How to Release an Electron App on the Mac App Store

Based on new rules for Catalina OS

By Steve Carey - Originally published Feb 4, 2020. Last updated Nov 18, 2021.

Example Code: Github

Introduction

You have two options for distributing an Electron App for MacOS. Either directly (see Notarizing Your Electron Application), or through the Mac App Store (MAS). All apps distributed though the MAS must be sandboxed, meaning they are completely self contained except for any approved entitlements. And Mac's Catalina OS released in Oct. 2019 added a number of changes impacting sandboxed apps, causing many existing Electron apps to crash and new apps to be rejected. Below are the steps you need to take to get a basic app successfully submitted for review and approved for distribution on the Mac App Store.

Reference: developer.apple.com/app-store/review
developer.apple.com/app-store/review/guidelines
electronjs.org/docs/tutorial/mac-app-store-submission-guide

1) Starting with your completed Electron Application.

You have to use Electron version 8.0.2 or later (or patched versions 6.1.7 and 5.0.13) since earlier versions used private APIs which Apple no longer allows.

Electron should be a development (or global) dependency not a production dependency.

```
$ npm install -D electron
```

2) Apple Developer Account

You need an Apple Developer Account. Costs ~$100/year. Sign up at developer.apple.com.

3) Create Two Signing Certificates

For each certificate it is a two step process.

3rd Party Mac Developer Application certificate:

A) Request a certificate:

- Launch Keychain Access located in /Applications/Utilities.
- Click the Keychain Access menu > Certificate Assistant > Request a Certificate from a Certificate Authority.
- In the Certificate Assistant dialog, enter an email address in the User Email Address field.
- In the Common Name field, enter a name for the key (e.g., 3rd Party Mac Developer).
- Leave the CA Email Address field empty.
- Choose "Saved to disk" saving it to the desktop, and click Continue. Filename is CertificateSigningRequest.certSigningRequest

B) Import the new certificate from your developer account to your Keychain:

- go to https://developer.apple.com/account > Certificates > + (to add a new certificate) > Mac App Distribution - continue > Choose File > Double-click the certificate file you downloaded above > Download the new certificate file > Double click the downloaded file and it will be automatically loaded to your keychain.
- Note that the name of the certificate on your developer account is Mac App Distribution, while the name of the certificate on your Keychain is 3rd Party Mac Developer. They are the same certificate.
- Delete the CertificateSigningRequest.certSigningRequest and the certificate download files from the desktop.

3rd Party Mac Developer Installer certificate:

Repeat the above process:

A) Request a certificate:

- Launch Keychain Access app > Keychain Access menu > Certificate Assistant > Request a Certificate from a Certificate Authority.
- In the Certificate Assistant dialog, enter an email address in the User Email Address field.
- In the Common Name field, enter a name for the key (e.g., 3rd Party Mac Installer).
- Leave the CA Email Address field empty.
- Choose "Saved to disk" saving it to the desktop, and click Continue. Filename is CertificateSigningRequest.certSigningRequest.

B) Import the new certificate from your developer account to your Keychain.

- go to https://developer.apple.com/account > Certificates > + (to add a new certificate) > Mac Installer Distribution - continue > Choose File > Double-click the certificate file you downloaded above > Download the new certificate file > Double click the downloaded file and it will be automatically loaded to your keychain.
- The certificate name is Mac Installer Distribution on your developer account and 3rd Party Mac Developer Installer on you Keychain.
- Delete the CertificateSigningRequest.certSigningRequest and the certificate download files from the desktop.

To view the certificates:

- To view the certificates in your keychain: Open the Keychain Access app in the Application Utilities folder > Click on the Login Keychain (should be selected by default) > Select Category: MyCertificates > they should be listed there along with your Team ID number like: 3rd Party Mac Developer Application: MyTeam Name (12345ABCDE). For a solo developer your team name is just your name.
- To view the certificates in you developer account go to developer.apple.com > Certificates, IDs, & Profiles menu > Certificates menu > Mac App Distribution and Mac Installer Distribution type certificates should be listed there.

