Installer for macOS

Manual

Note: This is for OS X Installer packages which are to be distributed outside the Mac App Store.
Packages for submission to the Mac App Store follow different rules.
If you have any questions then please don't hesitate to write to office@kamgam.com.

Table of contents

Overview (READ THIS FIRST)

Before any app from the internet is startet MacOS checks whether or not this software is to be trusted. In order for your software to be trusted you will have to sign and notarize the installer and the app within it. This tool provides a GUI to do that.

These are the steps in detail:

1. APP SIGNING
   You will have to sign the app bundle exported by Unity (.app).
2. APP NOTARIZATION
   The signed app then has to be upladed to Apple servers for notarization. Your whole .app will be uploaded and processed in an async queue. This can take from 10 seconds up to a few hours. Usually it takes just under a minute. Though be prepared to wait for results (see APP CHECK).
3. APP CHECK
   You need to wait for the notarization to finish and then check if it was successful. Only after the app has been successfully notarized you can create a notarized installer from it.
4. APP STAPLING
   Once the app has been notarized successfully you need to staple it (meaning add some notarization data to the app to make it work even if the user is offline).
5. INSTALLER CREATION (AND SIGNING)
   The installer has to be created and signed. It will contain the app and some extra files (like a post-install script, your logos, text, ...).
6. INSTALLER NOTARIZATION
   The signed installer then has to be upladed to Apple servers for notarization. Both the APP and the INSTALLER need to be notarized separately.
   The reason is that the installer needs to be trusted while installing the app and the app needs to be trusted when it is started (thus two notarizations are needed).
7. INSTALLER CHECK:
   Only after the installer package (.pkg) has also been successfully notarized you will have a fully functional installer which can be used on any mac.

It is quite an involved process and things may go wrong. Please come back to this manual for more details if needed.

Please proceed with checking the prerequisites.

Prerequisites

This manual assumes you are building on a mac and that you have:

1) macOS 10.13.6 or later
Building a new app for notarization requires macOS 10.13.6 or later.

2) An Apple developer account
You can join the Apple Developer Program here: https://developer.apple.com/programs/

3) A valid certificate for signing APPLICATIONS
See Prerequisites for Signing for more details.

4) A valid certificate for signing INSTALLERS
See Prerequisites for Signing for more details.

5) Xcode 13 or later (or Xcode 14 after fall 2023)
You can get Xcode here: https://developer.apple.com/xcode/

If you have multiple Xcode versions installed then please use
„`sudo xcode-select -s /path/to/Xcode13.app`"
to select the appropriate xCode version.

NOTICE: After fall 2023 notarization requires Xcode 14 and macOS 12.4
Source: https://developer.apple.com/videos/play/wwdc2022/10109/?time=90

6) Packaging & signing binares
Usually these binaries come bundled with macOS or the Xcode dev tools and no action by you is required.

- pkgbuild – to package the .app into a .pkg
- pkgutil – to verify the package signatures
- productbuild – to build the installer
- productsign – to sign the installer .pkg
- codesign – to sign the .app within the .pkg and verify the signatures
- spctl – to verify the notarization succeeded
- xcrun + notarytool – to notarize .pkg files
- sed – to replace text in templates
- plutil – to modify .plist files

7) Ensure your build machine has network access
See the Apple Notarization Documentation for more details.

8) **AFTER INSTALLATION of the tool please make sure you have execute permissions on the shell scripts.**

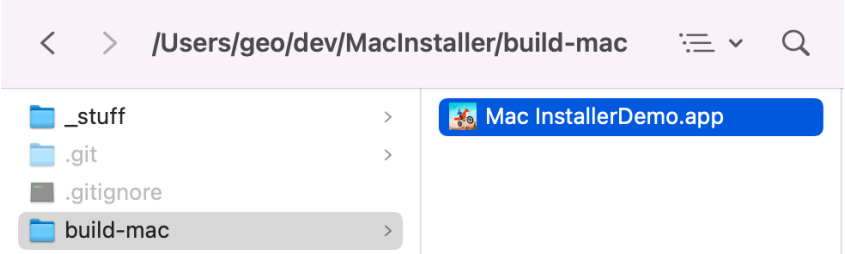
You can ensure this by executing these commands within the „Assets/Kamgam/MacInstaller/MacInstallerBuild/“ folder:

```
chmod u+x *.sh
```

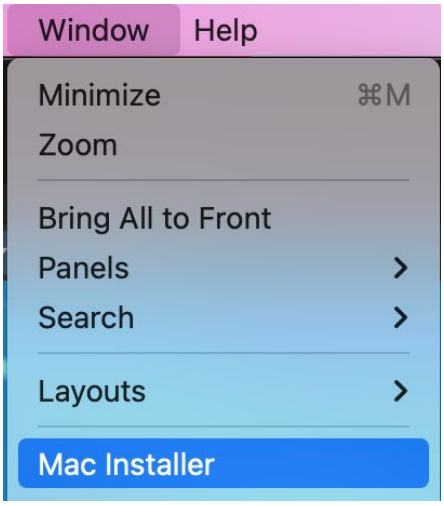
Making an Installer

As mentioned in the overview making an installer takes multiple steps.
! Please make sure you have execute permissions on the shell scripts, see prerequisites.

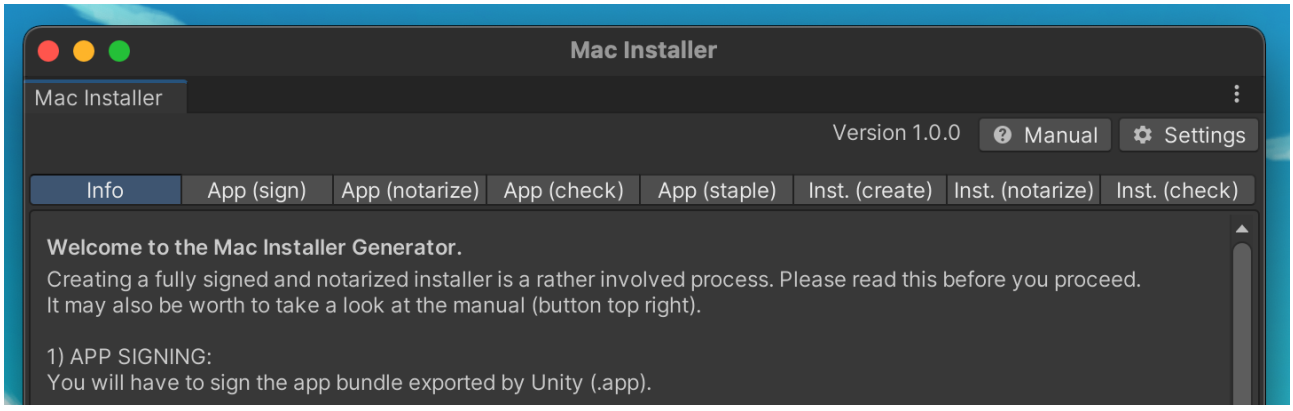
First you will have to build your app into an app bundle (also called „application“.
A bundle is the default output of the Unity build process for macOS (it's a folder ending with „.app“).



Once you have the app you can start with making the installer.
Go to „Window > Mac Installer“.



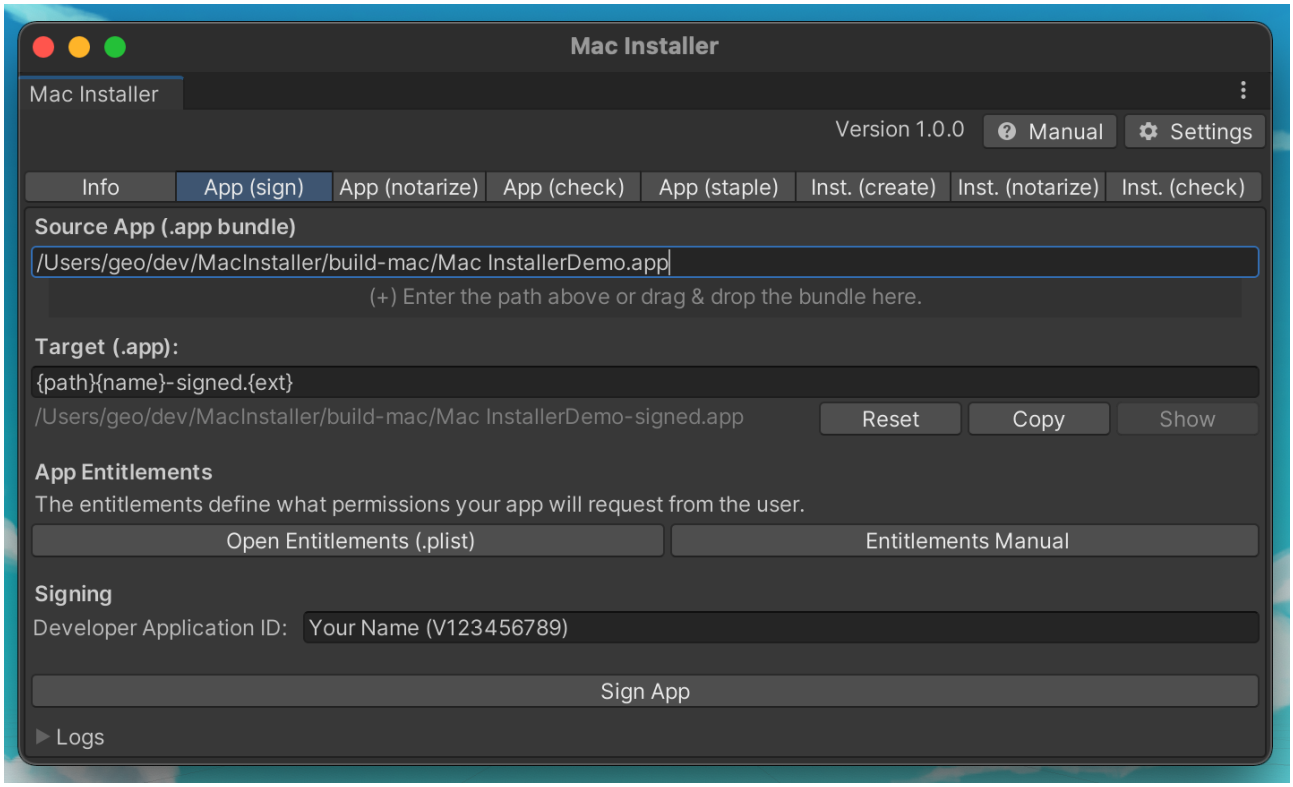
The info screen will be the first thing you see. It contains an overview and some useful links.



You should progress through all the steps from left to right.

1) App Signing

First you will have to sign the app which Unity has built for you.



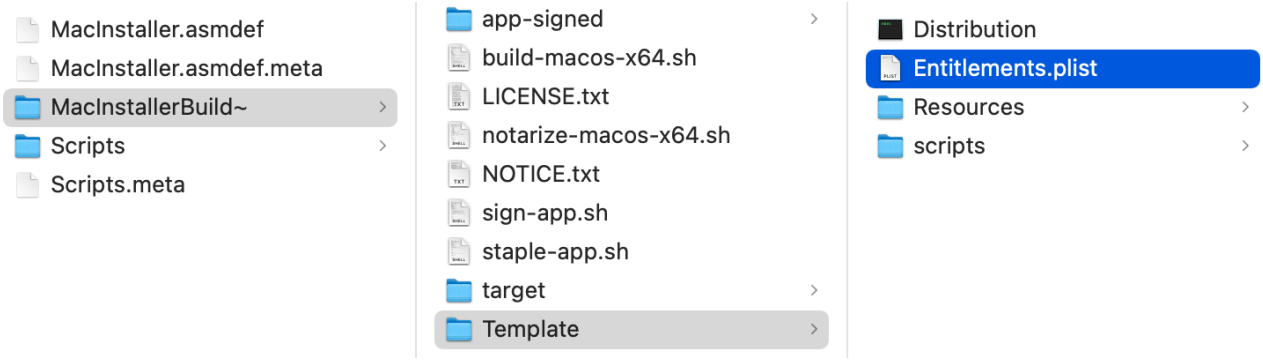
Source App: The „Source App" has to point to the „.app" bundle which you have built. Usually it takes that info from the last build report automatically.

