do that process manually, with the specific goal of creating a disk image that’s suitable for distribution outside of the Mac App Store.

IMPORTANT This manual code signing approach does not require that you build your code with Xcode. The same approach works for apps built using traditional UNIX tools (`make`, for example), third-party development environments and so on. The only requirement is that the input be structured appropriately (for example, if you’re signing an app, the input app must be structured like an app).

Setup

Signing code by hand is a tedious process. For any reasonably complex product you’ll want to automate this, and the logical tool for that automation is a shell script. The final script is at the end of this post but I’m going to describe the critical steps in detail. This description assumes that there’s an `ARCHIVE` shell variable that contains the path to the input Xcode archive.

The first step is to set up some variables that point to important locations:

```
WORKDIR="QShare-`date '+%Y-%m-%d_%H.%M.%S'`"
DMGROOT="${WORKDIR}/QShare"
APP="${WORKDIR}/QShare/QShare.app"
DMG="${WORKDIR}/QShare.dmg"
```

The script creates a work directory in which to operate. It creates this in the current working directory, with a structure like this:

```
QShare-yyyy-mm-dd_hh.mm.ss/             <- WORKDIR
    QShare/                             <- DMGROOT
        QShare.app/                     <- APP
            … as above …
    QShare.dmg                          <- DMG
```

The next step is to copy the app from the Xcode archive to the `DMGROOT` directory within that working directory:

```
mkdir -p "${DMGROOT}"
cp -R "${ARCHIVE}/Products/Applications/QShare.app" "${DMGROOT}/"
```

IMPORTANT This passes `-R` to `cp` because `-r` does not preserve symlinks.

What to Sign?

The first step in manually signing a complex product is to determine what needs to be signed. QShare has four separate code items:

```
Kind      Bundled? Main? Path
----      -------- ----- ----
app       yes      yes   QShare.app
appex     yes      yes   QShare.app/Contents/PlugIns/QShareExtension.appex
framework yes      no    QShare.app/Contents/Frameworks/QCore.framework
tool      no       yes   QShare.app/Contents/Frameworks/QCore.framework/Versions/A/Helpers/QCoreTool
```

For each item, you must decide whether it is:

- Bundled, or stands alone
- A main executable, or a library

These answers inform how you sign the code. For example:

- For a bundle you must sign the top of the bundle, whereas for standalone code you just sign the code itself.
- For a main executable you must set the hardened runtime flag and can optionally supply entitlements, whereas neither of those make sense for library code.

IMPORTANT Do not apply entitlements to code that isn’t a main executable. Such entitlements are not useful and can cause problems in specific circumstances.

Entitlements

Every main executable may have entitlements. In the case of QShare that means the app, appex, and tool. These entitlements are different for each executable:

- QShare is as sandboxed app, so all executables must have `com.apple.security.app-sandbox`.
- The tool must have `com.apple.security.inherit` so that it inherits its sandbox from its parent. The tool must not have any other entitlements.
- The appex makes outgoing network connections, so it needs the `com.apple.security.network.client` entitlement.
- The app also accepts incoming network connections, so it needs the `com.apple.security.network.server` entitlement as well.

Thus, you will need three separate `.entitlements` files, one for each of the executables, with the right entitlements in each.

Signing Order

The next step is to decide the signing order. Signing a Mac Product For Distribution says that you must sign your code from the “inside out”, but what does that mean in this case?

The code items in QShare have a number of dependencies:

- The app and appex both depend on the framework
- The appex is embedded within the app
- The tool is embedded within the framework

Thus, the signing order must be:

1. Tool
2. Framework
3. Appex
4. App

Sign

With all of the above sorted out, actually signing the code is pretty straightforward:

```
codesign -s "Developer ID Application" -f --timestamp -i com.example.apple-samplecode.QShare.QCoreTool -o
runtime --entitlements "${WORKDIR}/tool.entitlements"
"${APP}/Contents/Frameworks/QCore.framework/Versions/A/Helpers/QCoreTool"
codesign -s "Developer ID Application" -f --timestamp
"${APP}/Contents/Frameworks/QCore.framework"
codesign -s "Developer ID Application" -f --timestamp                                                -o
runtime --entitlements "${WORKDIR}/appex.entitlements" "${APP}/Contents/PlugIns/QShareExtension.appex"
codesign -s "Developer ID Application" -f --timestamp                                                -o
runtime --entitlements "${WORKDIR}/app.entitlements"   "${APP}"
```

The only oddity here is the `-i com.example.apple-samplecode.QShare.QCoreTool` argument when signing the tool. This is necessary because the tool has no `Info.plist`, so `codesign` can’t infer a reasonable signing identifier from the bundle ID. See Signing a Mac Product For Distribution for more on this.

Package

Once you have a final app, create a disk image using `hdiutil`:

```
hdiutil create -srcFolder "${DMGROOT}" -quiet -o "${DMG}"
```

Then sign that disk image using `codesign`:

```
codesign -s "Developer ID Application" --timestamp -i com.example.apple-samplecode.QShare.DiskImage "${DMG}"
```

Notarise

At this point you have a final product that’s ready for notarisation. For more information on how to notarise outside of Xcode, see Customizing the Notarization Workflow.

