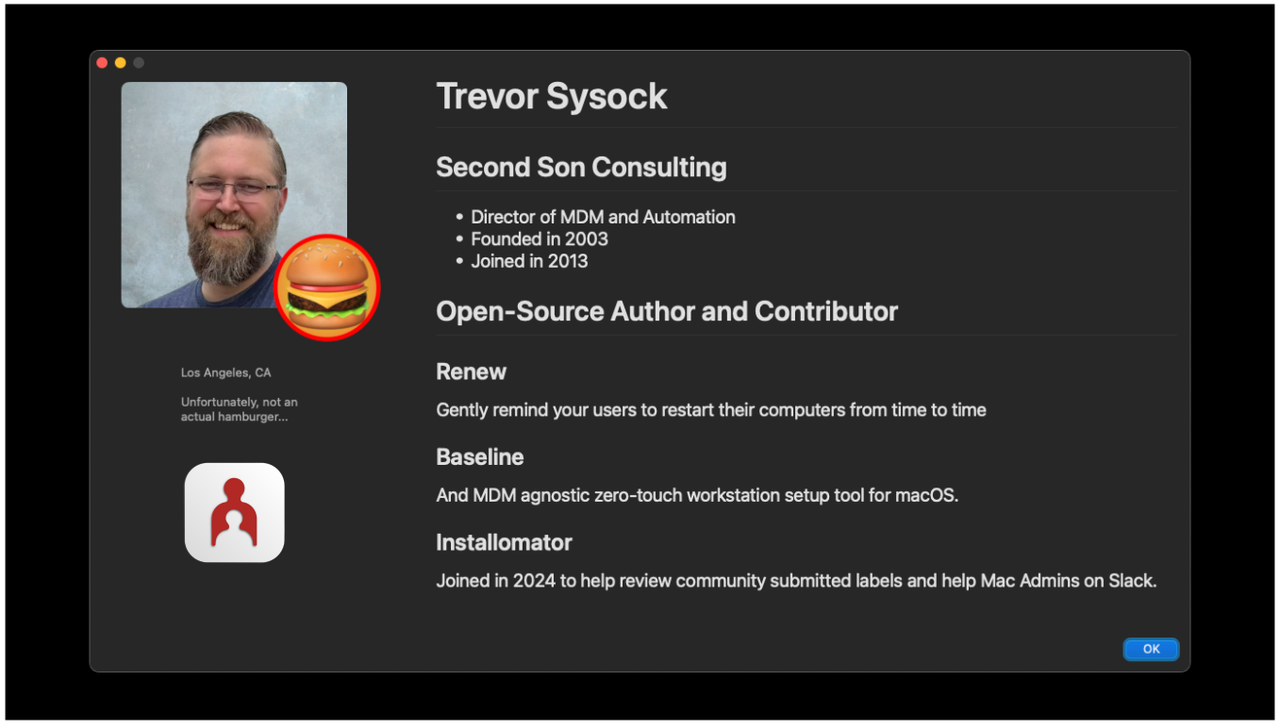
You Got Your Installomator in my swiftDialog!

Trevor Sysock - aka BigMacAdmin - 2024


Trevor Sysock
Second Son Consulting
• Director of MDM and Automation
• Founded in 2003
• Joined in 2013
Open-Source Author and Contributor
Renew
Gently remind your users to restart their computers from time to time
Baseline
And MDM agnostic zero-touch workstation setup tool for macOS.
Installomator
Joined in 2024 to help review community submitted labels and help Mac Admins on Slack.
Los Angeles, CA
Unfortunately, not an actual hamburger...
OK

1. Intro
2. What is Installomator
3. What is swiftDialog
4. Some ways to smash them together

Installomator and swiftDialog

What are they?

No dependencies outside of themselves

As a consultant I will collaborate with internal IT teams at various companies, and have used both of these tools in a large number of mdms

mosyle, kandji, intune, jamf, addigy, you name it.

macOS fundamentals

Installomator and swiftDialog

What are they?

- MDM Agnostic
- No Dependencies

Installomator and swiftDialog

What are they?

