

Package Builder User Guide



Software by **Araelium**

Table of Contents

Overview	4
What is an Installer Package?	5
Component Packages Create "Choices"	6
Installer Settings	8
Code Signing & Notarization	10
Code Signing	10
Notarization	10
Installer Scripting with Installer JS	11
Enabling Installer Scripting	11
Custom Functions	12
Install Check	12
Destinations	13
Install Choices	14
How to Create an Installer With No Choices	14
How to Create an Installer With Multiple Choices	14
User-Customized Installation	16
Descriptions	16
Component Package Settings	17
Files to Be Installed	18
Install Location	18
Install Locations are "Destination" Relative	19
File References	20

Window Content	21
Background Image	21
Introduction, Read Me, License, Summary	22
Localization	22
Pre/Post-Install Scripts	23
Extra Resources	23
Arguments	23
Environment Variables	24
Security	24
DMG Canvas Integration	26
Building Packages with DMG Canvas	26

Overview

Package Builder is used to create macOS installer packages (".pkg" files) which can be used to install applications, fonts, system extensions, device components, and any other type of file.

Using **Package Builder**, you specify which files to install, where to install them, and what content should be displayed to the user before and after installing, to introduce and guide them through the process.

Installer packages can be built with the **Package Builder** application, or with its included [pkgbuilder](#) command line tool which can be used as part of a larger automated workflow.

Package Builder can code sign and notarize the installer package to ensure it can be launched without being restricted by macOS's Gatekeeper security features.

What is an Installer Package?

A macOS installer package contains files that can be installed on a user's Mac, along with instructions for where they should be installed. When users open your installer package, it opens in the built-in Installer application which guides them through the installation process, ensuring that their computer meets the installation requirements defined in the package before placing the payload on the user's file system, among other tasks.

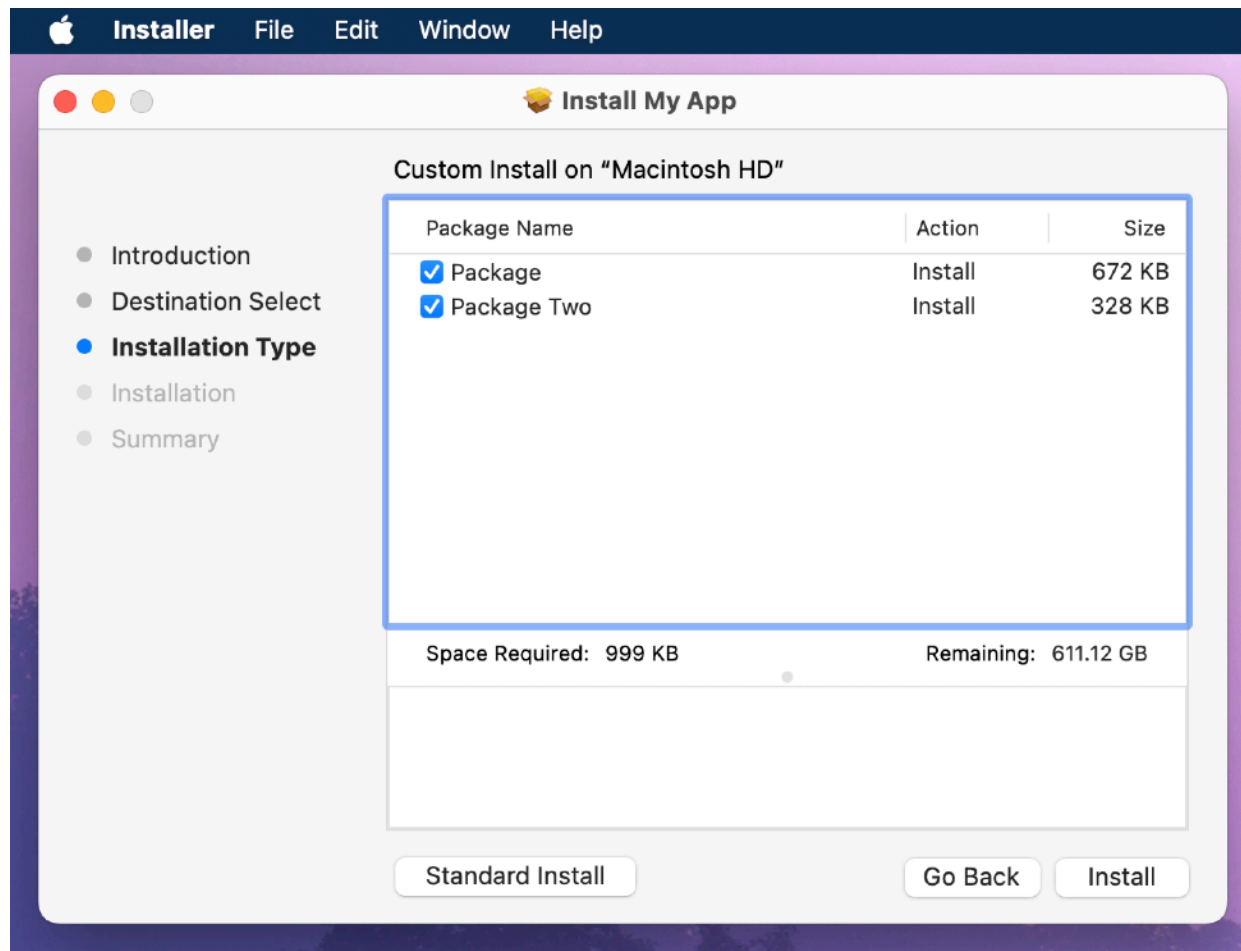
An installer package is the best choice if your product contains multiple components, must be copied to specific locations, or if you need to run custom code during installation.