HINT: you can drag & drop your .app folder on that input field (no need to type it all in).

Target: The „Target" path will define where the signed app will be stored. There are some handy placeholders „{..}" but you can also enter a custom path. It shows the resolved path below.

App Entitlements: You can specify some permissions for your app in an Entitlements.plist file. By default no extra permissions are set. To modify them click the „Open Entitlements" button. To find out more about it please refer to the Apple Manual on Entitlements.

The Entitlement.plist is part of the MacInstallerBuild~/Template folder. In there you can find all the template files used to build the app and installer. These template files are copied and then used for packaging the build (see „Modifying Templates").



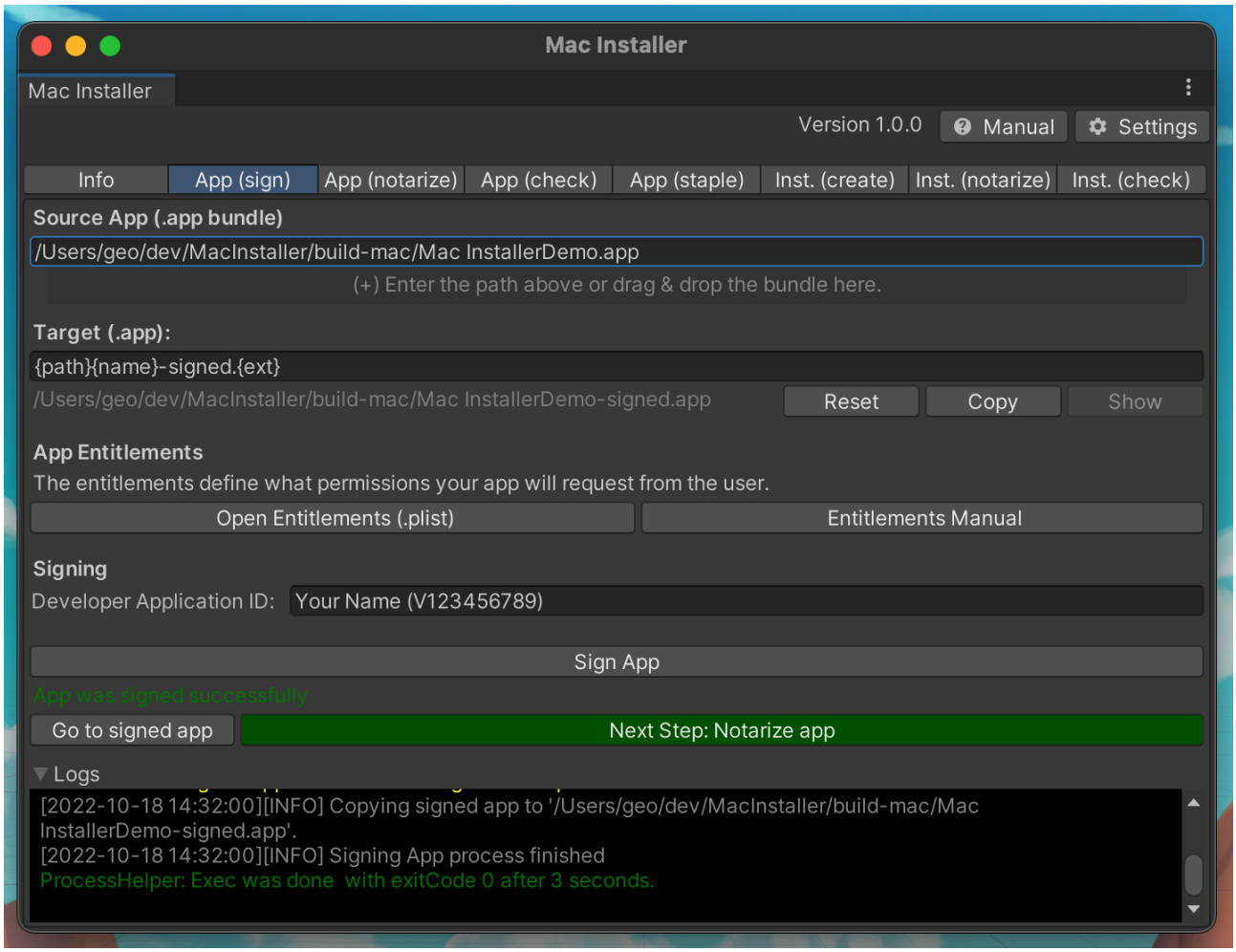
Signing: Enter your Developer Application ID. You can copy it from your „Developer ID Application" certificate.

HINT: If you need help getting one please read the „Prerequisites for Signing" section.

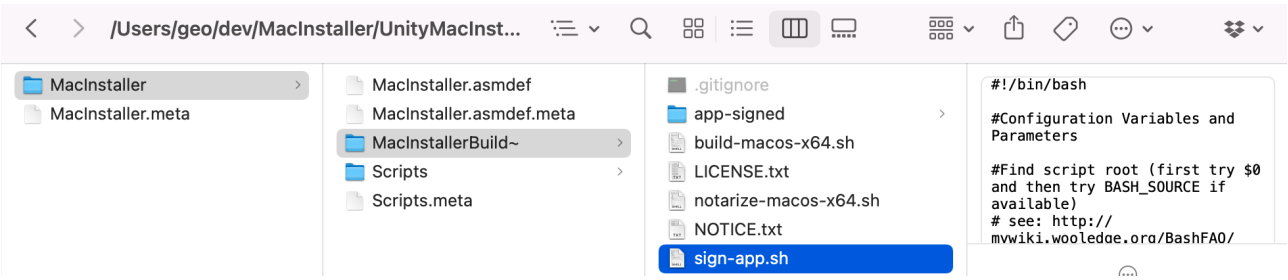
Pressing the „Sign App“ button will start the signing process. Depending on your app size this can take from few seconds up to a couple of minutes.

Once it has completed successfully you will see a green „Next Step“ button.

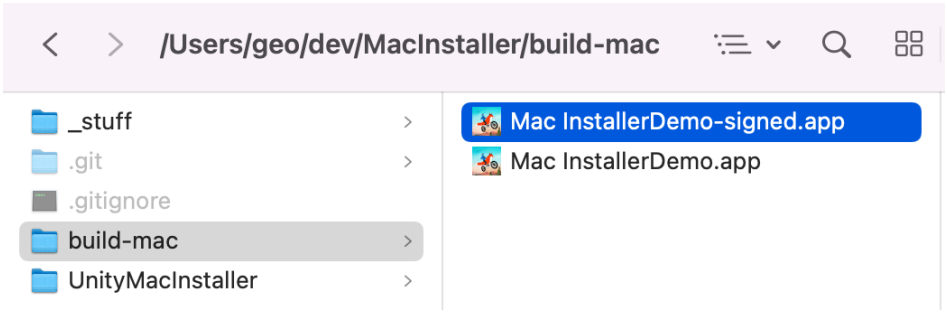
In case of errors you can look at the logs below. They will show the output of the commands used to sign the app.



HINT: The actual signing will be done by a bash script located in the MacInstallerBuild~ directory. Take a look at it in case you want to investigate the signing process.



If the signing was successful you will have a „...-signed.app“.



Now we can move to the next step. The notarization.

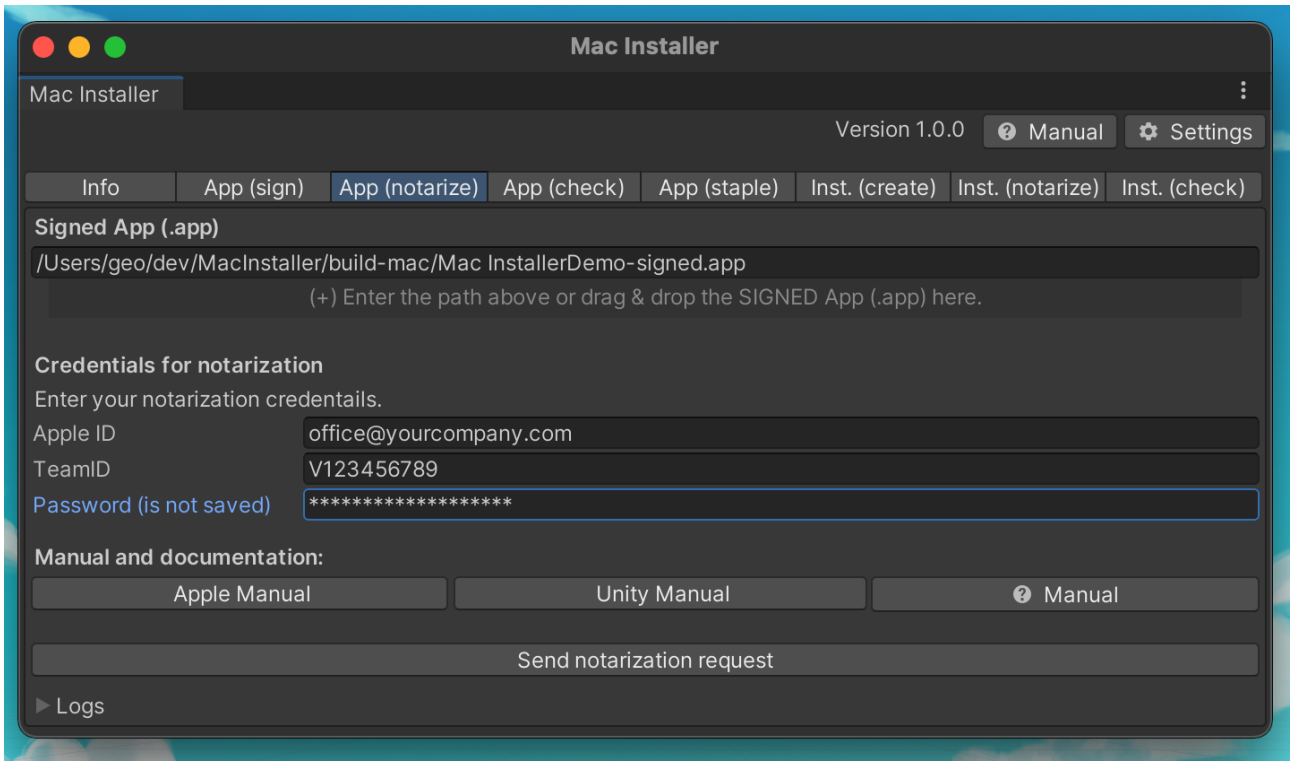
Press the „Next Step“ button.

2) App Notarization

The notarization is necessary so that the Apple servers know your app is trustworthy.

The signed app has to be upladed to Apple servers for notarization. Your whole .app will be uploaded and processed in an async queue. Usually it takes just under a minute. Though be prepared to wait for results, it may take longer.

Depending on the size of your app and your upstream speed the upload will take a while. Please be patient.



Signed App: The path to the signed .app from the previous step. Usually that is copied over automatically.