- **Building blocks**
- **Not finished products**

Building blocks to create larger scripts and user experiences

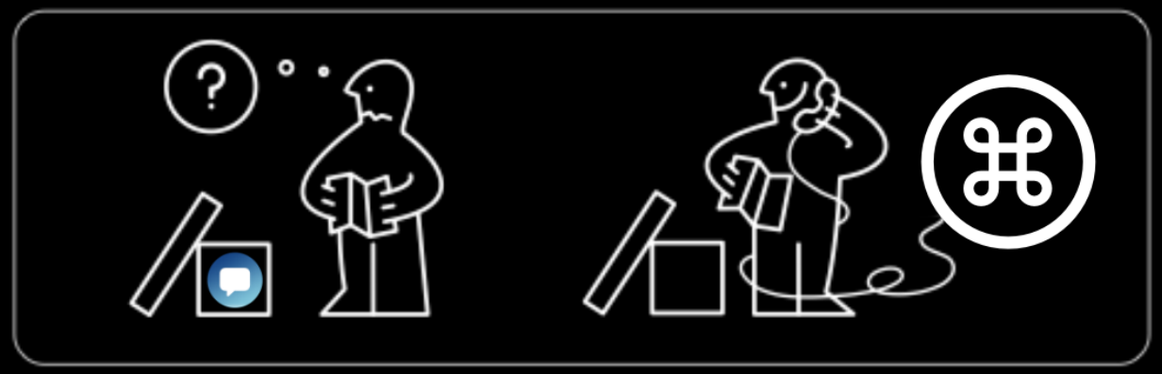
It may end up being one small part of a larger strategy, or you may want to build full deployment workflows around it.

Because of the "agnostic" nature of these tools,
discussing specifics about "the best way to use Installomator"
become very detailed and tailored to your organization and management structure.

Installomator and swiftDialog

What are they?

Get support on the Mac Admins Slack

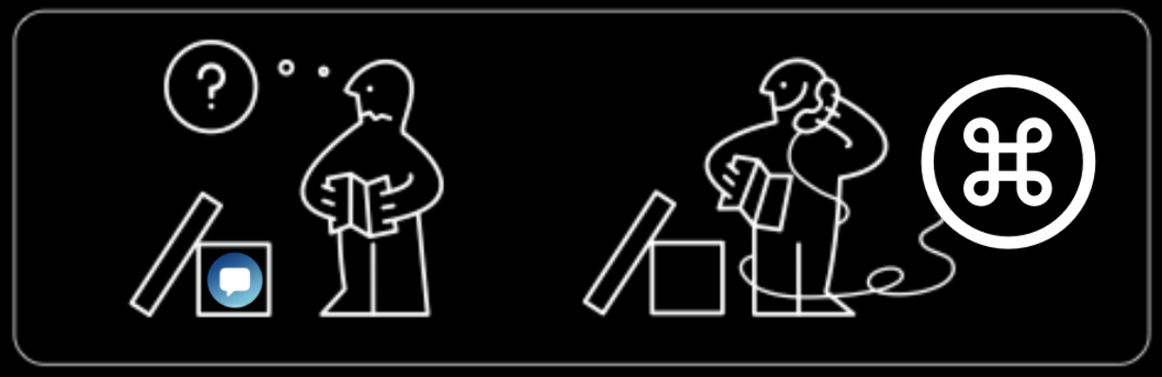


This is why the Mac Admins Slack exists, there are lots of folks including myself there to help talk through these things.

Installomator and swiftDialog

What are they?

Thousands of admins in #swiftdialog #installomator



Installomator and swiftDialog

Why Choose Installomator?

- Easy to use
- Community maintained
- Deploy the latest version *fast*

Easy to use - hopefully after today you'll feel confident to start testing

Community maintained - Lots of admins using in lots of scenarios and willing to help

Fast action - No need to repackage or wait for MDM vendors to approve new versions

Installomator and swiftDialog

Why Choose Installomator?

- Easy to use
- Community maintained
- Deploy the latest version *fast*

Installomator and swiftDialog

Why Choose Installomator?

- No infrastructure to maintain
- Not beholden to a vendor

No servers no s3 buckets

Not relying on vendors to keep things the same or add applications to their library.

Potential MDM migrations become easier

Installomator and swiftDialog

Why Choose Installomator?

- No infrastructure to maintain
- Not beholden to a vendor

Installomator and swiftDialog

Why Choose Installomator?