Component Packages Create "Choices"

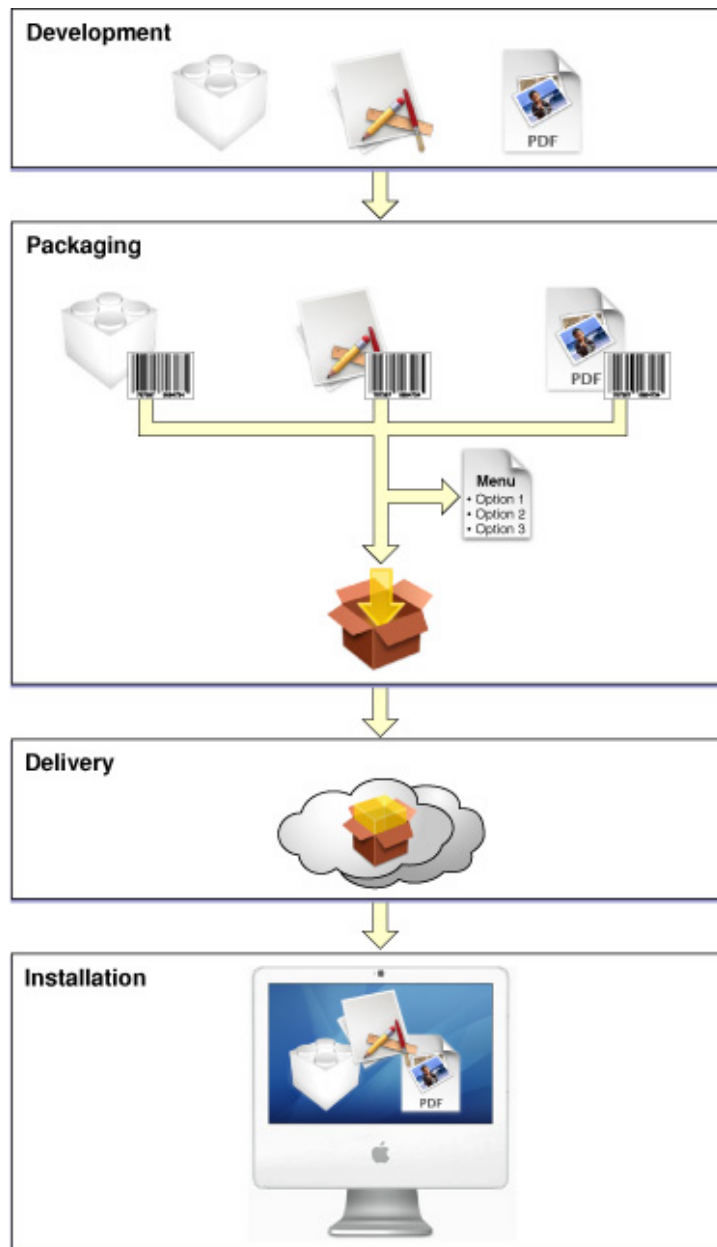
The installer package file (.pkg) that a user will open is referred to by Apple as the "product" or "distribution" package, and is internally composed of one or more "component" packages.

When there is more than one component package in an installer, each component is displayed as an independent "choice" which can be separately installed.



In this screenshot, "Package" and "Package Two" are each a separate component packages. Package Builder supports creating installer packages which contain multiple choices like this, or a single installer with no choices, where all files in the installer are always installed.

Example



As an example, an installation may include a system plugin, an application, and a pdf file for documentation. Each of these could be contained within a separate *component package*, inside of your installer package. Each component then becomes a choice (as displayed above) in the installer process. A user could then choose to install only the documentation, or could choose to install the application and plugin without installing the documentation.

Installer Settings

Each installer itself has a Name, Identifier, and Version, and *these are distinct* from the same settings on each individual package within the installer.

Installer Name

The installer Name is used as the suggested file name (ex: Name.pkg) and may be displayed within the installer window's titlebar during the installation process.

Identifier

The identifier is standard macOS reverse-domain formatted bundle identifier which you choose, that uniquely identifies your installer.

For example:

`com.mycompany.myproduct.installer`

The macOS Installer database uses the identifier and version number combination to determine if a package has been installed before, and if a newer version is already installed. This can be used in advanced scenarios.

Version

The version number of your installer package may typically correspond to the version number of the product it is installing. Though it is technically optional and will default to version 1, every time you release a new build, you should increase this number.

Gatekeeper

See [Code Signing & Notarization](#).

Requirements

In addition to the min/max OS version, processor architecture, and minimum RAM requirements, you can also use a custom JavaScript function which can determine and control whether the installation can proceed.

See [Installer Scripting](#).

Destination

macOS allows install into one of three domains:

1. User Folder
2. Startup Disk
3. Any Disk

Each file that is installed, will be placed at a location (specified by you) that is always *relative* to the user-chosen "Destination" during the installer process.

Choose whichever destinations are appropriate for your installer. If the allowed destinations must be determined at the time the installer is opened, you can customize this with a JavaScript function. See [Installer Scripting](#).

Code Signing & Notarization

Code Signing

Code signing your installer package ensures to the recipient that the contents of the installer have not been modified.

To code sign your installer, you must:

4. Have a **Developer ID: Installer** certificate from Apple installed on your Mac. You obtain this certificate by signing up and creating it in a developer account on Apple's website.
5. All executables that will be installed by the installer (applications, command line tools, and even shell scripts etc) must be code signed with a corresponding **Developer ID: Application** certificate.
You must code sign these contents before the package is built.