Credentials for Notarization:

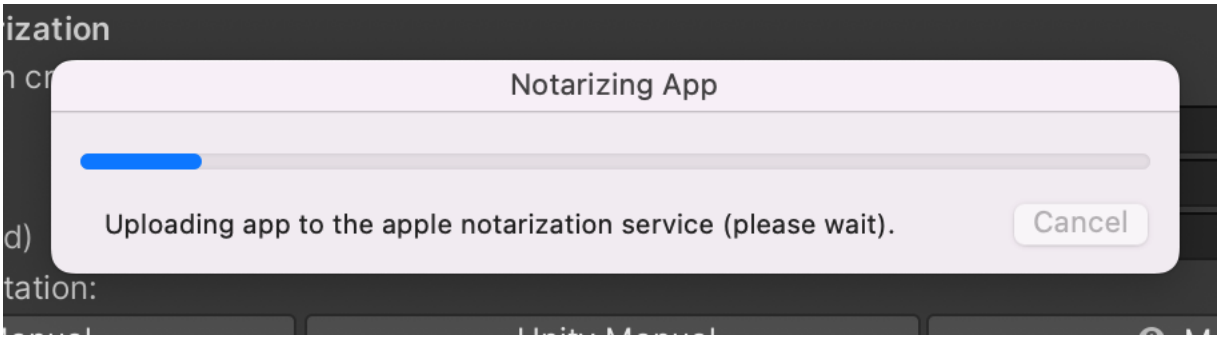
Please refer to the „Prerequisites for Notarization" section if you do not yet have notarization credentials.

Apple ID: Your developer account Apple ID. Usually that's your email address.

Team ID: The developer Team ID. Usually that's the same as in the developer certificate.

Password: Your notary app-specific password. Usually in the form of „xxxx-xxxx-xxxx-xxxx".
NOTICE: „app-specific" in this context does not mean you need an extra password for every app you build. Instead it refers to the Apple „app-specific" password mechanism.

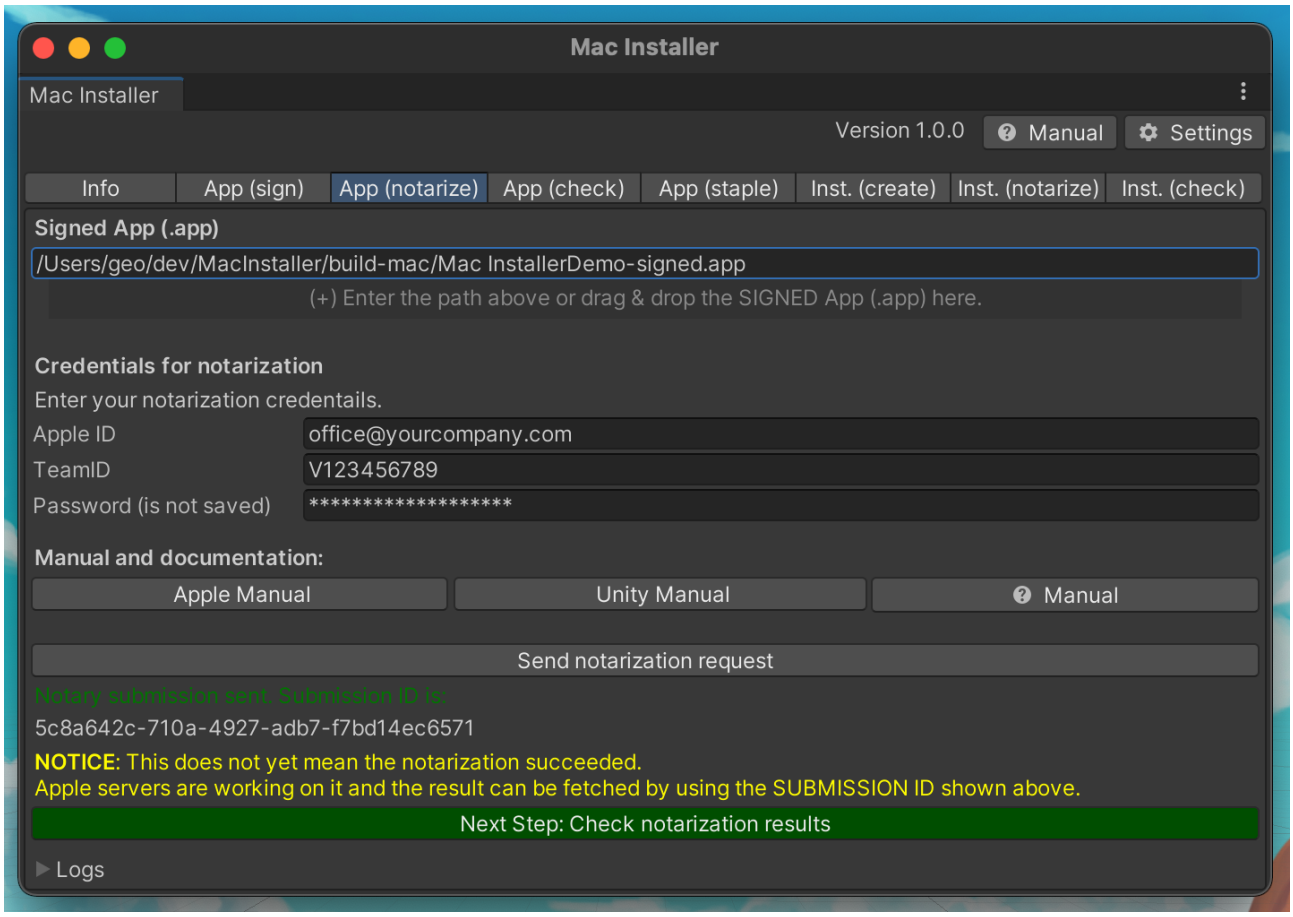
Pressing the „Send notarization request“ button will start sending your app to Apple for notarization.



This may take a while. Please be patient.

Please refer to the documentation for notarizing custom installer packages if you need more info on how things work behind the scenes.

Once the upload has completed successfully you will see the green „Next Step“ button.



NOTICE: Having successfully uploaded your app for notarization does NOT mean it has been notarized.

Now would be the right time to grab a cup of coffee as you will have to wait until the Apple servers have analyzed your app. Usually this takes just about one minute but it may take longer (up to 24 hours according to Apple).

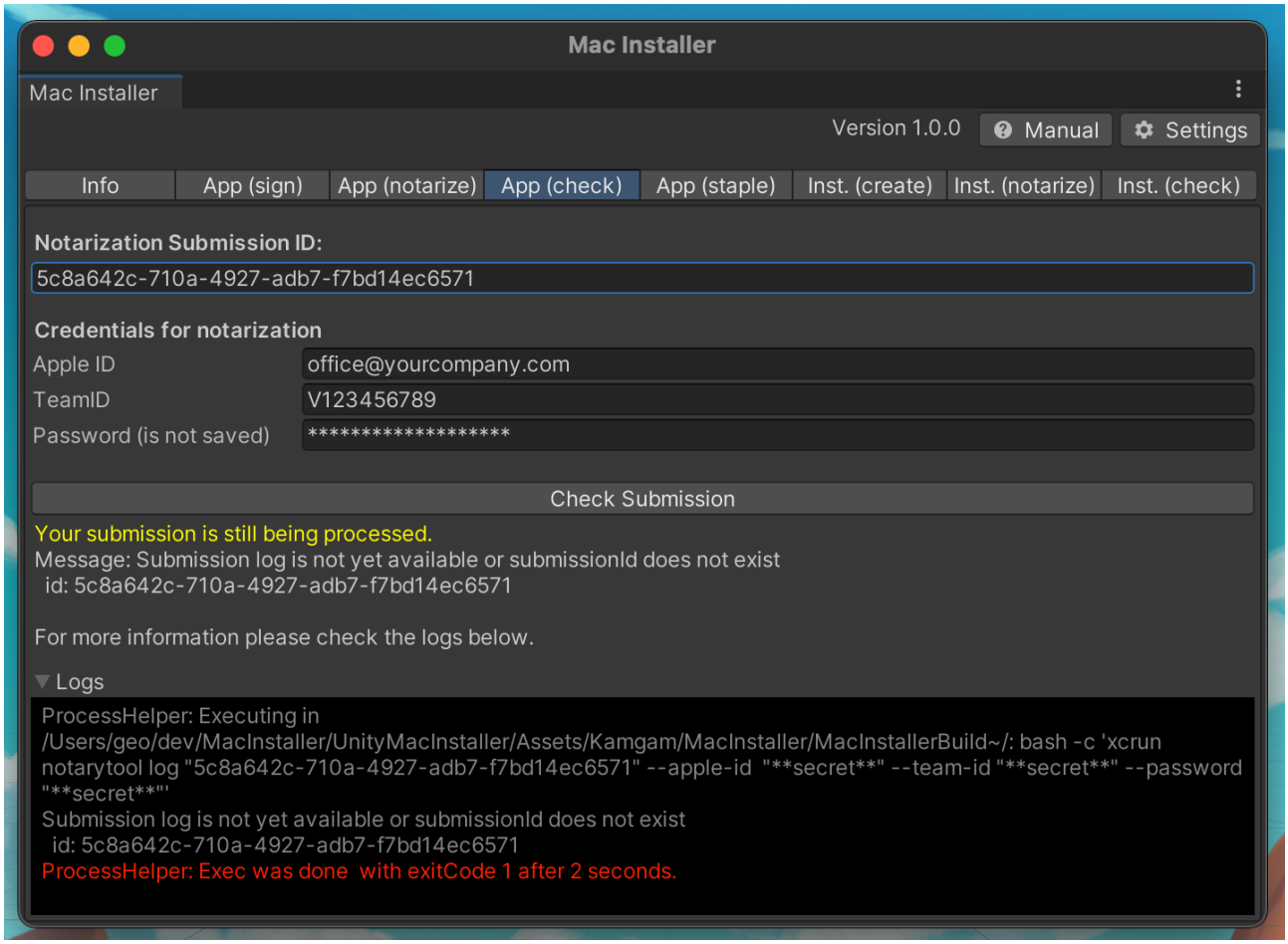
In order to check the status of you submission you can go to the next step „App (check)“.

Press the „Next Step“ button once you are ready to proceed.

3) App Check (Notarization)

You need to wait for the notarization to finish and then check if it was successful. Only after the app has been successfully notarized you can create a notarized installer from it.