4) Create an App ID

- Go to your developer account - developer.apple.com > Certificates, IDs & Profiles > Identifiers > + (to add a new id) > select App IDs > continue button.
- On the Register App ID page:
  - Platform: MacOS
  - Bundle ID: Explicit. Apple recommends using a reverse-domain name style string (i.e., com.domainname.appname). This does not need to correspond with an actual website. This will need to match the appId property in the package.json file.
  - When done click continue, review it, then click register.

5) Add the App to your developer account

Reference: help.apple.com/app-store-connect
Official instructions for adding an app: help.apple.com/app-store-connect/#/dev2cd126805

- Go to appstore connect
- Click on My Apps > + (to create a new app) > select New MacOS App > Select the appId you registered previously.
- Fill out the App Info section:
  - Enter the name you want your app to appear on the Mac App Store as. Two apps can't have the same name so your preferred app name may be taken.
  - Your app needs a privacy policy URL and (in another section) a support URL.
  - Select the appropriate category for your app. See Category list.
  - Add a license agreement. You can use Apple's Standard License Agreement https://www.apple.com/legal/internet-services/itunes/dev/stdeula/
- Fill out the Pricing and Availability section:
  - Select the price you want to charge for your app. Be aware that Apple keeps 30%.
  - You can also select specific countries to make your app available in. Default is the whole world.
- Fill out the MacOS section:
  - For marketing you can add up to 10 screenshots and up to 3 15-30 second videos. Be aware that these must be in specific dimensions.
  - The App Store Icon is taken from your app's icon.icns file so you don't need to upload them separately.
  - Enter the version number using semantic versioning (e.g., starting with 1.0.0).
  - If your app requires any entitlements enter them here along with the explanation as to why you need that entitlement. The app will be rejected if you request entitlements you don't need. See the Entitlements section below for details.
  - Enter the support URL (required) and a marketing URL (optional).
  - When you upload your app (using the Transporter App - discussed below) you need to select it here.
  - You can add attachments that may be useful to the reviewer such as a video of how to use the app (can be any dimensions).

6) Create an icon set

Reference: electron.build/icons | developer.apple.com/design/human-interface-guidelines/macos/icons-and-images/app-icon

- You need to create your icon in at least two different sizes: 512x512 px and 1024x1024 px saved as png files. But ideally use all the Apple recommended sizes which additionally include 16x16, 32x32, 64x64, 128x128 and 256x256.
- Follow this guide: How to create high resolution icns files. A few things to highlight:
  - Make a folder called icon.iconset (or any name that has .iconset as the extension) to hold the images.
  - Use Apple's naming convention for the image files (e.g., icon_512x512.png, icon_512x512@2x.png, etc.).
  - The @2x files are double the size stated in terms of pixels. However in terms of display, Apple just doubles the density of the pixels instead of doubling the width and height. As such, some of the files will be the same image dimensions but two different files (e.g., icon_256x256@2x.png and icon_512x512.png are two separate files but are the same image size). If you are simplifying the images at smaller sizes, make sure the @2x image is the same as the 1x, just at double the size.
  - When you get down to the small sizes such as icon_16x16.png you'll likely need to simplify the image, otherwise it will just look blurry.
  - Convert the iconset into an icns file in the Terminal from the directory holding the icon.iconset folder with the png images. Enter the iconutil command:
    `$ iconutil -c icns icon.iconset`
  - Put the resulting icon.icns file into the build folder of your electron project.

7) package.json file

The values in italics need to be changed to your info.

```
{
  "name": "AppName",
  "version": "1.0.0",
  ...
  "author": "AuthorName",
  "build": {
    "appId": "com.companyname.appname",
    "productName": "App Name (can include spaces & special characters)",
    "buildVersion": "1.0.0",
    "copyright": "Copyright © 2021 Developer/Company Name",
    "mac": {
      "category": "public.app-category.categoryName",
      "icon": "build/icon.icns",
      "target": "mas",
      "hardenedRuntime": false,
      "entitlements": "build/entitlements.mas.plist",
      "entitlementsInherit": "build/entitlements.mas.inherit.plist",
      "provisioningProfile": "build/MacAppStore.provisionprofile",
    }
  }
  "scripts": {
    "postinstall": "electron-builder install-app-deps",
    ...
  },
  ...
}
```

Only include the postinstall script above if you have native dependencies that need to be recompiled by Electron (see section 13 below).