Notarization

Notarizing your installer package ensures to the recipient that the contents of the installer have been verified by Apple to not contain malicious code. Notarization is optional.

The notarization process involves uploading the fully built and code signed installer package to Apple. After an analysis of your files is complete (which usually takes only a few minutes), the notarization succeeds, and "notarization ticket" is embedded within the installer package which allows any recipient to know it has been notarized by Apple.

To notarize your installer, you must:

1. Have a **Developer ID: Application** certificate from Apple installed on your Mac. You obtain this certificate by signing up and creating it in a developer account on Apple's website.
2. Have the installer package code signed by Package Builder, as detailed previously.

Installer Scripting with Installer JS

The **macOS application "Installer"** is what displays the interface of your installer when opened, validates and determines whether your installation can occur, runs the installation, and displays the result.

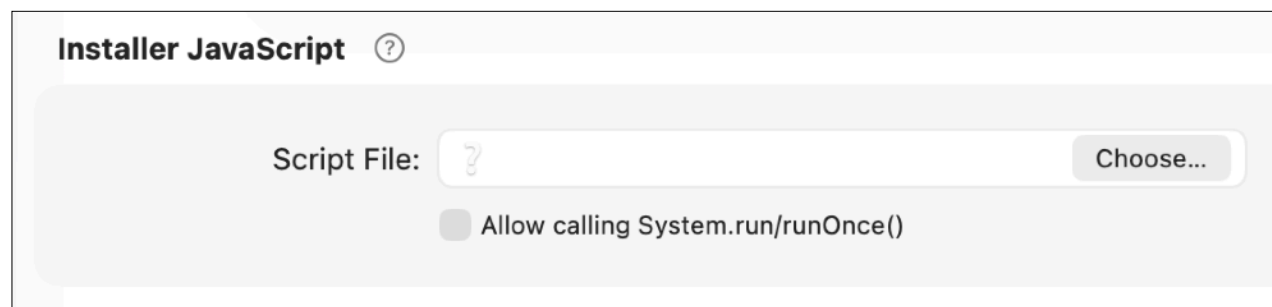
Some of the processes within Installer can be customized dynamically while the installer is open, by using **JavaScript**. This includes:

- Determining install requirements
- Enabling allowed install destinations

See Apple's **Installer JS** documentation for information about types and services Apple offers:
https://developer.apple.com/documentation/installer_js

Enabling Installer Scripting

Select the **Installer → Scripting** item in the sidebar, and create a Script File.

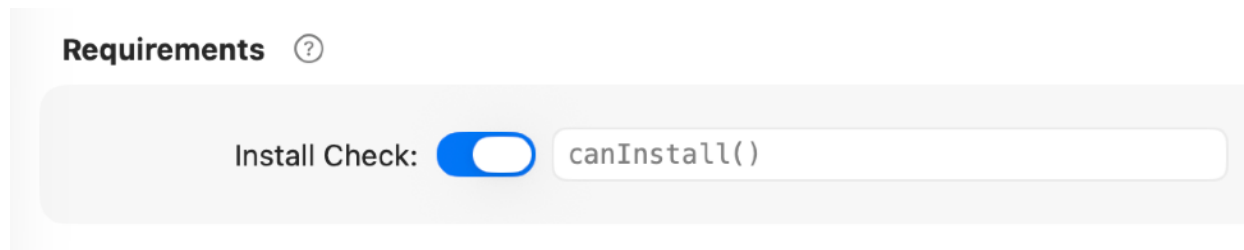


In this JavaScript file, you can add any support functions you need, as well as the functions identified for customization.

Note: Allow calling System.run/runOnce(): Because System.run allows running external scripts which may have been created or modified by others, turning on this option will cause the Installer to show a prompt to the end user asking permission to run a program.

Custom Functions

For each capability you want to customize with JavaScript, turn the option on, and specify the function name to be called. If you do not explicitly enter a function name, a default name will be used.



The customization function must return an appropriate value, and if necessary, set other specific variables to pass other information back to the Installer application.

Installer JS places a number of variables into the global object variable `my`, as seen below.

Install Check

Return: `Boolean`

If the function returns `true`, the installation can proceed (as long as the other requirements set in the Installer Settings are met).

If it returns `false`, it must also set `my.result.type` and `my.result.message` to indicate the severity of the problem and provide a message to display. If `my.result.type` is `"Warning"`, the installation may be allowed to proceed after informing the user.

```
function canInstall() {  
  const num = Math.floor(Math.random() * 2);  
  if (num == 0) {  
    my.result.type = "Fatal";  
    my.result.message = "The odds are not in your favor.";  
    return false;  
  } else {  
    return true;  
  }  
}
```

Example code for an Install Check function.

Destinations

Return: Boolean

The function returns **true** or **false** to allow or disallow the installation into the destination.

Install Choices

See [*What is an Installer Package?*](#) for an explanation of component packages and choices.

How to Create an Installer With No Choices

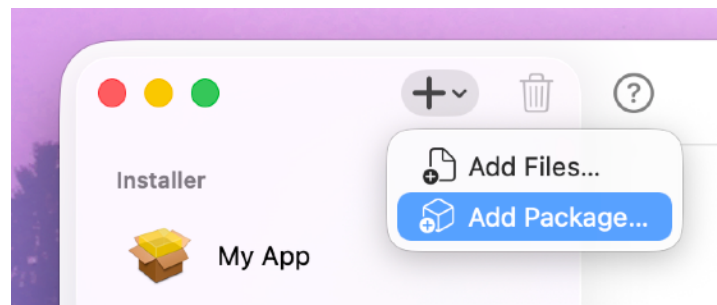
If an installer has only a single component package, there will no user-visible choices. All files added to the installer will be installed.