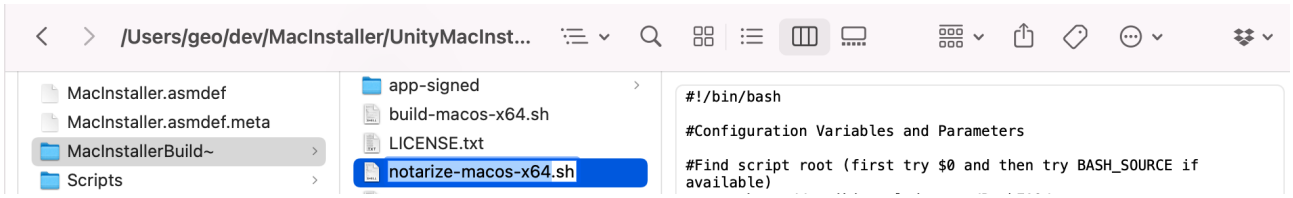
This is how the screen will look like if your App has not yet been processed by Apple:



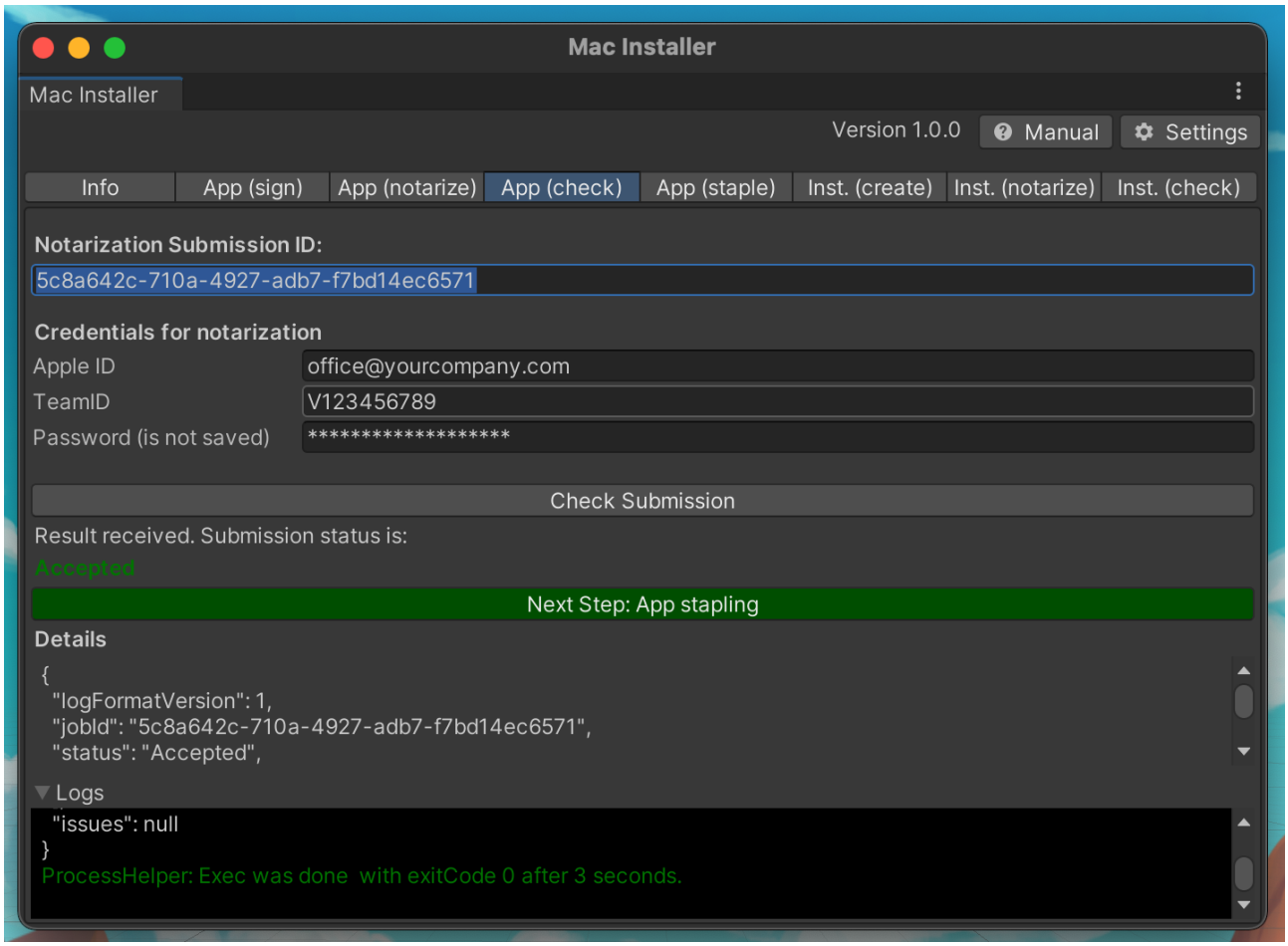
Usually it takes just a few minutes to notarize an app. If it takes longer then maybe something went wrong or the Apple servers are very busy.

If something went wrong then the logs will contains detailed informtion about that in the JSON format.

HINT: The notarization is be done by a bash script located in the MacInstallerBuild~ directory. Take a look at it in case you want to investigate the process.



After a while you should be able to get a positive response from the notarization servers.



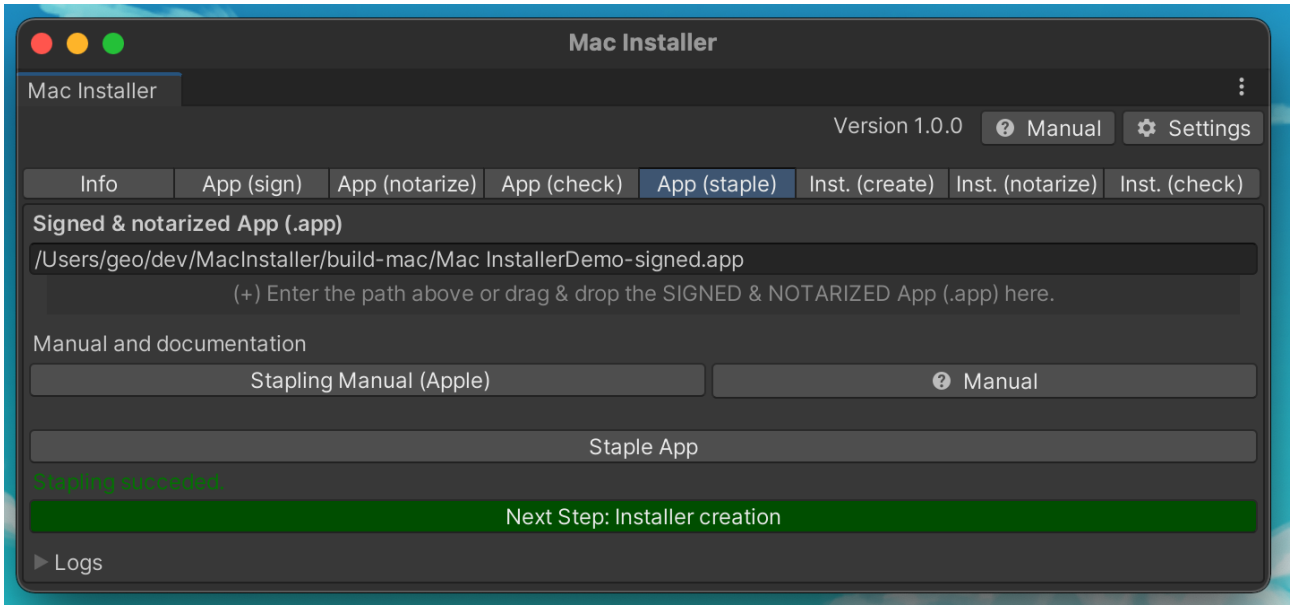
If you are having trouble getting the notarization to work then please refer to the Apple Notarization Manual.

Press the „Next Step“ button once you are ready to proceed.

4) App Stapling

Once the app has been notarized successfully you need to staple it (meaning you need to add some notarization data to the app to make it work even if the user is offline).

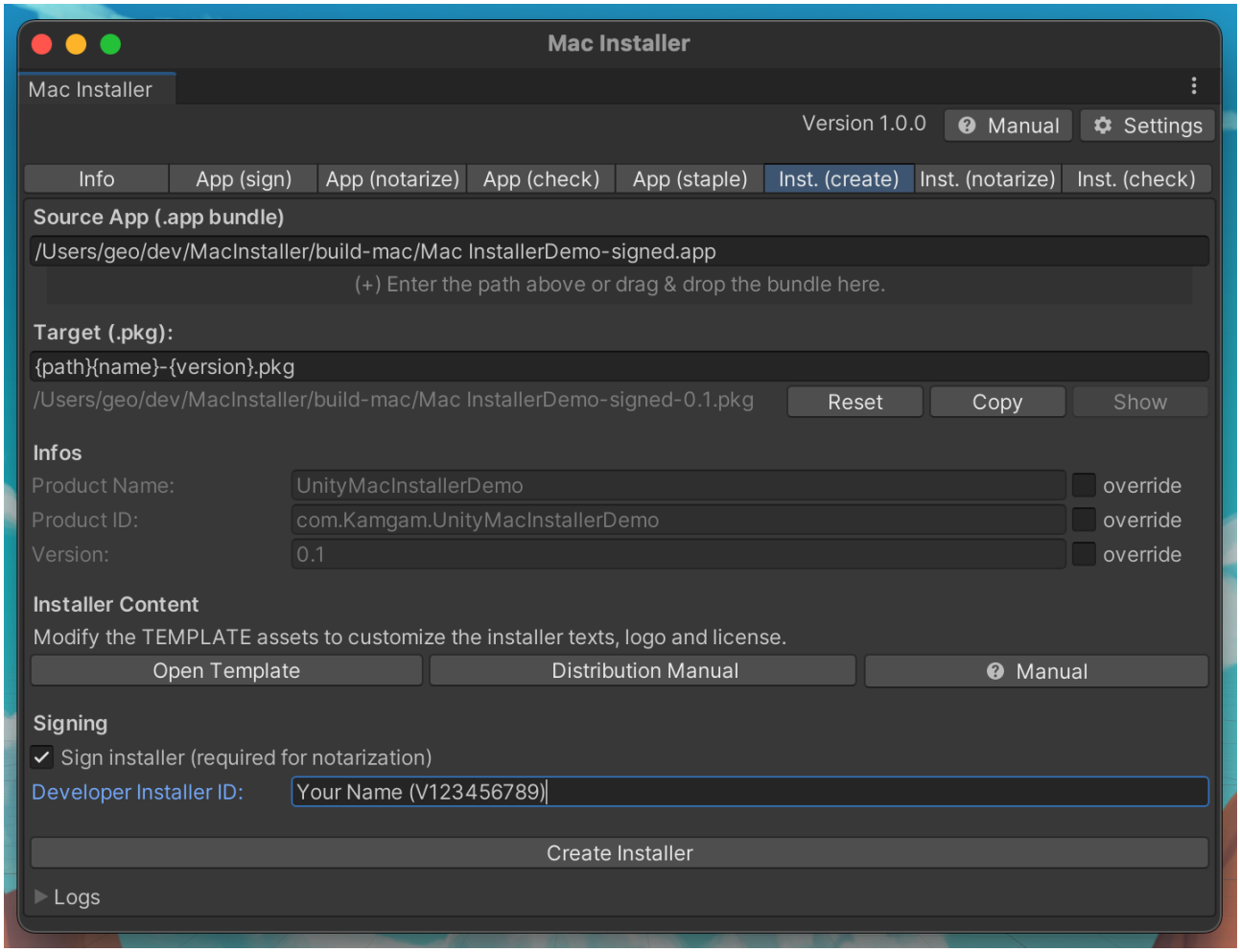
The only input required for stapling is the path to the notarized app. In your case that's the signed app. Usually the path is set automatically by the previous notarization step.



Press the „Next Step" button once you are ready to proceed.

5) Installer Creation and Signing

The installer has to be created and signed as well. It will contain the notarized app and some extra files (like a post install script, your logos, license text, ...).

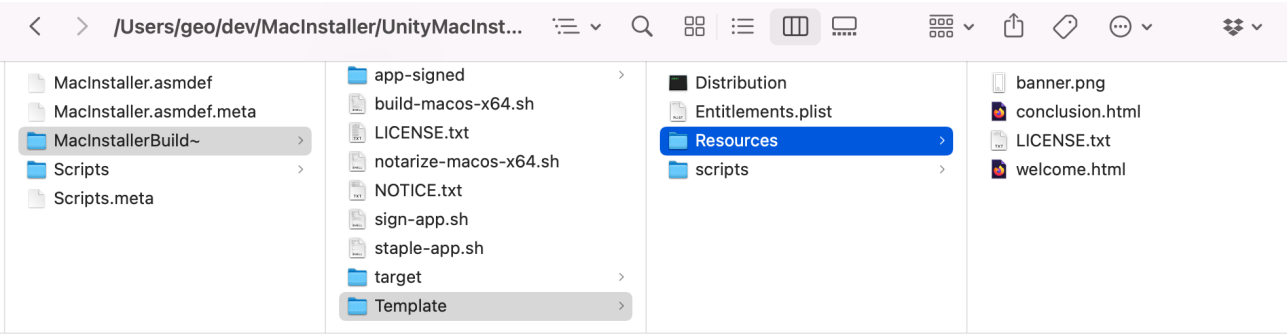


Source App: The path to the signed and notarized app. Usually that's set automatically by the previous step.