Reference:

- General configuration including metadata: electron.build/configuration/configuration
- Mac-specific configuration: electron.build/configuration/mac
- MAS-specific configuration: electron.build/configuration/mas

A few things to note:

- Name and productName: The name key will be used as the display name for your app on the user's computer. It cannot contain spaces or special characters. If you want to use spaces or special characters in the name then add a productName key within the build key. You set the name of your app as it appears in the Mac App Store at appstoreconnect.apple.com.
- Version and build number: The version number in package.json should correspond with the version number of your app at appstoreconnect.apple.com. If you upload your app to appstoreconnect but decide to make changes before submitting it for review, you can't delete what you uploaded. Rather you submit a revised app with a different build number. Electron-builder sets the build number equal to the version number in the package.json file which is problematic if you need to submit a second build. To set a different build number, use the buildVersion property.
- Use the same appId as you created for your app at developer.apple.com.
- The category key should align with the category you set for your app in appstoreconnect.apple.com. Mac uses this key in the Finder via View > Arrange by Application Category when viewing the Applications directory. List of possible categories.
- Set your target to "mas".
- Set Hardened Runtime to false. You would set it to true if you want to distribute the app outside the MAS. For more on hardened runtime see developer.apple.com/documentation/security/hardened_runtime_entitlements

Keys you don't need to set because the default is already correct:

- The icon key points to where your icon.icns file is. The default folder and filename is build/icon.icns so you can leave it out if it is located there.
- The type key can be distribution or development. Distribution is the default so you can leave this key out.
- The identity key sets the name of the certificate to use when signing. However, for the MAS build, the signing certificate will be taken from the provisioning profile (covered below). And Electron-builder will automatically use the 3rd Party Mac Developer Installer certificate in your Keychain to sign the pkg file.
- GatekeeperAssess key is set to false by default. Mac's Gatekeeper ensures that apps distributed outside the app store are signed and notarized. Since this build is for the MAS you can leave this at false.

8) Entitlements

Reference: electronjs.org/docs/tutorial/mac-app-store-submission-guide

- Make parent and child entitlement property list files using XML. Put them in the build folder.
- These files correspond to the entitlements and entitlementsInherit keys in the package.json file.

The parent entitlement file:

/build/entitlements.mas.plist

```
<?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.application-groups</key>
    <array>
      <string>TEAM_ID.com.companyname.appname</string>
    </array>
    <!-- Put any entitlements your app requires here. Below is an example -->
    <key>com.apple.security.files.user-selected.read-write</key><true/>
  </dict>
</plist>
```

Child entitlement file (inherits from the parent):

/build/entitlements.mas.inherit.plist

```
<?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>
```

- First thing to notice is app-sandbox is set to true. All apps in the MAS must be sandboxed. That means they don't have access to resources outside the app, unless you specifically request an exception (an entitlement) and it is approved. For example to be able to read and write to files in the user's computer (outside the app) you need to request user-selected files read-write access. If approved your app can read/write files that users have specifically opened or saved using the mac Open or Save dialog box.
- These entitlements must correspond to the entitlements you request for your app in Appstoreconnect.apple.com
- Read about entitlements here.
- The list of available app sandbox entitlement keys are here.
- The application-groups key is an array containing your app identifier (TeamID and appId).

9) Generate a Provisioning Profile

- Go to your apple developer account profiles page developer.apple.com/account/resources/profiles/list
- Click the + sign to add a new profile which will take you to the "Register a New Provisioning Profile" page.
- Select Mac App Store - Create a distribution provisioning profile to submit your app to the Mac App Store. Then click Continue.
- Select your appID and continue.
- Select your Mac App Distribution certificate and continue.
- Give it a recognizable name like "MacAppStore" then download it to your computer and place it in the project's build directory. The file name in this case would be MacAppStore.provisionprofile. This corresponds to the provisioningProfile key in the package.json file. Whatever you name it, during the build process it gets renamed to embedded.provisionprofile.
- This is a binary file. If you want to read it's contents in XML format and you have X-Code's Command Line utilities installed, from the Terminal run:
  `$ security cms -D -i build/MacAppStore.provisionprofile`

10) Build the App with electron-builder

- For building the app there are a few options. We will use the Electron-Builder package. Install it as a global dependency:
  `$ npm install -g electron-builder`
- Build the app for mac:
  `$ electron-builder build --mac`