- **Install or update software with a single line of code:**

```
/usr/local/Installomator/Installomator.sh supportapp
```

Installomator and swiftDialog

Why Choose swiftDialog?

- **Branded and consistent messaging**
- **Prompts and notifications as beautiful as macOS**
- **Full control over your workflows, not beholden to your MDM**

That last one is big for us, at Second Son. We obviously have our preferred tools, but we're more than just an MSP we are also consultants.

I want to know that something I'm building for one environment could be easily translated into another if needed, whether Jamf, Mosyle, Addigy, whatever.

Installomator and swiftDialog

Why Choose swiftDialog?

- Transparency
  - Keep your users informed

In general, I subscribe to Apple's philosophy that the end user should know what we're doing on their devices even if they can't always opt out.

Installomator and swiftDialog

Why Choose swiftDialog?

- Compliance
  - Encourage or enforce your users to approve actions
  - Allow users to choose when these occur

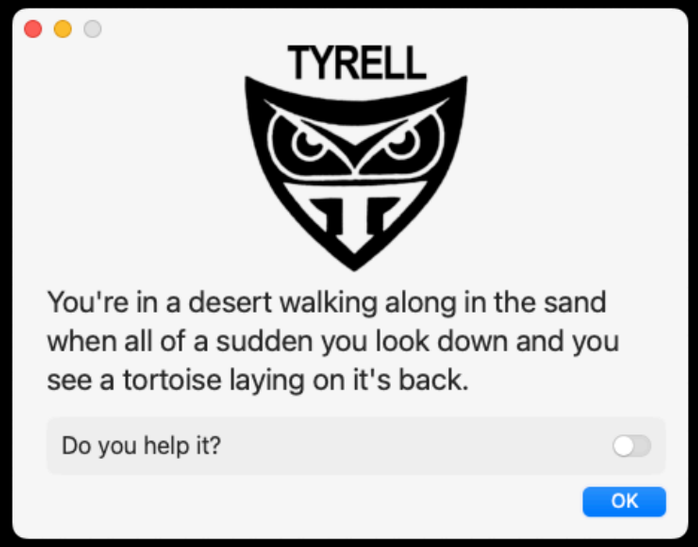
We have a free open source tool on our GitHub called Renew, and it does recurring reminders for users to restart their workstation. It works on deferrals and deadlines, similar to Nudge. It allows us to keep devices from getting absurd uptimes, while also allowing users some control over when that happens.

We also have an example framework for creating a deferral script for any other task you might want users to approve.

Installomator and swiftDialog

Why Choose swiftDialog?

- Survey your staff to easily identify replicants

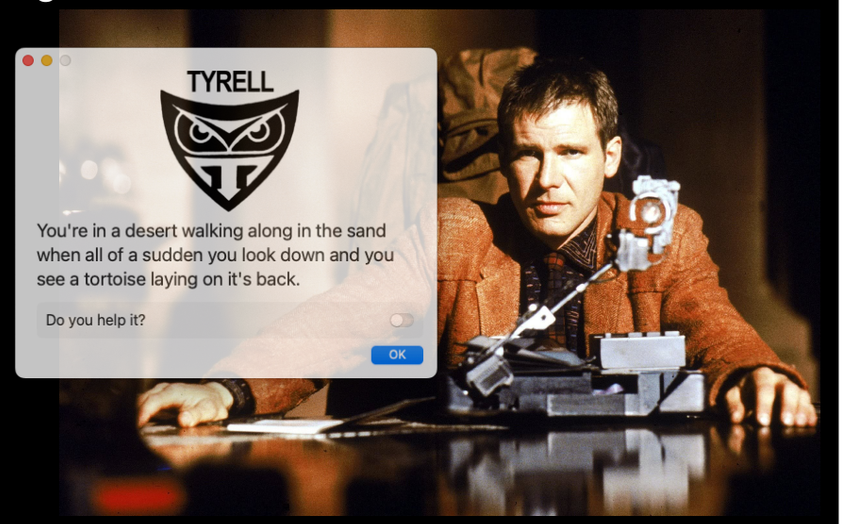


If you've ever wondered if one of your coworkers is a replicant, you can pepper them with ethical questions to try and initiate a hard lock up

Installomator and swiftDialog

Why Choose swiftDialog?

- **Survey your staff to easily identify replicants**



If you've ever wondered if one of your coworkers is a replicant, you can pepper them with ethical questions to try and initiate a hard lock up

Installomator and swiftDialog

Better together...