Target: The path where the installer package (.pkg file) should be created.

Infos: Some of the infos for the installer are taken from the Unity build settings. If you want you can override them here.

Installer Content: The texts, logos, etc. used for creating the installer are located in the Template folder. Edit them to change the contents of your installer.



There are some strings that will be replaced in any of the *.html files.

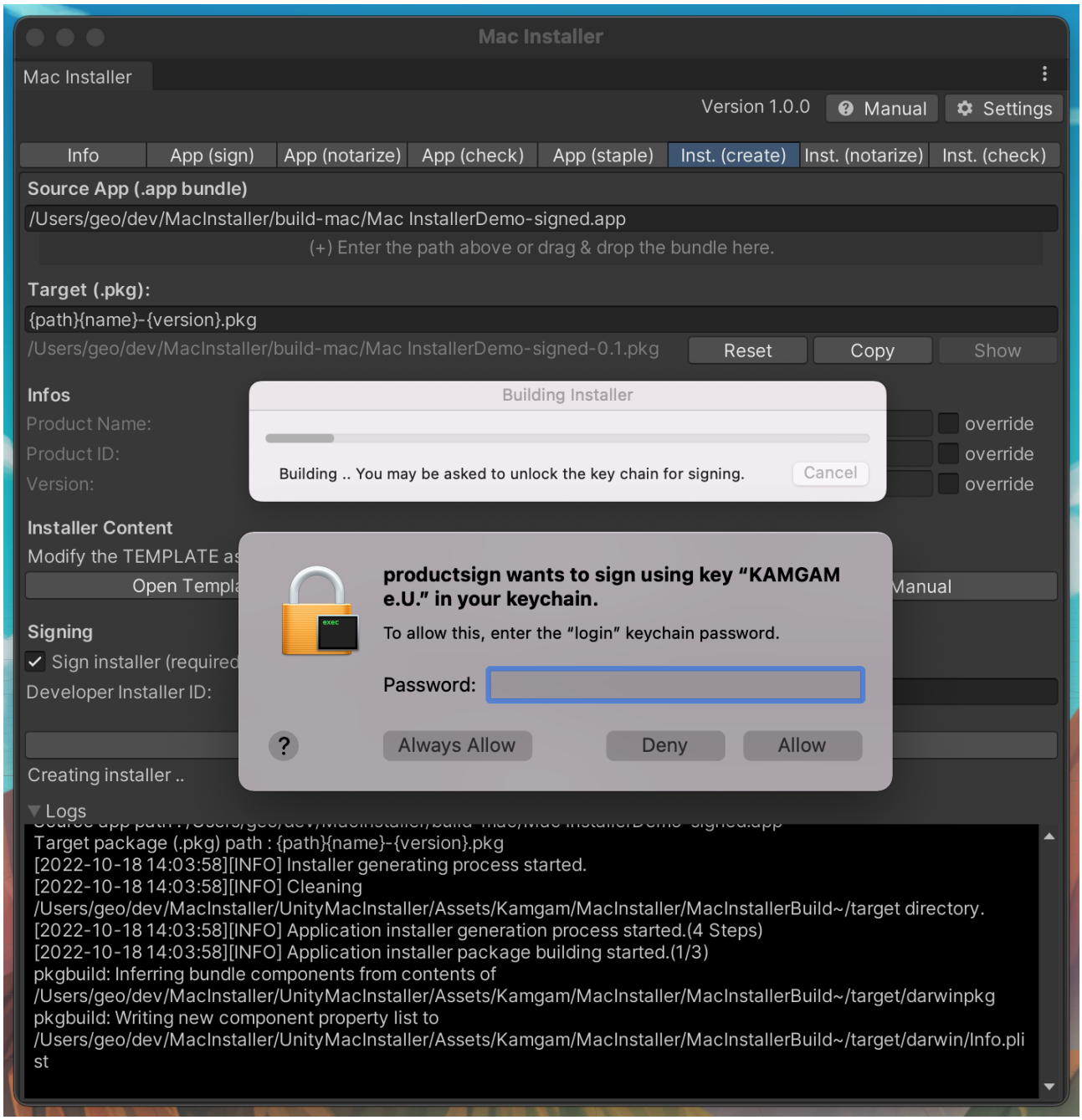
__PRODUCT__ => Will be replaced with the Product Name from the Infos section. __VERSION__ => Will be replaced with the Version from the Infos section.

Signing: Enter your Developer Installer ID. You can copy it from your „Developer ID Installer“ certificate.

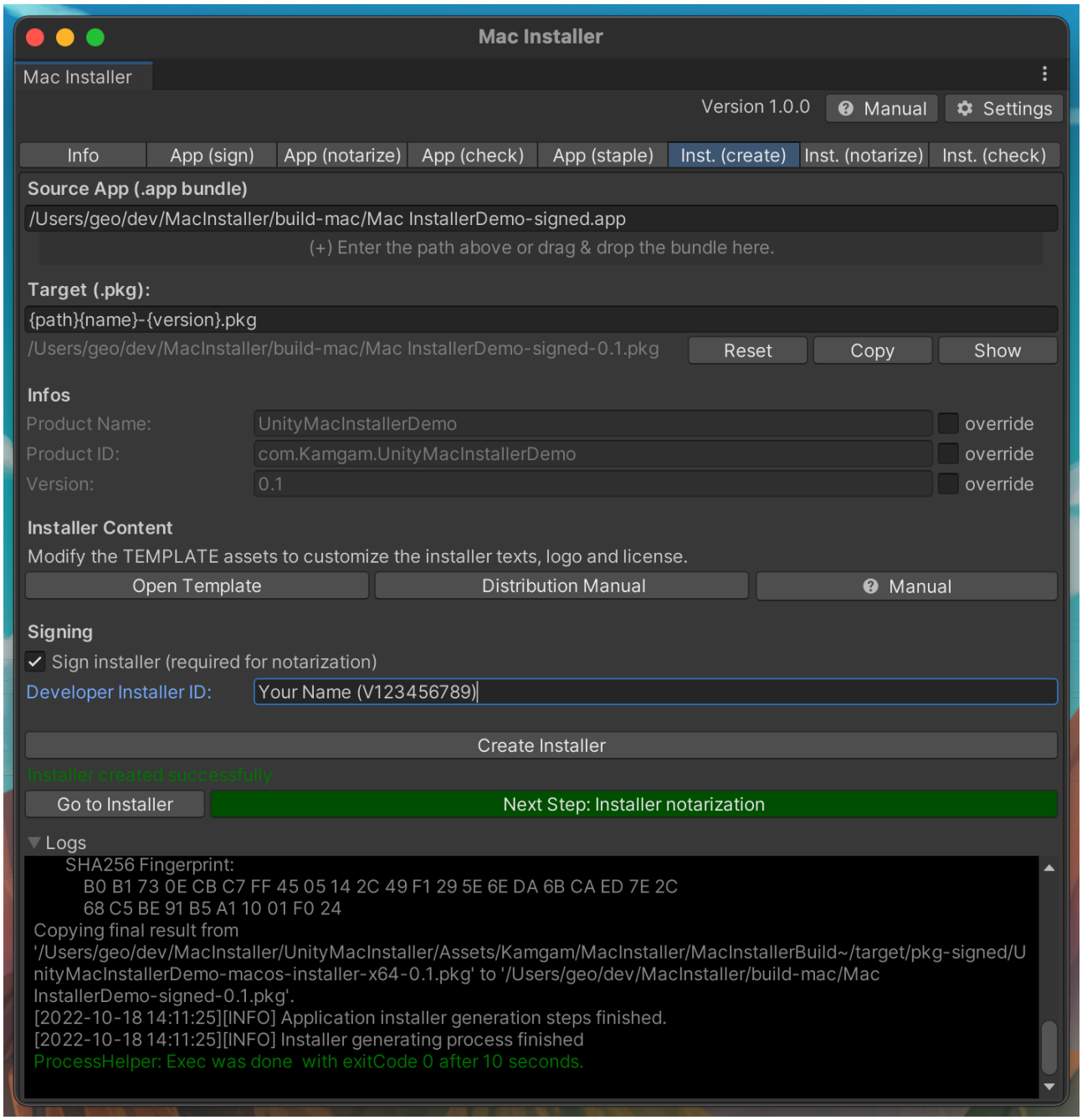
HINT: If you need help getting one please read the „Prerequisites for Signing“ section.

NOTICE: You can disable the signing process but then you won’t be able to notarize your installer.

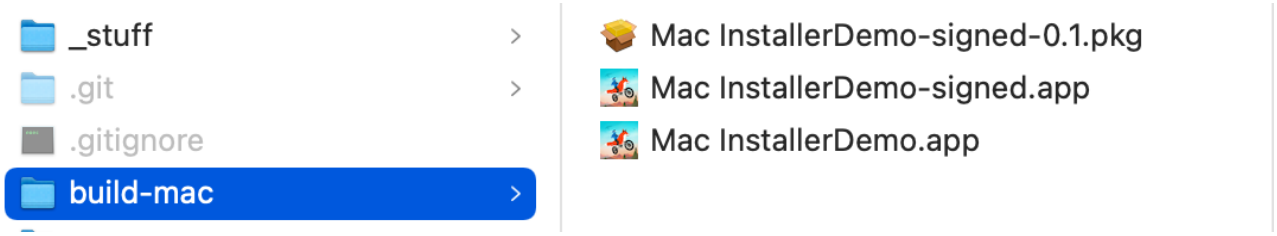
During the signing process you may be asked multiple times to enter your password to allow the „productsign“ command access to the certificates in the keystore.



Once the installer has been created you will see the green „Next Step“ button.



Now you will have an installer package (.pkg) right next to your app.