11) Test the app

- You can test the signature if you have X-Code's CLI utilities installed. To check that your app is signed, go to the directory holding your app (e.g., MyApp/dist/mas) and enter the below command. If the app is signed it will display the details in the Terminal.
  `$ codesign --display --verbose=2 MyApp.app`
- To check that the .pkg file is signed enter the below command from the .pkg file's directory. If it is signed it will display the details in the Terminal.
  `$ pkgutil --check-signature MyApp-1.0.0.pkg`
- The Transporter app (see next section) has built-in checks on your app that will let you know about some (but not all) problems with your app. Simply load the app and send it to appstore connect. Don't worry, this doesn't submit it for review. That's a different step.
- To test the MAS version of the app you need to install it using the .pkg file on another operating system that doesn't have your same credentials. Three options are:
  - Install it on another Mac.
  - Install a second MacOS operating system on your computer to use for testing. That entails creating a second APFS volume. Then you can log into the second (testing) OS, move the app or pkg file in and launch it from there. We won't go into the steps here but set aside a few hours, ensure you have sufficient extra space on your hard drive (maybe 30+GB), and make sure everything is backed up first.
  - Install a Virtual Machine like Oracle's virtual box.
- If you are getting errors you could start by making sure all the basics are working by using this process on a simple Hello World Electron app. The Github example app link at the top is to such an app. Just add in your developer and app datails and provisioning profile.

12) Upload the pkg file

- There are three different tools you can use to upload the pkg file: help.apple.com/app-store-connect.
- The Transporter app can be downloaded free from the Mac App Store.
- Open the Transporter app > Drag and drop your Electron app's pkg file into the Transporter app > Deliver.
- If no errors it will send the app to your developer account.
- Go to appstoreconnect.apple.com and click MyApps > Prepare for Submission > Build + icon > Select the uploaded file - Done
- Note: the build may have a "Missing Compliance" warning next to it. If so, when you submit the app you will be asked about any encryption in your app, so just answer the questions given.
- If you are ready to submit then hit submit for review.
- If you decide to make additional changes before submitting for review, you have to load another .pkg file through the Transporter app with a different build number.

13) Native Node NPM packages

If you installed one or more native node modules as a local dependency in your app such as Sqlite3 to use a sqlite database, their executable file(s) need to be code signed in addition to the overall app and the .pkg file. Native node modules are npm packages that, like Node.js core modules, include C or C++ code. These modules have to be compiled after installation with electron-rebuild. Ref: Electron Docs: using native node modules

This presents a problem for MAS builds. Apple can't read signatures inside the built binary app. To solve this, unpack the native node dependencies by adding asarUnpack to the mac key in your package.json file.

```
"build": {
  "mac": {
    "asarUnpack": [
      "**/*.node"
    ],
    ...
  }
}
```

That will pull all node executable files (which have the extension .node) out of your app's binary file (called app.asar) and into a sibling folder called app.asar.unpacked. Apple can then read the dependency's signing certificate.

After building the app you can confirm that the .node files are signed:

- Find the path to the .node executable file(s) in Finder by right clicking the built app in the dist/mas folder > Select "Show Package Contents" > Drill down to Contents/Resources/app.asar.unpacked/node_modules... until you find the executable file (ending in .node)
- In the Terminal cd to the built app: `$ cd dist/mas`
- Run the XCode CLI command:
  `$ codesign --display --verbose=2 MyApp.app/Contents/Resources/app.asar.unpacked/node_modules/package_name/path/to/binary_file.node`
  Sqlite3 example:
  `$ codesign --display --verbose=2 MyApp.app/Contents/Resources/app.asar.unpacked/node_modules/sqlite3/lib/binding/napi-v3-darwin-x64/node_sqlite3.node`

So now you won't get the signing error anymore. But at least in my case I encountered a new, unexpected issue. The app.asar.unpacked folder permissions/ownership was being changed when the app was flattened into the .pkg file. And when I put it in the Transporter app and hit send it came back with an error saying:

```
ERROR ITMS-90255: The installer package includes files that are only readable by the root user. This will prevent verification of the application's code signature when your app is run. Ensure that non-root users can read the files in your app.
```