For installers with multiple component packages, you can:

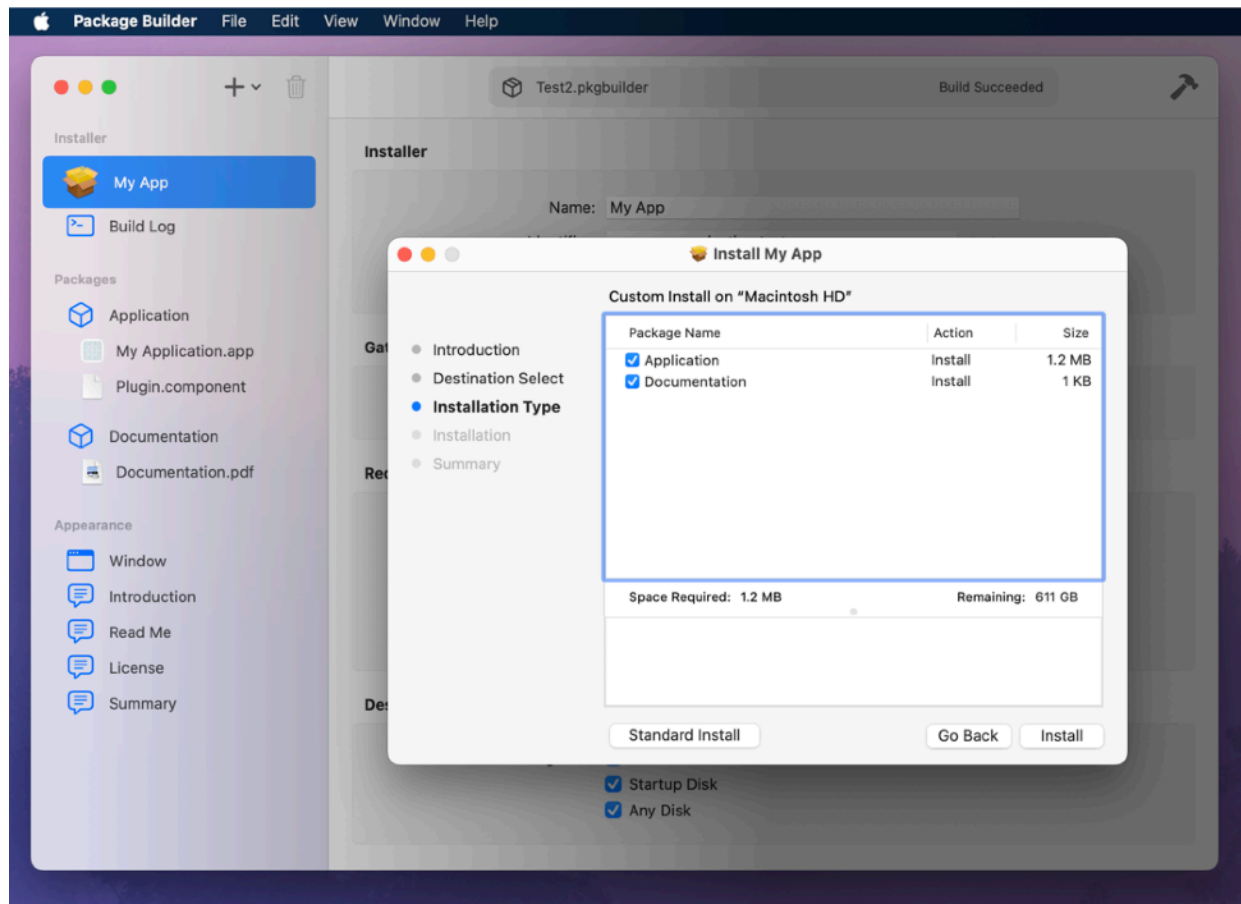
1. Set *Customized Installation* to *Not Allowed*
2. All packages will be enabled and installed.

How to Create an Installer With Multiple Choices

Adding two or more packages will present each package as an optional choice which can be installed by the user.