Deploy apps quickly with minimal overhead

Keep users informed of what's going on

How do we do it...

Installomator and swiftDialog

Better together...

- You want to deploy and update apps quickly and with minimal or no overhead...

Installomator and swiftDialog

Better together...

- You want to deploy and update apps quickly and with minimal or no overhead...
- **You want to keep your users informed of what's going on...**

Installomator and swiftDialog

Better together...

- You want to deploy and update apps quickly and with minimal or no overhead...
- You want to keep your users informed of what's going on...

- **So let's explore how we get there...**

Installomator

The one installer
script to rule them all.

Installomator

The one installer script to rule them all.

The Team

Armin Briegel
@scriptingosx

Adam Codega
@acodega

Isaac Ordonez
@isaacatmann

Søren Theilgaard
@Theile

Trevor Sysock
@BigMacAdmin

Installomator

The one installer script to rule them all.

The Team

And ~~Viewers~~ Mac Admins Like You...

Installomator

The one installer script to rule them all.

Over 800 community supported labels...

Installomator

The one installer script to rule them all.

Over 800 community supported labels...

Ask your CISO if Installomator is right for you…

Pro Indicators

- Latest is greatest
- User initiated installs
- Mobile fleet
- Low maintenance overhead
- No management server infrastructure

Not going to be right for every org

latest version from the developer at any given time

Great if you need low maintenance overhead

Great for no bandwidth constraints/mobile fleet

Ask your CISO if Installomator is right for you…

Pro Indicators

- Latest is greatest
- User initiated installs
- Mobile fleet
- Low maintenance overhead
- No management server infrastructure

Counter Indicators

- Version control
- Required test and approval process
- Local bandwidth constraints
- Tightly controlled app environment

Ask your CISO if Installomator is right for you…

Pro Indicators

- Latest is greatest
- User initiated installs
- Mobile fleet
- Low maintenance overhead
- No management server infrastructure

Counter Indicators

- Version control
- Required testing and approval process
- Local bandwidth constraints
- Tightly controlled app environment

AUTOPKG MUNKI

In the beginning…

Instlalomator came about around 2020.

Prior to this, admins had various scripts which all tended to do the same thing

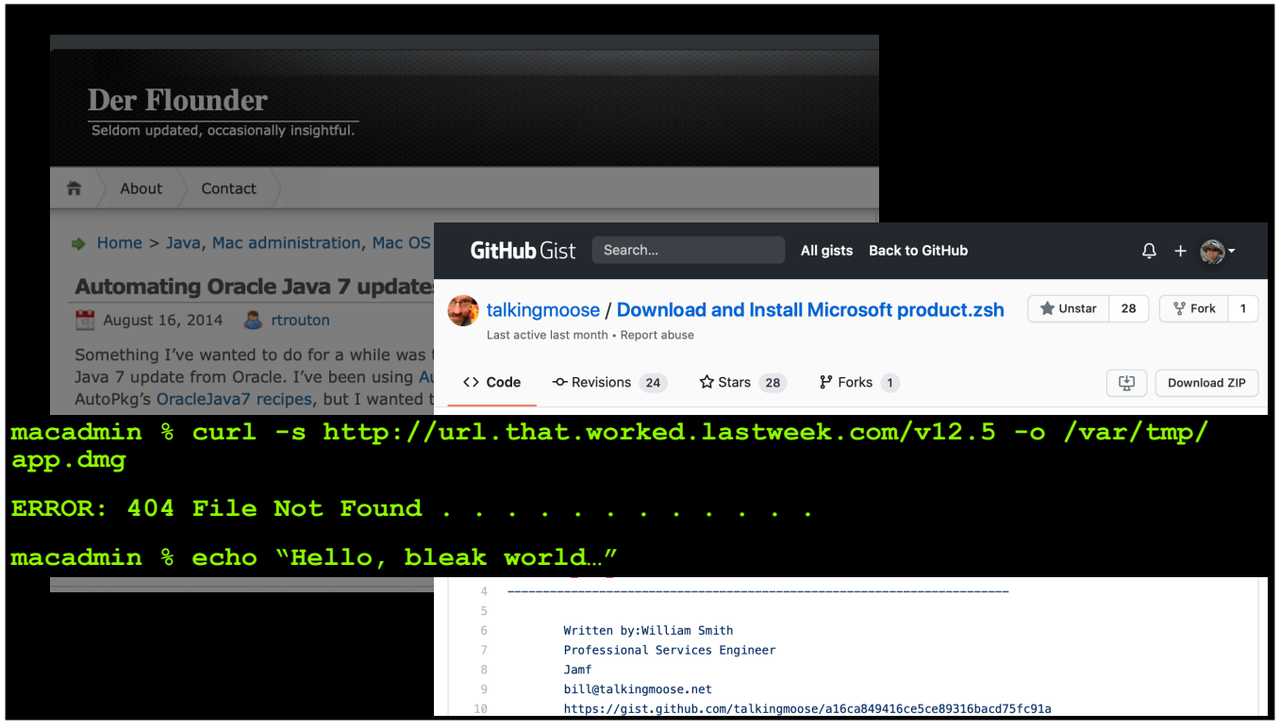
"Download this thing, verify this thing, install this thing, and cleanup after"