If you run into this error here is the fix:

- After you run the mas build, keep the *MyApp*.app file but delete the *MyApp-1.0.0*.pkg file.
- Change the permissions to allow read and execute access to all directories and files in the app.asar.unpacked folder. From *MyApp*/dist/mas run:
  `$ sudo chmod -R 755 MyApp.app/Contents/Resources/app.asar.unpacked`
- Then create a signed pkg file
  `$ npx electron-osx-flat "MyApp.app" --verbose`

Note, this signing node dependencies issue doesn't affect apps distributed outside the Mac App Store as long as they aren't sandboxed. If distributing outside the Mac App Store you just need to include the below two entitlements:

```
<key>com.apple.security.cs.allow-unsigned-executable-memory</key><true/>
<key>com.apple.security.cs.disable-library-validation</key><true/>
```

Comments:

19 Comments Tech and Startup Disqus' Privacy Policy 1 Login
Favorite 1 Tweet Share Sort by Best
Join the discussion…
LOG IN WITH OR SIGN UP WITH DISQUS ?
Name

vaibhav · a year ago
Thanks for the write up. I am able to successfully release my app on app store.
1 · Reply · Share ›

Johannes M · a year ago · edited
Thank you very much for this great write up. This sums it up very nicely. I wonder however if you or somebody else ever encountered the following error:
```
Unable to process application at this time due to the following error: Invalid Provisioning Profile. The provisioning profile included in the bundle com.super-productivity.app [com.super-productivity.app.pkg/Payload/superProductivity.app] is invalid. [Missing code-signing certificate.]
```
?

I've been stuck with it for over a year now and was unable to find a solution despite putting a lot of effort into it. Help would be much much appreciated! I also opened a github issue.
1 · Reply · Share ›

Max K → Johannes M · a year ago
Same issue.
Reply · Share ›

Tamas Geschitz · 2 years ago
Once I have the MAS, should I be able to start it? It always crashes with CODESIGNING error.
I also tried to create a VM with a MacOS Catalina and run it there, but I see the same error. I can run the installer (pkg) and it works fine, but the app cannot start due to the code signing problem.

I don't understand this. All verification tools (codesign, pkgutil etc) report success. If I start up the installer I can see that the certificate is valid (lock icon in the corner). Transporter app doesn't find any problems. Yet, I can't run the app, this is what I get immediately:

Namespace CODESIGNING, Code 0x1

What could be wrong? I'm on Catalina and am using electron 6.1.9 and electron-builder 21.2.0
1 · Reply · Share ›

steve981cr Mod → Tamas Geschitz · 2 years ago
To isolate the issue try making a version of the example app I put on Github (link is at the top of the article) and see if that works for you. When I was prepping my app for MAS submission there were two issues I was dealing with, including the one you are having. Once I got a super basic app pkg working I was able to work on the second issue (installing Node dependencies). Also, I am using electron-builder version 22.3.2. It's a step up from your version, but not the latest. Seems they made a breaking change for the latest version.
Reply · Share ›

steve981cr Mod → Tamas Geschitz · 2 years ago · edited
One thing is you have to build the app on a different machine than the one you install it on. Or to be more precise on a different operating system. It sounds like this MIGHT be the issue... except you tried on a Virtual Machine and it still doesn't work. But for sure it won't work if you build it then install it on the same Operating System instance.
I'm not necessarily recommending this but in my case I have some expensive 32 bit software that won't run on Catalina, I partitioned my Mac to two operating systems, Mojave and Catalina. That's a fair amount of work and hogs up a lot of disk space but one side benefit is I can build the app on Mojave, log out, log into Catalina, then install the app and it works.
Ideally, if you have access to a different mac, try building it on one, then installing it on the other.
Reply · Share ›

Tamas Geschitz → steve981cr · 2 years ago
Thanks Steve for the info, I'm gonna give this a try.
This is so strange, since I'm pretty sure this worked fine before. I mean, I was able to build and run the app without error.
Anyway, so the CODESIGN error is normal and should be expected if I try to run it on the builder machine?
It should print some more informative message..
Reply · Share ›

steve981cr Mod → Tamas Geschitz · 2 years ago · edited
Yeah, you can't build the pkg file and install it on the same operating system. Not sure why that is or if that behavior started with Catalina. And there's no specific warning about it, just the general one like you got.
One thing though, you did the CLI checks in step 11 and they passed, and the transporter gives you no errors? If so the pkg part may be fine. Still a good idea to confirm though.
Reply · Share ›