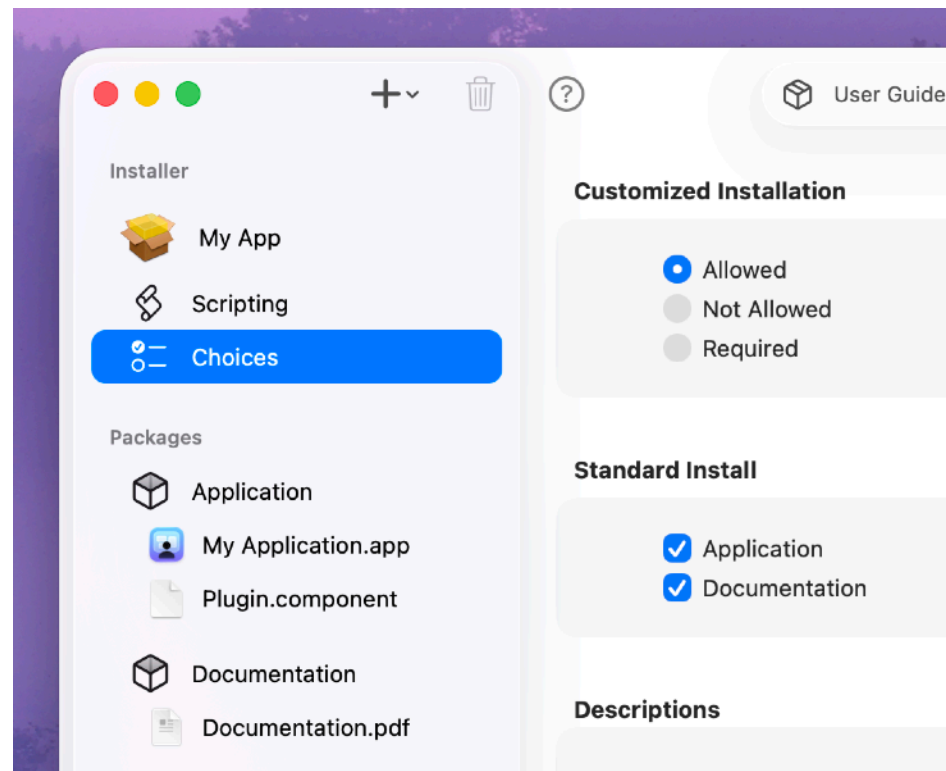
Example – Multiple Choices



In this screenshot, note that the two component packages in Package Builder are presented to the user in the installation process as two separate install choices.

User-Customized Installation

When there are two or more component packages, you can allow the user to perform a customized installation, and select which packages are part of the standard installation.



Descriptions

When the user does a customized installation, a description can be seen when they click on each installation choice. These descriptions are not shown anywhere else.

This description should be brief. **Do not** put important installation information or license information in this description. That information should be in the other window content sections such as *Read Me* and *License*.

Component Package Settings

Clicking on the package in the sidebar will display settings for that specific package.

Each component package has its own Name, Identifier, and Version. If, for example, the package installs an application, these should likely reflect the name, identifier, and version of the application being installed.

Package Name

The package Name is displayed within the installation process. When multiple packages are available, each package may be a "choice" shown to the user, using this name.

Identifier

The identifier is standard macOS reverse-domain formatted bundle identifier which you choose, that uniquely identifies your installation package.

For example:

`com.mycompany.myproduct`

The macOS Installer database uses the identifier and version number combination to determine if a package has been installed before, and if a newer version is already installed. This can be used in advanced scenarios.

Version

The version number of your package typically corresponds to the version number of the product it is installing. Though it is technically optional and will default to version 1, every time you release a new build, you should set this number.

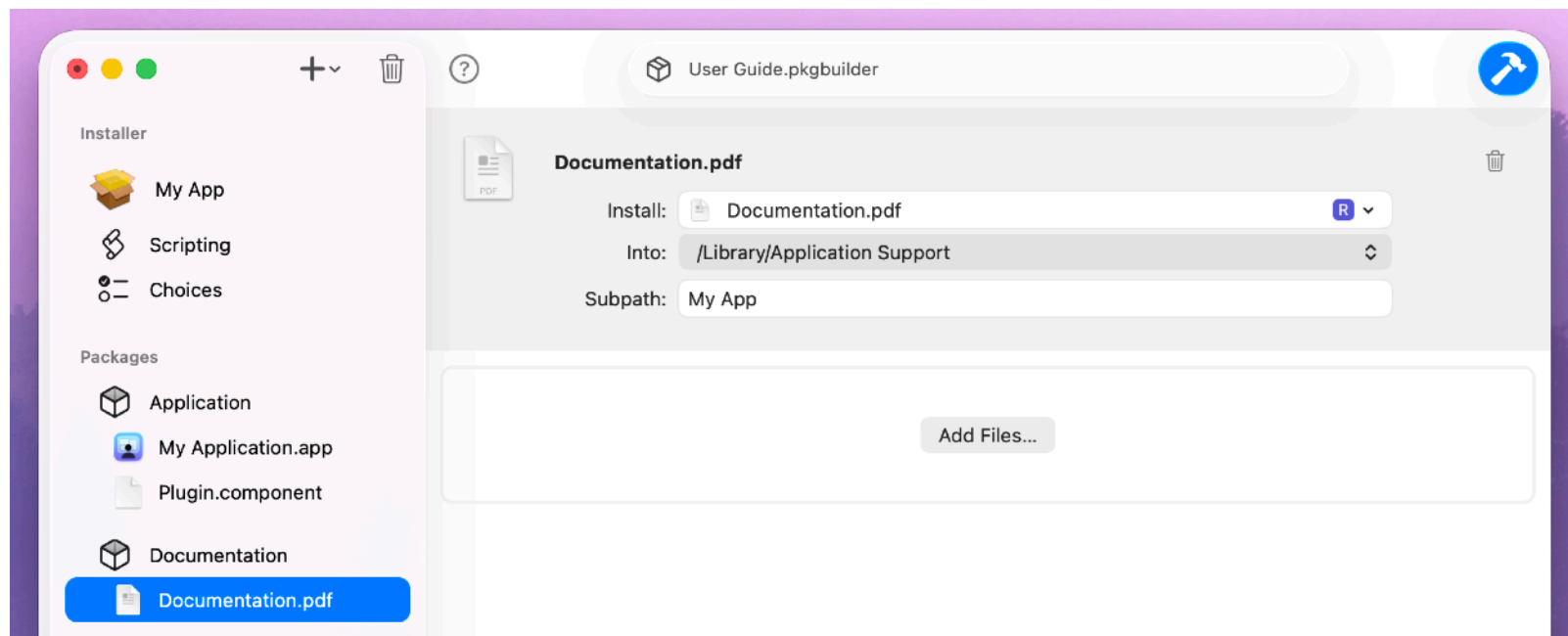
Quit Applications

If the files being installed may affect the behavior of running applications, you can supply a list of those applications which will be required to be quit before the installation can proceed. Enter the application bundle's identifier into this list.