IMPORTANT This article describes how to notarise a zip archive but don’t make the mistake of thinking that you *must* notarise a zip archive. The notary service accepts disk images, so you can submit the disk image from the previous step directly.

The Final Script

Here’s the final script to sign and package the QShare app:

```
#!/bin/sh

# Fail if any command fails.

set -e

# Check and unpack the arguments.

if [ $# -ne 1 ]
then
    echo "usage: package-archive.sh /path/to.xcarchive" > /dev/stderr
    exit 1
fi
ARCHIVE="$1"

# Establish a work directory, create a disk image root directory within
# that, and then copy the app there.
#
# Note we use `-R`, not `-r`, to preserve symlinks.

WORKDIR="QShare-`date '+%Y-%m-%d_%H.%M.%S'`"
DMGROOT="${WORKDIR}/QShare"
APP="${WORKDIR}/QShare/QShare.app"
DMG="${WORKDIR}/QShare.dmg"

mkdir -p "${DMGROOT}"
cp -R "${ARCHIVE}/Products/Applications/QShare.app" "${DMGROOT}/"

# When you use `-f` to replace a signature, `codesign` prints `replacing
# existing signature`.  There's no option to suppress that.  The message
# goes to `stderr` so you don't want to redirect it to `/dev/null` because
# there might be other interesting stuff logged to `stderr`.  One way to
# prevent it is to remove the signature beforehand, as shown by the
# following lines.  It does slow things down a bunch though, so I've made
# it easy to disable them.

if true
then
    codesign --remove-signature "${APP}"
    codesign --remove-signature "${APP}/Contents/PlugIns/QShareExtension.appex"
    codesign --remove-signature "${APP}/Contents/Frameworks/QCore.framework"
    codesign --remove-signature "${APP}/Contents/Frameworks/QCore.framework/Versions/A/Helpers/QCoreTool"
fi

# Create various entitlement files from 'here' documents.

cat > "${WORKDIR}/app.entitlements" <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.network.client</key>
    <true/>
    <key>com.apple.security.network.server</key>
    <true/>
</dict>
</plist>
EOF

cat > "${WORKDIR}/appex.entitlements" <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.network.client</key>
    <true/>
</dict>
</plist>
EOF

cat > "${WORKDIR}/tool.entitlements" <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.inherit</key>
    <true/>
</dict>
</plist>
EOF

# Sign the app from the inside out.
#
# Notes:
#
# * The tool is within the framework, so we have to sign that before the
#   framework.
#
# * The app and the appex depend on the framework, so we have to sign the
#   framework before of those.
#
# * The appex is within the app, so we have to sign that before the app.
#
# * The tool is not bundled, thus doesn't have an `Info.plist`, and thus
#   you have to explicitly set a code signing identifier.
#
# * The tool, appex and app are all executables, and thus need to the
#   hardened runtime flag.
#
# * The tool, appex and app all need unique entitlements.

codesign -s "Developer ID Application" -f --timestamp -i com.example.apple-samplecode.QShare.QCoreTool -o
runtime --entitlements "${WORKDIR}/tool.entitlements"
"${APP}/Contents/Frameworks/QCore.framework/Versions/A/Helpers/QCoreTool"
codesign -s "Developer ID Application" -f --timestamp
"${APP}/Contents/Frameworks/QCore.framework"
codesign -s "Developer ID Application" -f --timestamp                                                -o
runtime --entitlements "${WORKDIR}/appex.entitlements" "${APP}/Contents/PlugIns/QShareExtension.appex"
codesign -s "Developer ID Application" -f --timestamp                                                -o
runtime --entitlements "${WORKDIR}/app.entitlements"   "${APP}"

# Create a disk image from our disk image root directory.

hdiutil create -srcFolder "${DMGROOT}" -quiet -o "${DMG}"

# Sign that.

codesign -s "Developer ID Application" --timestamp -i com.example.apple-samplecode.QShare.DiskImage "${DMG}"

echo "${DMG}"
```

Change history:

- 30 Mar 2020 — First posted.
- 26 Feb 2021 — Fixed the formatting. Minor editorial changes.

Gatekeeper Developer ID

Asked 11 months ago by eskimo

Reply to this question

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 › Apple Developer Forums

Discover: macOS, iOS, watchOS, tvOS, Safari and Web, Games, Business, Education, WWDC
Design: Human Interface Guidelines, Resources, Videos, Apple Design Awards, Fonts, Accessibility, Internationalization, Accessories
Develop: Xcode, Swift, Swift Playgrounds, TestFlight, Documentation, Videos, Downloads
Distribute: Developer Program, App Store, App Review, Mac Software, Apps for Business, Safari Extensions, Marketing Resources, Trademark Licensing
Support: Articles, Developer Forums, Feedback & Bug Reporting, System Status, Contact Us
Account: Certificates, Identifiers & Profiles, App Store Connect

To view the latest developer news, visit News and Updates.

Copyright © 2021 Apple Inc. All rights reserved.