LIVECODE (https://livecode.com) Platform (https://livecode.com/products/livecode-platform/) Resources (https://livecode.com/resources/)

Pricing (https://livecode.com/products/livecode-platform/pricing/) Services (https://livecode.com/services/) Blog (https://livecode.com/blog/)

login (https://livecode.com/login/)

Dictionary (https://livecode.com/resources/api/) Guides (https://livecode.com/resources/guides/) Lessons (https://lessons.livecode.com/) Courses (https://livecode.com/products/learn/)

Signing and Notarizing macOS Apps for Gatekeeper

This is a checklist of all the tasks I ran into that were required to get a LiveCode macOS app signed and notarized so that Gatekeeper will accept it when the user downloads it from someplace other than the Mac App Store. Because Apple requirements change continually, this checklist will slowly get out of date and experience bit-rot. Let me know if these instructions fail you and I'll attempt to fix them.

Distributing an app yourself, not through the Mac App Store, gives you a bit more freedom in what your app can do. The steps to get it signed and notarized are a bit fewer than uploading to the Mac App Store.

Please note, line wrapping within this article for the command line commands can hide or imply spaces. Suggest you copy the command lines into a text editor to see the correct command, spaces or not. Kee Nethery

My thanks to; Mark Alldritt of Late Night Software makers of Script Debugger (for AppleScripts) and SD Notary (for Notarizing macOS apps), Peter Lewis of Keyboard Maestro maker of ... Keyboard Maestro, Monte Goulding of LiveCode, and various other anonymous friends of mine who are very smart.

1. Prepare your stack for building

Run whatever automated script you have that cleans up your stack before you turn it into an application.

2. Set the version number and bundle identifier

In the LiveCode Standalone Application Settings window for Mac, set the version numbers and the Bundle Identifier.

Apple expects version numbers of no more than 3 numbers "1.2.3" whereas LiveCode defaults to 4 numbers "1.0.0.0". Change the version number to 3 numbers.

Apple expects a Bundle Identifier, which is typically your company URL in reverse with the name of the application (no spaces). For example: "com.<company_name>.<app_name>" or as another example: "com.mycompany.myapp" without the quotes.

3. Build the standalone

Save as standalone application. Build for Mac OS X 64-bit only.

4. Remove extended attributes

LiveCode saves a bunch of extended attributes into the files in the standalone application and they must all be removed. Use the Terminal command line to see all the stuff that needs to be removed.

NOTE: Whenever you see something like "<path_to_file>" in a Terminal instruction, drag that file into Terminal and Terminal will enter in the entire path correctly. Don't type, drag.

```
sudo xattr -lr <path_to_standalone_app_bundle>
```

And to actually remove all that cruft:

```
sudo xattr -cr <path_to_standalone_app_bundle>
```

Run the first command line "-lr" a second time to verify that all the cruft has been removed.

5. Set the tsNet bundle identifier

NOTE: tsNet.bundle is only included in apps that utilize internet connectivity. Skip this step if you do not have "tsNet.bundle" in Show Package Contents / Contents / MacOS / Externals / tsNet.bundle.

There is a code resource within LiveCode app bundles named tsNet and your tsNet in your app needs to be "owned" by you otherwise you get a CFBundleIdentifier Collision when uploading your app. Go to Show Package Contents / Contents / MacOS / Externals / tsNet.bundle / Show Package Contents / Contents / Info.plist If your domain is "YOURCOMPANY.com" and your app is named "YOURPRODUCT" then replace "au.com.techstrategies.external.tsNet" with "com.YOURCOMPANY.YOURPRODUCT.tsNet" in the Info.plist file in the tsNet.bundle.

For my app, I edited it to:

```
        <key>CFBundleIdentifier</key>
        <string>com.txfconvert.txfconvert.tsNet</string>
```

6. Apple Developer Certificates

To upload to the App Store you need an Apple Developer account and corresponding developer certificates. Enroll in Apple's developer program at:

https://developer.apple.com/programs/enroll/ (https://developer.apple.com/programs/enroll/)

Create your Apple Developer installer and application certificate pair, and store them in your keychain on your development Mac.