Files to Be Installed

To add files to be installed, either:

- Drag and drop files into the Package in the sidebar
- Or, Select "Add Files..." from the Add menu in the sidebar

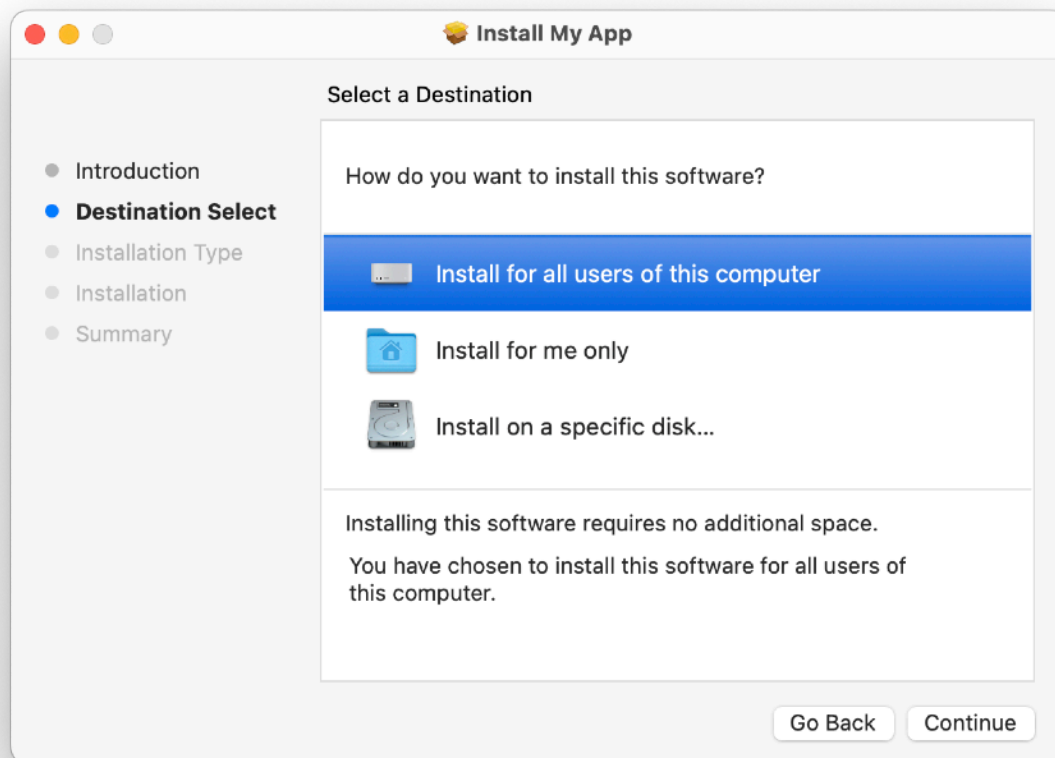


Install Location

For each file, specify the location the file will be installed at. In the example above, the [Documentation.pdf](#) file will be installed into:

[.../Library/Application Support/My App/Documentation.pdf](#)

... relative to the user's chosen Destination during the installation process. See the next section for an explanation.



Install Locations are "Destination" Relative

Depending on the *Destination* choices you allow in the Installer Settings, when the user installs your package they can choose to install it into the Startup Disk, Any Disk, or their User Folder.

The install locations chosen for each file individually, are always installed *relative* to the chosen "Destination" during the installer process.

For example, if the user chooses to install the above package into their user folder ("Install for me only"), then [Documentation.pdf](#) file will actually be installed into

[/Users/the-user's-name/Library/Application Support/My App/Documentation.pdf](#)

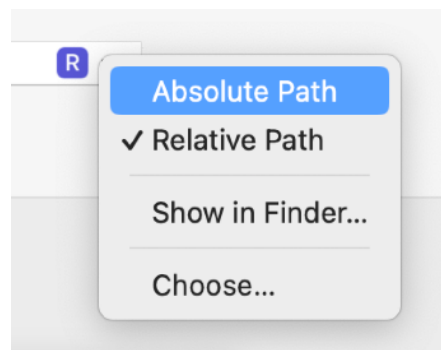
File References

All files used by the Package Builder document (such as for installable files, the window background image, licenses, scripts, etc) are referenced either by:

- **Absolute file path**, or
- **Relative file path**, relative to the saved Package Builder document

You can change which reference style is used by:

- Editing the text path to be either absolute or relative
- Or, by selecting the choice from the menu at the right side of the file reference control.