Instlalomator came about around 2020.

Prior to this, admins had various scripts which all tended to do the same thing

"Download this thing, verify this thing, install this thing, and cleanup after"

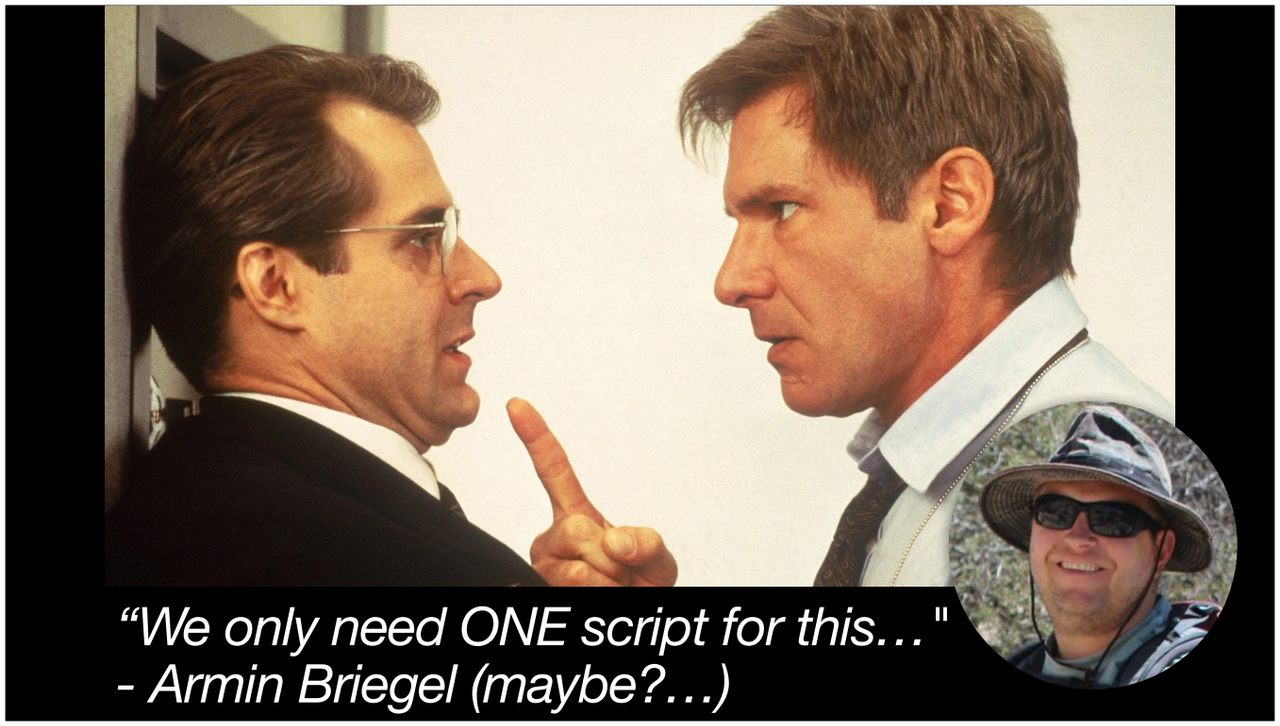
Most of the logic was the same in all of these different scripts used to install different products.