7. Code sign executables

All of the executables in your app must be code signed with your Apple Developer ID certificate. Open the application "Keychain Access" in your Utilities folder. Select the keychain "login" and the category of "My Certificates". In the list should be two:

Developer ID Application: <your_name> (<your_ID>)

Developer ID Installer: <your_name> (<your_ID>)

For example:

Developer ID Application: Kee Nethery (CEASNJ1234)

Developer ID Installer: Kee Nethery (CEASNJ1234)

If you have "Developer ID Application: ..." and "Developer ID Installer: ..." in the "Certificates" section within "KeyChain Access", and no certificates within the "My Certificates" section, you will need to create the certificates that get stored in "My Certificates".

To create these certificates, in developer.apple.com look for the area "Certificates, Identifiers and Profiles" also displayed as "Certificates, IDs and Profiles". Select that. In the upper left, select "macOS" to create macOS certificates. From that point the path is: macOS / Production / What type of certificate do you need? / Production / Developer ID / [Continue] / Developer ID Application. You'll create a CSR with your Apple ID email and company name that gets saved to disk. Continue, select the CSR, and it will create a file in your Downloads folder. Run through the process again and select the other choice "Developer ID Installer". You need both certificates. Continue in the web site and you'll be able to select them and have them be imported into your keychain so that they show up in "My Certificates".

You will need the "Application" portion to do the code sign. You will also need Xcode installed with it's utilities to perform the code sign. Using the above sample, in Terminal, code sign the .app standalone (it automatically handles signing everything inside the app):

```
sudo codesign --verbose --deep --force --timestamp --options=runtime --strict --sign "Developer ID Application: Kee Nether
y (CEASNJ1234)" <path_to_standalone_app_bundle>
```

Just so you know, "--options=runtime" tells codesign to enable "hardened runtime" which is essential for a notarized app.

8. Verify the signing

Just to confirm all is well, verify the code signing:

```
sudo codesign --verify --verbose <path_to_standalone_app_bundle>
```

No errors in the response means that it is code signed.

9. Test against system policies

use the spctl utility to determine if the software to be notarized will run with the system policies currently in effect. "-vvv" tells it to be ultra verbose in its response.

```
sudo spctl -vvv --assess --type exec <path_to_standalone_app_bundle>
```

10. Create the installer package

Create the installer package. The Info-plist file is inside the app. Use Show Package Contents to open the app and locate the Info.plist file.

```
sudo productbuild --component <path_to_standalone_app_bundle> /Applications --sign "Developer ID Installer: Kee Nethery (C
EASNJ1234)" --product <path_to_standalone_app_bundle_Info.plist> <path_to_standalone_app_bundle_delete_".app">.pkg
```

For that last path, it is the path to the standalone application bundle with the .app suffix replaced by .pkg

For example:

LiveCode Projects/myApp.app

would become:

LiveCode Projects/myApp.pkg

This will create a properly signed installer package file in the same folder as the application. I find it easier to drag the app bundle into the Terminal command and then use backspace to replace the ".app" with ".pkg".

11. Test the installer

Test the installer. (Note the "-target /" that comes after the path to the pkg file)

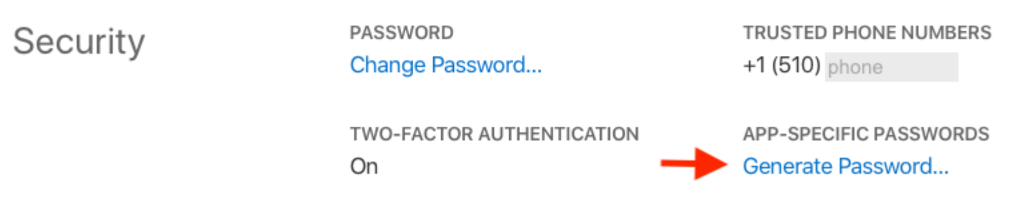
```
sudo installer -store -package <path_to_app_pkg_file> -target /
```

It will go through the installation process to confirm that the installer package is OK and install the app in your Applications folder.

12. Create an App-Specific Password

To notarize your application package you need an App-Specific Password for it. The password for notarization is NOT your developer ID password.

Log into appleid.apple.com and in the security section.



Select: Generate Password...

Provide a label for the app password. It creates one app-specific password for you, for example:

"Your app specific password is: jurt-wsma-pdnd-qumk".