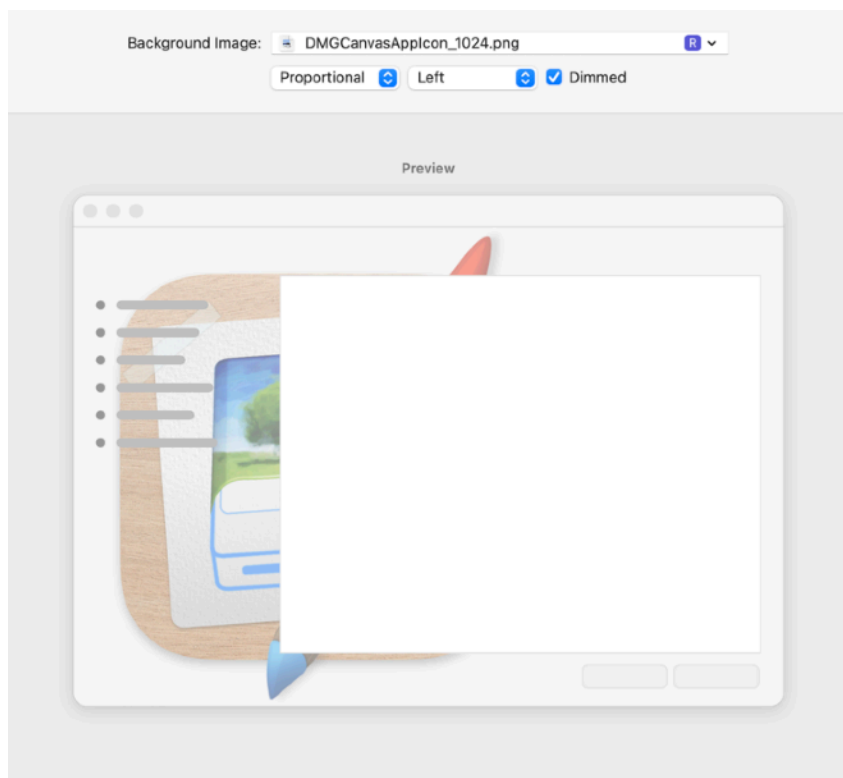
Use absolute paths when: the files being referenced will not change locations, but the Package Builder document may change locations.

Use relative paths when: the files being referenced and the Package Builder document may change locations. This is typically the preferred option when sharing files between devices.

Window Content

Background Image

The Installer application can display an image in the background of the window behind the installer's interface. It's most commonly used to place a "dimmed" image of the application icon fitted proportionally to the left edge of the window.



The macOS Installer application has used the same size window for many years now, and there appears to be no reason to suspect it will change anytime soon. An image of size 620 x 418 (or @2x) will fill the entire window while maintaining its aspect ratio.

Introduction, Read Me, License, Summary

As part of the installation process, the Installer application can have distinct steps which display text, rtf, or html content. These steps are called (by the Installer application itself):

- Introduction
- Read Me
- License
- Summary

Introduction – This is the first screen shown when the installer is opened. Generally it should be a short friendly welcome message.

Read Me – This can contain additional and lengthy information about the installation or product being installed. The Installer application also shows *Print* and *Save* buttons so this text can be saved for later by the user.

License – In this view, your End User License Agreement can be displayed, and must be explicitly agreed to by the user (by clicking Accept) before the installation can proceed.

Summary – This view is displayed after the installation takes place.

It is not required that packages provide all of these sections. Any section without content is not displayed as part of the install process.

Localization

Each of these sections is composed of one or localized files which contains what is displayed. All files **must** be the same file type, and can be plain text (.txt), rich text (.rtf), or HTML (.html).

Pre/Post-Install Scripts

Installer packages may run custom command line scripts before and after installation. Script files can be written in any language as long as they are marked executable and have a "shebang" at the beginning of the file. Any exit value other than zero indicates a failure.

If your installer package has no installation choices (the default), then you can set the scripts for Before and After installation in the Installer Settings view by selecting the installer in the sidebar.

If your installer package has multiple component package choices, then you can set install scripts for *each* component separately, by selecting the component package in the sidebar.

Extra Resources

By adding files and folders to the **Extra Resources** list, these files will be bundled inside the installer. Scripts can then refer to these files by a relative file path as they will be in the current working directory (the same directory as the running script file).

Arguments

macOS Installer provides some arguments when executing the scripts. The availability of arguments may change.

Arg 0	The path to the running script.
Arg 1	The path to the installer package.
Arg 2	The path to the root folder where the component package's file are being installed into. (Ex: If the user chooses to install into their user folder, this would be / Users/the-user-name)
Arg 3	The path to the target volume the user has chosen to install into. (Ex: /)

Environment Variables

macOS Installer provides some environment variables when executing the scripts. The availability of variables may change.

COMMAND_LINE_INSTALL	If present, the installation is being installed through the command line instead of the Installer application.
HOME	The path to the current user's home folder.
INSTALLER_PKG_SESSION_ID	The component package's identifier.
USER	The user name of the current user.
DSTROOT	Same as Arg #2
PACKAGE_PATH	Same as Arg #1
SCRIPT_NAME	The name of the script file (Ex: postinstall, preinstall)
INSTALLER_SECURE_TEMP	A temporary folder the script can use.
INSTALLER_PAYLOAD_DIR	The root folder of the component package's payload. This contains the files to be installed, in their root-relative hierarchal structure.
PWD	The current working directory, which contains the script files, and the "extra" resource files you add.
APPLE_PKGKIT_ESCALATING_ROOT	If 1, the script is being executed with root.

Security

Note that because the script may be executed in a privileged environment, the script should avoid loading untrusted settings. For example, loading user shell configuration can run *user space config*, with *root privileges*, which is susceptible to abuse. To guarantee avoidance, use one the following shebangs.

- SH: `#!/bin/sh` (sh in macOS won't load any external settings without being explicitly told)
- ZSH: `#!/bin/zsh --no-rcs`
- Bash: `#!/bin/bash --noprofile --norc`

You should also be mindful of what other user code might be executed when loading Perl, Ruby, Python, etc.