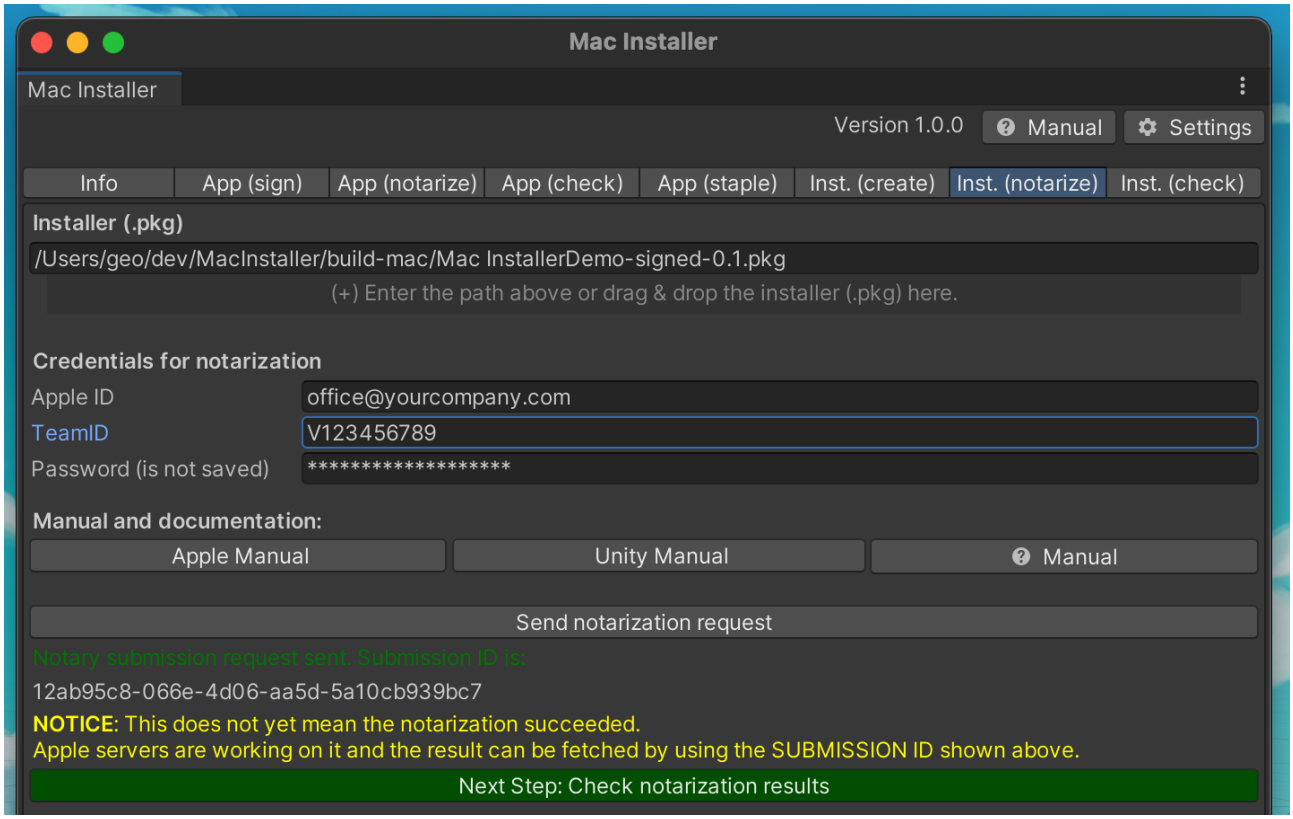
But we still need to notarize this package.

Press the „Next Step“ button once you are ready to proceed.

6) Installer Notarization

The signed installer has to be upladed to Apple servers for notarization.

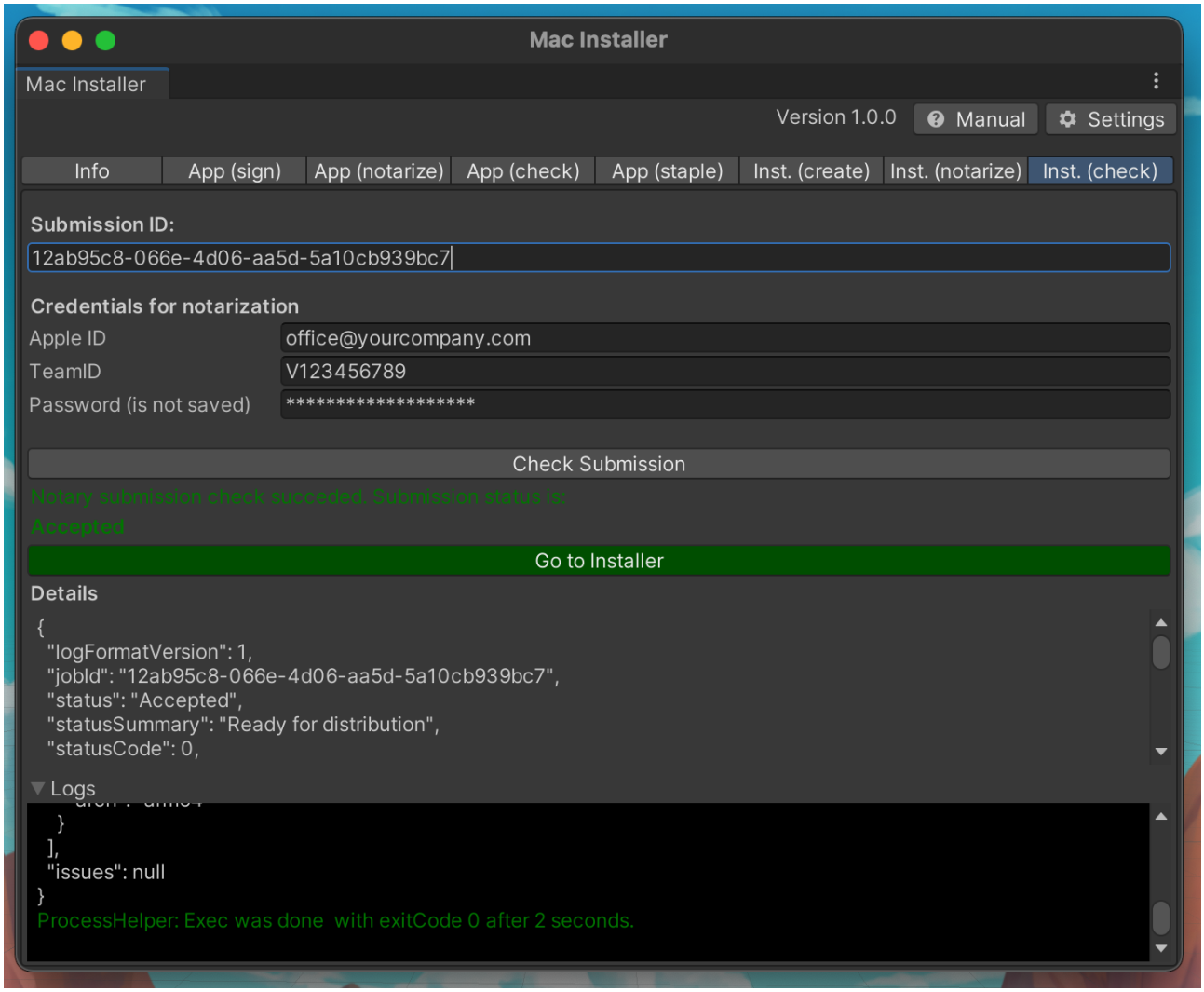
INFO: Both the APP and the INSTALLER need to be notarized separately.
The reason is that the installer needs to be trusted while installing the app and the app needs to be trusted when it is started (thus two notarizations are needed).



7) Installer Check (Notarization)

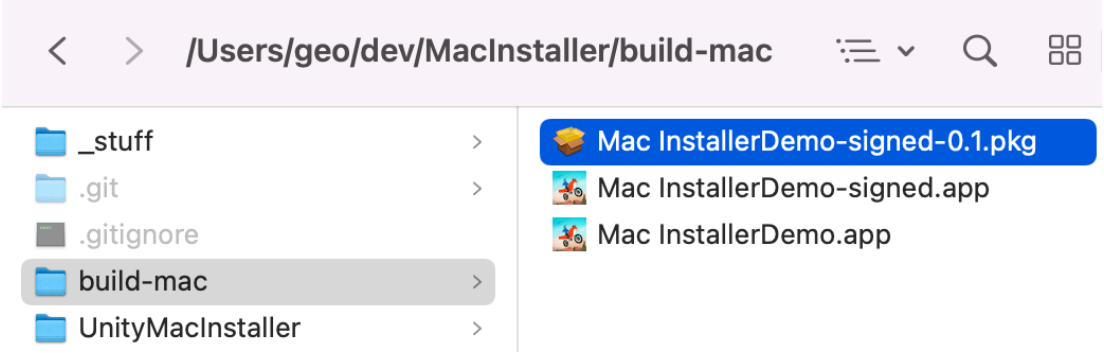
Now the only thing left to do is to wait for the results of the installer package notarization.

Time for another coffee.