Store the password someplace so that you can refer to it in the future. You'll need it each time you notarize (or attempt to notarize) this app.

NOTE: Once you've created the app specific password, you should re-use it for each notarization attempt for that app.

13. Notarize your application package

<bundle_id> is the "Bundle Identifier" that you entered in Standalone Application Settings/Mac/Bundle Identifier. It's the reverse URL format that specifies your app. For example: "com.<your_domain>.<your_app_name>"

<username> is your Apple Developer ID, the email (typically) that you use to log into developer.apple.com For example: "user@domain.com".

```
xcrun altool --notarize-app -f <path_to_app_pkg_file> --primary-bundle-id <bundle_id> --verbose -u <username>
```

When it asks for the password, give it the app-specific password you just created (or stored and are re-using).

```
<username>'s password:
```

Assuming all is well, after many many minutes of waiting for Terminal to provide any kind of response (be patient) your app will get completely uploaded and assigned a RequestUUID.

```
RequestUUID = 55f6d3f4-519b-4223-85af-3290fe2709a5
```

After that your app goes into some gigantic machine at Apple and gets analyzed until the machine emails you a result. In my case for my first upload:

> The Mac software that you uploaded was not notarized. Please review the notarization log with Xcode or altool, address the issues it shows, and upload your software again.
>
> Bundle Identifier: com.mycompany.myapp
> Request Identifier: 55f6d3f4-519b-4223-85af-3290fe2709a5

14. Notarization Info and Status

Once Terminal is done with xcrun altool, check the status of the notarization.

```
xcrun altool --notarization-info <RequestUUID> -u <username>
```

Enter your app-specific password when requested. The result will include a really log obscure URL, LogFileURL, and by going to that URL you'll see the list of everything that needs to be corrected before your app can be notarized (or that your app was successfully notarized).

15. Staple the Ticket to Your Distribution

Once you have notarized your .pkg, you must staple the ticket, that got stored in a temp directory on your Mac, to the .pkg file that the customer downloads.

```
sudo xcrun stapler staple <path_to_app_pkg_file>
```

16. Verify Staple worked

Last but not least, make sure the staple actually worked. Your Mac must be connected to the internet for validate to work.

```
sudo xcrun stapler validate --verbose <path_to_app_pkg_file>
```

17. Useful URLs

macOS Code Signing in Depth: https://developer.apple.com/library/archive/technotes/tn2206/_index.html (https://developer.apple.com/library/archive/technotes/tn2206/_index.html)

Notarizing Your App Before Distribution: https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution?language=objc#3087730 (https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution?language=objc#3087730)

Customizing the Notarization Workflow:
https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution/customizing_the_notarization_workflow?language=objc (https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution/customizing_the_notarization_workflow?language=objc)

Resolving Common Notarization Issues:
https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution/resolving_common_notarization_issues?language=objc (https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution/resolving_common_notarization_issues?language=objc)

Troubleshooting Failed Signature Verification: https://developer.apple.com/library/archive/technotes/tn2318/_index.html (https://developer.apple.com/library/archive/technotes/tn2318/_index.html)

Entitlements Troubleshooting: https://developer.apple.com/library/archive/technotes/tn2415/_index.html (https://developer.apple.com/library/archive/technotes/tn2415/_index.html) Note, my example doesn't have entitlements but it is possible that yours might need entitlements after runtime hardening. The exceptions you might need after hardening can be seen at: https://help.apple.com/xcode/mac/current/#/devf87a2ac8f (https://help.apple.com/xcode/mac/current/#/devf87a2ac8f)

18. Feedback, Corrections, Improvements

There are libraries that LiveCode adds other than the ones I discuss; revsecurity.dylib and tsNet.bundle. If your app includes other libraries, please let me know what they are, where they are in the app package (for example: "Go to Show Package Contents / Contents / MacOS / Externals / tsNet.bundle"), and anything you did or didn't have to do to make them work with the notarization process. I'll add your information to this lesson.

I have been told that LiveCode libraries should be moved into a folder you would create at: "Contents / Frameworks /" but it has been my experience that moving them from where LiveCode puts libraries to this new location is not required. Perhaps that will change in the future.

0 Comments