Additionally, always specify the full path to binaries in scripts when possible, as PATH may not always be trusted and could point to a malicious binary.

ex. `/bin/mkdir` instead of `mkdir`

From Mykola Grymalyuk: <https://khronokernel.com/macOS/2024/06/03/CVE-2024-27822.html>

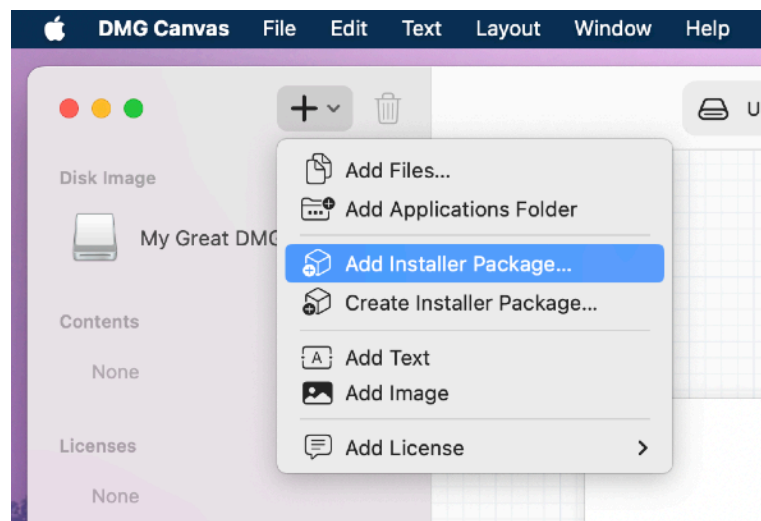
DMG Canvas Integration

It is common to ship an installer package (.pkg file) in a macOS disk image (.dmg). DMG Canvas (which is an application for visually designing, laying out, and building a disk images similar to Package Builder) has integrated support for using Package Builder.

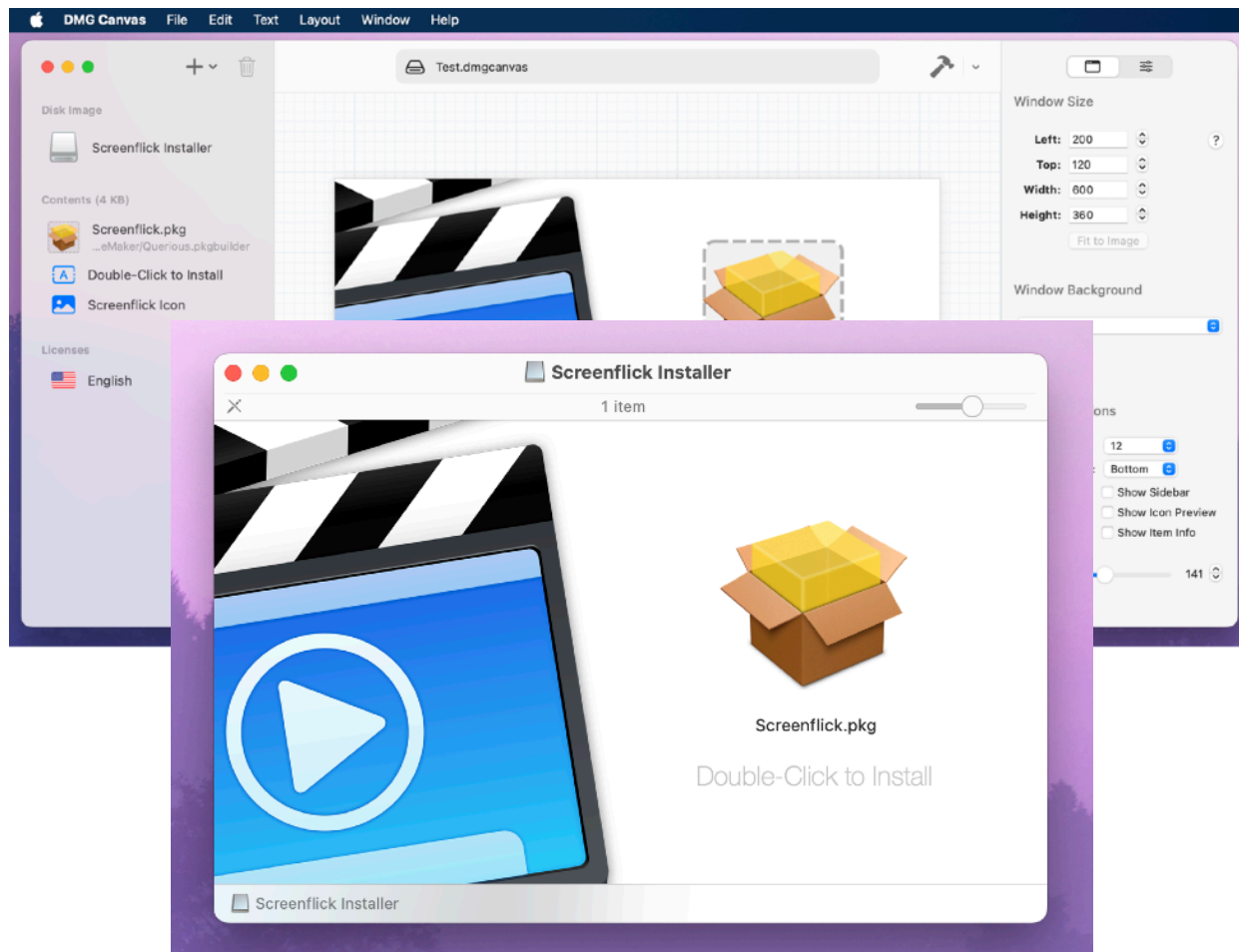
Building Packages with DMG Canvas

To have DMG Canvas trigger the building of a Package Builder document, and use the resulting .pkg file in the disk image:

1. Select "Add Installer Package..." or "Create Installer Package..." from the Add menu in the main sidebar in a DMG Canvas document.



2. The .pkg file which will be built by Package Builder will appear in the DMG Canvas file layout view with a dashed outline.



3. Building the DMG Canvas document will automatically trigger building the installer package.