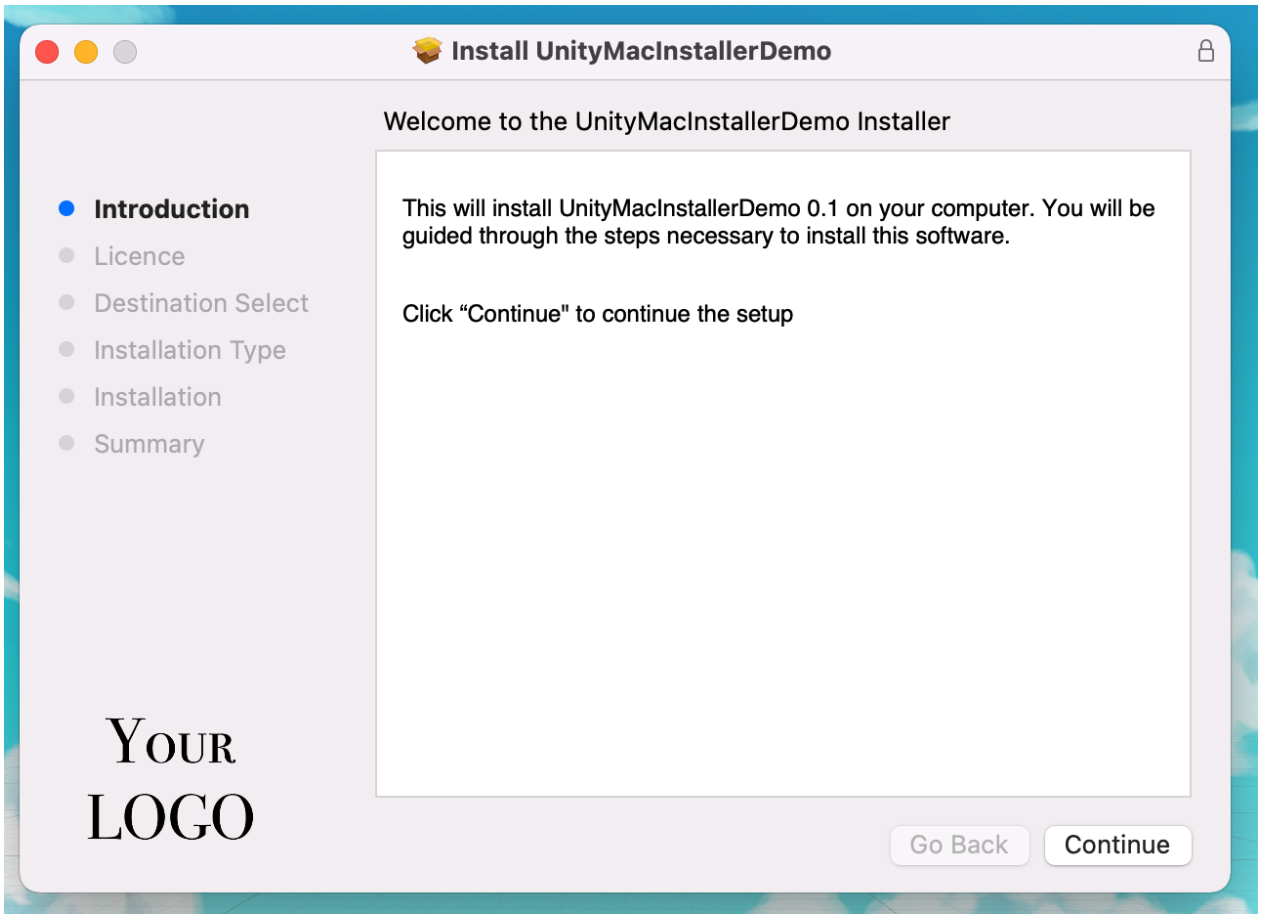


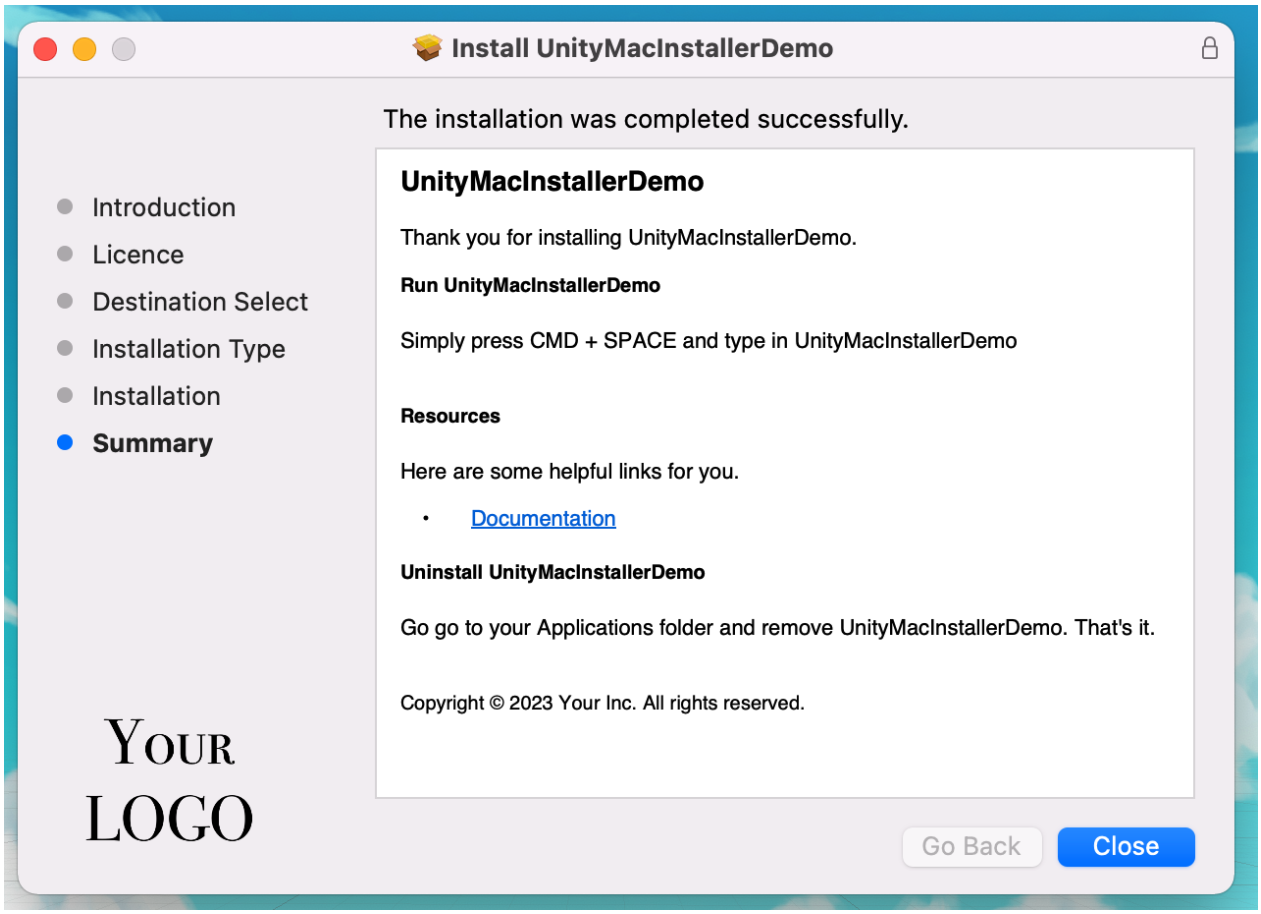
Congratulations! It is done.

Press the „Go to Installer“ to open the folder with the final installer in it.



This is how the installer should look like once you start it:

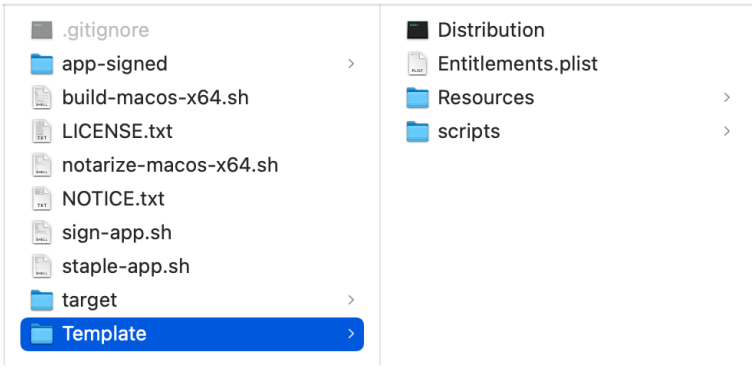




Modifying Templates

Most likely you will want to define your own license text, logo and welcome message.

The MacInstaller/MacInstallerBuild~/Template directory contains template files for the app entitlements and the installer content.



Distribution

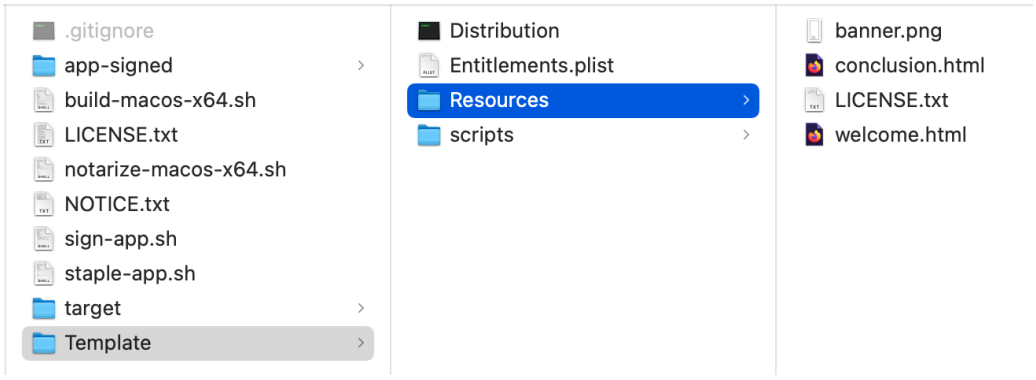
This is an xml file which contains the configuration for the installer content. The language used in the Distribution <script> tags is JavaScript.
For more details about the distribution file please refer to the Apple Distribution Manual.

Entitlements

The entitlements are added to the .app with the signature. It is a .plist file and defines what permissions your app will request.
For more details about the entitlements file please refer to the Apple Entitlements Manual.

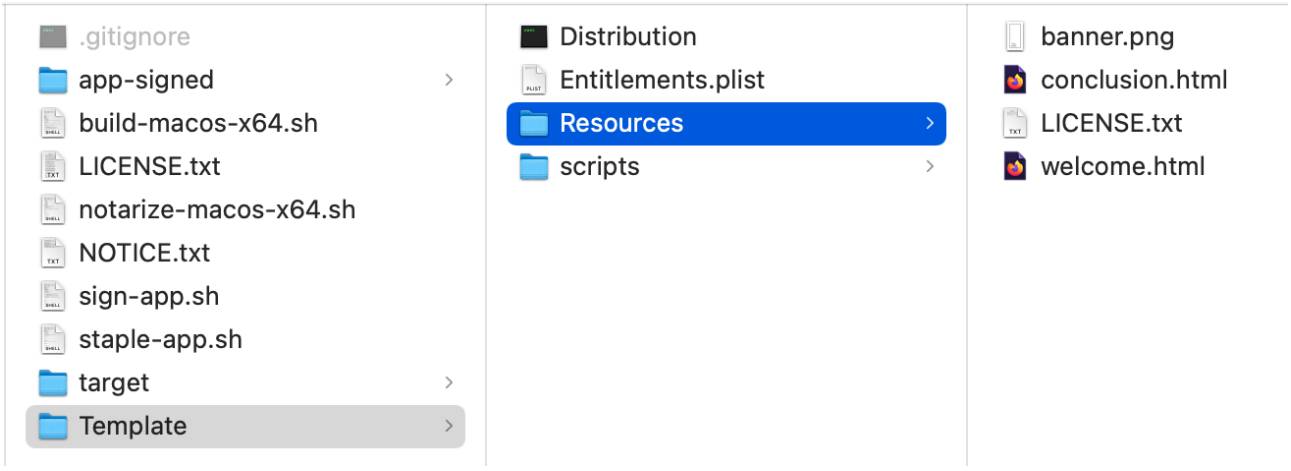
Resources

The resources folder is where the files used by the installer are stored. Each html or txt file is one page of the installer. The banner.png is the logo background (origin bottom left).



Scripts

Scripts is the folder where the post-install bash script is located. By default the script does nothing. Add your own post-build commands there if you need any.


.gitignore
app-signed
build-macos-x64.sh
LICENSE.txt
notarize-macos-x64.sh
NOTICE.txt
sign-app.sh
staple-app.sh
target
Template
Distribution
Entitlements.plist
Resources
scripts
banner.png
conclusion.html
LICENSE.txt
welcome.html

Prerequisites for Signing

To get a properly working installer you will have to do two things:
1) You need to sign it.
2) For macOS 10.14.5 or later Apple has to notarize it for you. - If you distribute through the Mac App Store then notarization is not needed as it's already part of the review process. Though in that case you would not create an installer anyways.

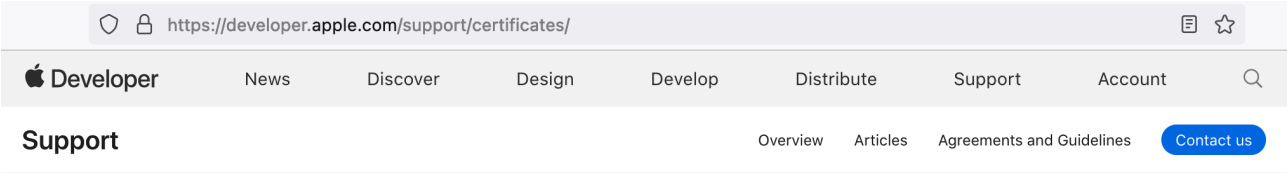
Signing the app created by Unity.

In order to sign the app created by Unity you will need a „Developer ID Application" certificate. Do NOT confuse this with the „Developer ID Installer" certificate which you will need to sign the installer.

In total you will need two certificates („Application ID" and „Installer ID").

Here is how to get them:

1) Go to https://developer.apple.com/support/certificates/ and follow the „Certificates, Identifiers & Profiles" link.



Certificates

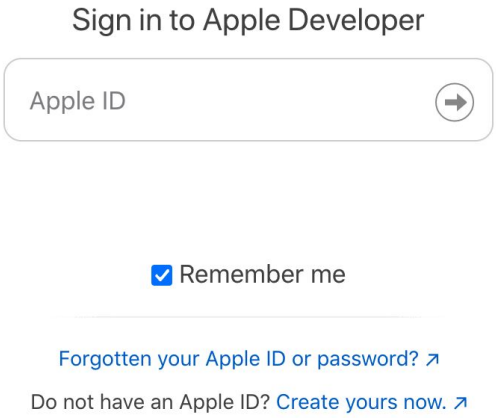
Apple Developer Program membership is required to request, download, and use signing certificates issued by Apple.