Channa Kao · a year ago
Thank you very much but I have a issue in entitlements.plist I set <key>com.apple.security.app-sandbox</key><true/> it show error codesign --sign 437D133F3F6F92C4F41239CB6D82A2AD5DDDCFBED37 --force --entitlements /var/folders/c0/gmwzy_rx0299f4n94z0pngpm0000gn/T/tmp-entitlements-31bd-0.plist /Users/pos/Desktop/pos/pos-desktop-app/build/mas/pos.app
| Could not create string from entitlements data
Reply · Share ›

Tamas Geschitz · 2 years ago
Update: the Transporter tool seems to generate false positives sometimes... this is just great.
I just created another build (the only change is productName property added to config) and now it says there are 3 issues....

It complains about bundle id of the electron frameworks and its tools..

ERROR ITMS-90277: "Invalid Bundle Identifier. The application bundle contains a tool or framework Electron Helper (GPU) [hu.icell.etcd.pkg/Payload/ETCD Manager.app/Contents/Framew... Manager Helper (GPU).app] using the bundle identifier 'hu.icell.etcd.helper.(GPU)', which is not a valid bundle identifier."

How can I fix this?
Reply · Share ›

steve981cr Mod → Tamas Geschitz · 2 years ago
You are saying the issue appears only when you add a productName property? If you remove it again then Transporter doesn't complain? I tried to recreate the issue using the electron-quickstart-mas app with my own app's info plugged in and I don't get an issue.
Reply · Share ›

Tamas Geschitz → steve981cr · 2 years ago
Sorry Steve,. my bad. It's an electron-builder issue, I have to upgrade it.
But it's true that the Transporter app sometimes reports success even if there are errors in the app. It prints all is fine, and then when I hit deliver I get an email from AppStore connect with the error list.. strange.
Reply · Share ›

Tamas Geschitz · 2 years ago
When I import the certificates to Keychain they don't appear under Login -> my certificates. They are are located in System -> Certificates.
Could this be an issue?
Reply · Share ›

Tamas Geschitz · 2 years ago
Btw thanks for this article! It's very detailed and useful indeed. Actually I was able to to build a working MAS package with the previous MacOS version (Mojave), but since I upgraded to Catalina, all I get is a code signing error.
Reply · Share ›

steve981cr Mod → Tamas Geschitz · 2 years ago
Catalina is kind of a sea change OS for Mac including around Mac App security. Stuff that worked fine for years suddenly now throws fatal errors on Catalina.
Reply · Share ›

junjie shao · 2 years ago
Thanks for the great article! I cloned your sample repo and made the necessary edits (appID, provision files, etc.). It runs in dev mode but when I build it (in mas-dev mode) and run, it crashes on launch. Any ideas why this failed?
I'm runing on 10.15.1
Reply · Share ›

steve981cr Mod → junjie shao · 2 years ago · edited
It should work. I've tested it on Mojave and Catalina (10.15.2). Try walking through the instructions steps 3, 4, 5 and 9 covering the certificates, appId and provisioning profile. See if they match what you have done.
Also, if you build it with target mas-dev you need to have a Mac Developer certificate which is not the same as the 3rd Party Mac Developer Application certificate. Try building it with target "mas" and see if that works.
Reply · Share ›

junjie shao → steve981cr · 2 years ago
I understand the differences between mas and mas-dev, and I'm pretty sure that I was using the correct certificate. The point is that if I only build with mas, then there is no way to validate the build. You have to submit it to apple and wait for their response. It would be better if I can at least have a working mas-dev version before submitting.
Reply · Share ›

junjie shao → junjie shao · 2 years ago
Just tried it again with a clean electron-quick-start setup, on 10.14. I set it to "mas" and it did run. So Apple or electron changed some cofigs? In the past only the mas-dev version can run locally.

Anyway this is very helpful, I'll keep diggig into it, thanks a lot!
Reply · Share ›

Subscribe Add Disqus to your site Do Not Sell My Data DISQUS