Using certificates

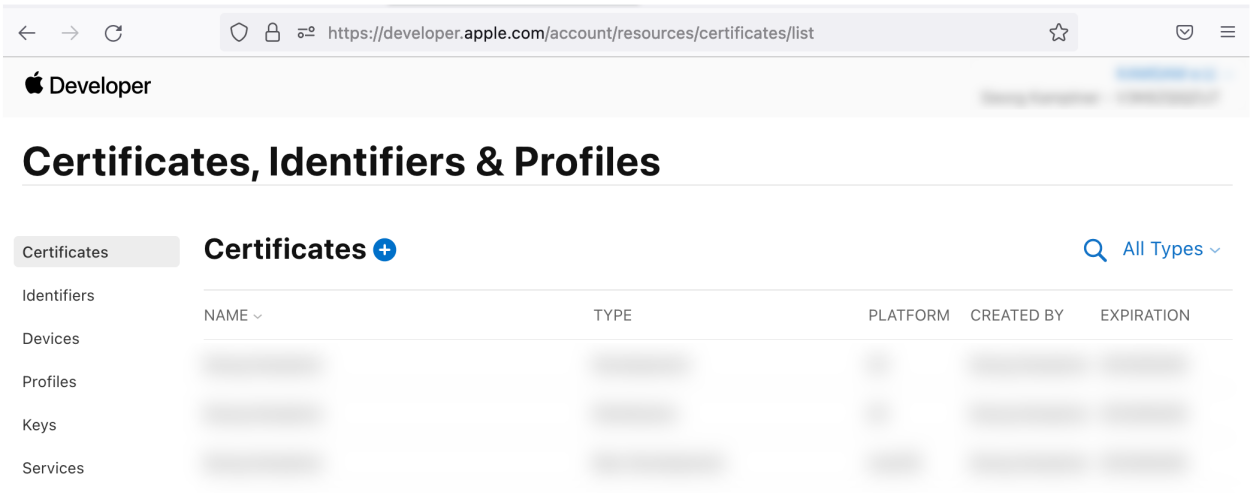
In most cases, Xcode is the preferred method to request and install digital certificates. However, to request certificates for services such as Apple Pay, the Apple Push Notification service, Apple Wallet, and Mobile Device Management, you'll need to request and download them from Certificates, Identifiers & Profiles in your developer account. Distribution certificates can be requested only by Account Holders and Admins.

For more information on how to use signing certificates, review Xcode Help.

You will have to log in to your developer account. If you do not have one you will have to create one.



Once you are logged in you will land on the „Certificates List“ page.
It looks like this:



If you do not yet have a „Developer ID Application“ and a „Developer ID Installer“ certificate then create them now.

If you are planning to relase on the Mac Store then you will have to use a „Mac Installer Disctribution“ certificate instead, but that’s not the focus of this tutorial.

First let’s create the „Developer ID Application“ certificate (we will use it to sign the .app).

Certificates, Identifiers & Profiles

‹ All Certificates

Create a New Certificate

Continue

Software

- **Apple Development**
  Sign development versions of your iOS, macOS, tvOS, and watchOS apps. For use in Xcode 11 or later.
- **Apple Distribution**
  Sign your apps for submission to the App Store or for Ad Hoc distribution. For use with Xcode 11 or later.
- **iOS App Development**
  Sign development versions of your iOS app.
- **iOS Distribution (App Store and Ad Hoc)**
  Sign your iOS app for submission to the App Store or for Ad Hoc distribution.
- **Mac Development**
  Sign development versions of your Mac app.
- **Mac App Distribution**
  This certificate is used to code sign your app and configure a Distribution Provisioning Profile for submission to the Mac App Store.
- **Mac Installer Distribution**
  This certificate is used to sign your app's Installer Package for submission to the Mac App Store.
- **Developer ID Installer**
  This certificate is used to sign your app's Installer Package for distribution outside of the Mac App Store.
- (●) **Developer ID Application**
  This certificate is used to code sign your app for distribution outside of the Mac App Store.

Download the „Developer ID Application“ certificate.

Certificates, Identifiers & Profiles

‹ All Certificates

Download Your Certificate

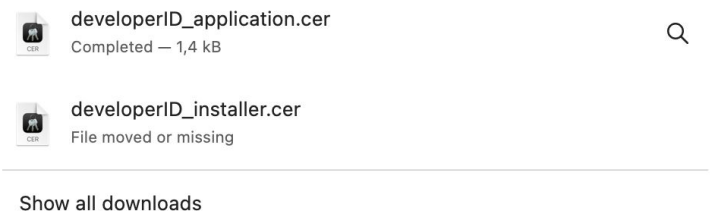
Download

Software Distribution Reminder
If you're generating your first Developer ID certificate, the software that you sign it with must be notarized by Apple in order to run on macOS 10.14.5 or later. Learn how to submit your software for notarization ›

You will also need a certificate for the Installer package (.pkg).

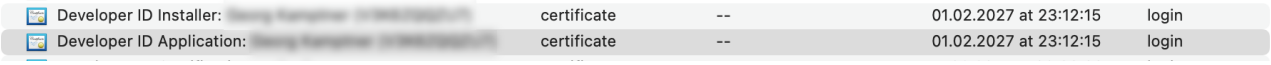
- **Mac Installer Distribution**
  This certificate is used to sign your app's Installer Package for submission to the Mac App Store.
- (●) **Developer ID Installer**
  This certificate is used to sign your app's Installer Package for distribution outside of the Mac App Store.
- **Developer ID Application**
  This certificate is used to code sign your app for distribution outside of the Mac App Store.

Save and install the two certificates.



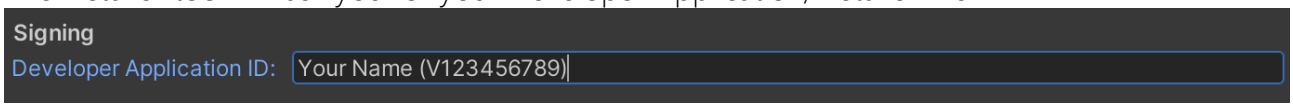
Now you should have two new certificates in your key chain.

- the „Developer ID **Application**“ certificate
- the „Developer ID **Installer**“ certificate

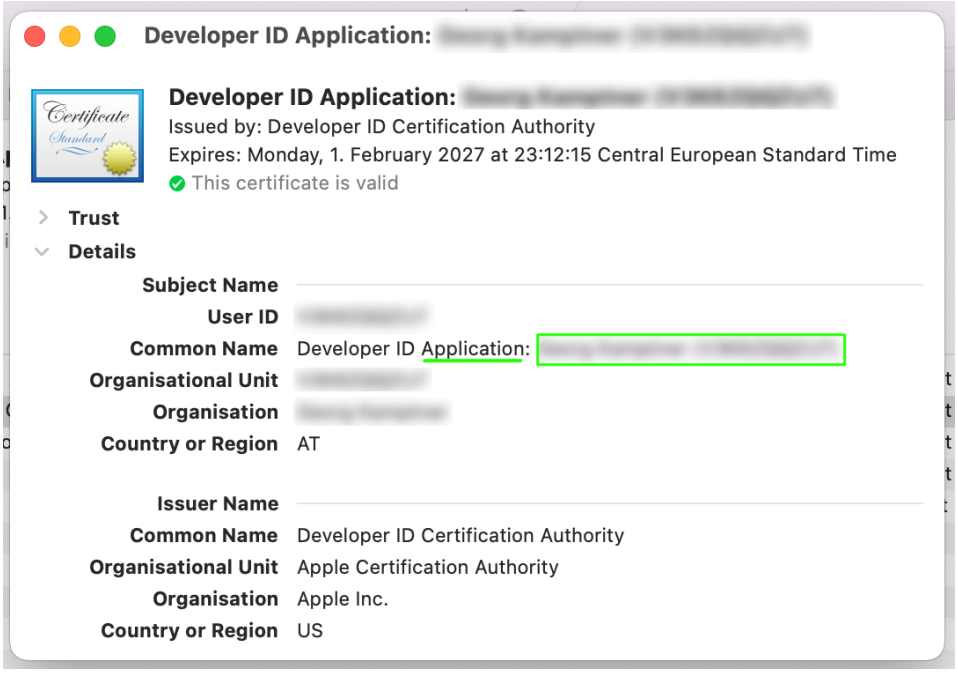


Copying the ID

The Installer tool will ask you for your Developer Application/Installer IDs



The ID has the form of „Your Name (10-char-id)“, like „Harry Pitter (V123456789)“.
If you open your certificates in the key chain then you can copy it.



You can also get them via the commandline by using „`security find-identity`“. That's also a nice way of verifying that they are installed properly.

Signing Xcode projects (not supported)

If you are using Xcode to build your app (you exported as an Xcode project) then Xcode can manage the signing process for you. Please follow this guide for signing using Xcode:

https://developer.apple.com/documentation/xcode/distributing-your-app-for-beta-testing-and-releases

NOTICE: Xcode can only create installers for distribution through the AppStore. It does NOT create standalone installers!

Prerequisites for Notarization

Notarization requires Xcode 10 or later. Building a new app for notarization requires macOS 10.13.6 or later. Stapling an app requires macOS 10.12 or later. These are hard requirements by Apple and can not be avoided.

To notarize your preexisting software you will have to prepare the following:

1. Make sure Xcode 13 or later is used.

If you have multiple versions installed then you can do that by calling „`sudo xcode-select -s /path/to/Xcode13.app`".

If you have only one Xcode version installed then you can skip this step.

2. Create an app-specific password for the notarytool

Because App Store Connect requires two-factor authentication (2FA) on all accounts, you must create an app-specific password for notarytool, as described in Using app-specific passwords.

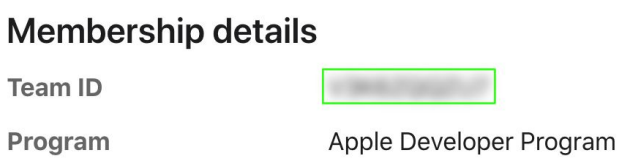
How to generate an app-specific password?

1. Sign in to appleid.apple.com.
2. In the Sign-In and Security section, select App-Specific Passwords.
3. Select Generate an app-specific password or select the Add button Blue plus sign icon., then follow the steps on your screen.
4. Enter or paste the app-specific password into the password field of the app.

Upload your software to the Apple notary service, as described in Upload your app to the notarization service.

Where do I find my Team ID?

Login to https://developer.apple.com/account/ and then go to „Membership details". There you will find your TeamID. Usually the Team ID is also listed in your certificates.



Frequently Asked Questions

I want to use this in Unity Cloud Build or my CI/CD chain.

At the moment CI/CD is not officially supported but it is possible to integrate (you would have to write some code).

One reason I have not yet added this is that Apple states that notarization can take up to 24 hours. So any automated build prozess would have to routinely check for the notarization result for up to 24 hours. Right now that's a manual step. Another big reason is ongoing support (see below).

Integration Guide (Unity):
You can write a c# script that goes through all the steps automatically. To do that please check out the „public static async" methods of the „MacInstaller" class. They are named after the steps in the UI so implemementation should be pretty straight forward (if you have all the singing infos prepared).

Integration Guide (shell scripts):
In the end all the work is done by the shell scripts contained in "Assets\Kamgam\MacInstaller\MacInstallerBuild". In theory you don't even need Unity to do it all (just call the shell scripts with the correct paramters from your CI/CD build chain). In this scenario you would have to make your toolchain wait for the results of the scripts and in case of the notarization you would have to make your toolchain check the notarization results repeatedly until they have completed or an error occurred or the 24 hours have passed (which would also constitute an error).

At the moment I have no plans to implement cloud build support myself.

The main reason is not the initial workload for adding this but support. That woul be doable. However, I pride myself in giving excellent detailed support, yet cloud builds are inherently tricky and giving support on these external systems is hard and very time consuming (time that I am not getting paid for). Sorry that I have to say it this hard but I just can't afford to spend time debugging peoples cloud builds (that often fail due to errors outside the scope of my assets). - There, I have said it. It's a money issue. If you are determined to get cloud builds to work on your project then we could make a custom deal (just contact me).