Der Flounder
Seldom updated, occasionally insightful.
About
Contact
Home > Java, Mac administration, Mac OS
Automating Oracle Java 7 update
August 16, 2014
rtrouton
Something I've wanted to do for a while was
Java 7 update from Oracle. I've been using Au
AutoPkg's OracleJava7 recipes, but I wanted t
GitHub Gist
Search...
All gists
Back to GitHub
talkingmoose / Download and Install Microsoft product.zsh
Last active last month • Report abuse
Unstar
28
Fork
1
Code
Revisions 24
Stars 28
Forks 1
Download ZIP
macadmin % curl -s http://url.that.worked.lastweek.com/v12.5 -o /var/tmp/app.dmg
ERROR: 404 File Not Found . . . . . .
macadmin % echo "Hello, bleak world…"
Written by:William Smith
Professional Services Engineer
Jamf
bill@talkingmoose.net
https://gist.github.com/talkingmoose/a16ca849416ce5ce89316bacd75fc91a

Armin Briegel tired of maintaining or not maintaining all of these various scripts.

Found a better way.

Installomator was born.


“We only need ONE script for this…"
- Armin Briegel (maybe?…)

Key Uses and Features

Installomator

- Initial Deployment
- Self-Service
- Application Patching

Key Uses and Features
Installomator

- Application Patching
  - Sometimes...

Key Uses and Features

Installomator

- Application Patching
  - If the vendor gives us a way to find the latest version number in a script
  - If the vendor doesn't give us better options, like an MDM profile to manage autoupdates.

Can still use installomator if no app version, but less efficient

Key Uses and Features

Installomator

- Application Patching
  - MDM profiles which manage a built in update tool might be the best first choice

Key Uses and Features

Installomator

- Application Patching
  - MDM profiles which manage a built in update tool might be the best first choice
    - Google Chrome
    - Zoom
    - Microsoft Office

One Script
Two Primary Components:

Core Script

Labels (Recipes)

Core - handles logic and workflow

labels - are recipes for individual applications

Core Script

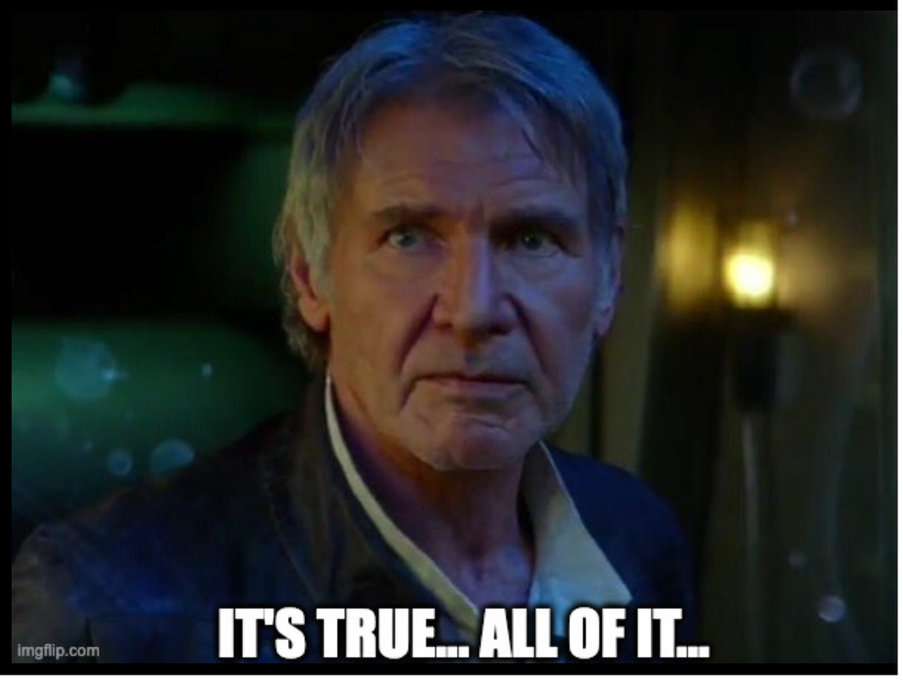
- About 1200 lines of shell code with Labels and Comments removed

Core Script

- Downloads the installer
- Validates authenticity of the download
- Interacts with the end user (sometimes)

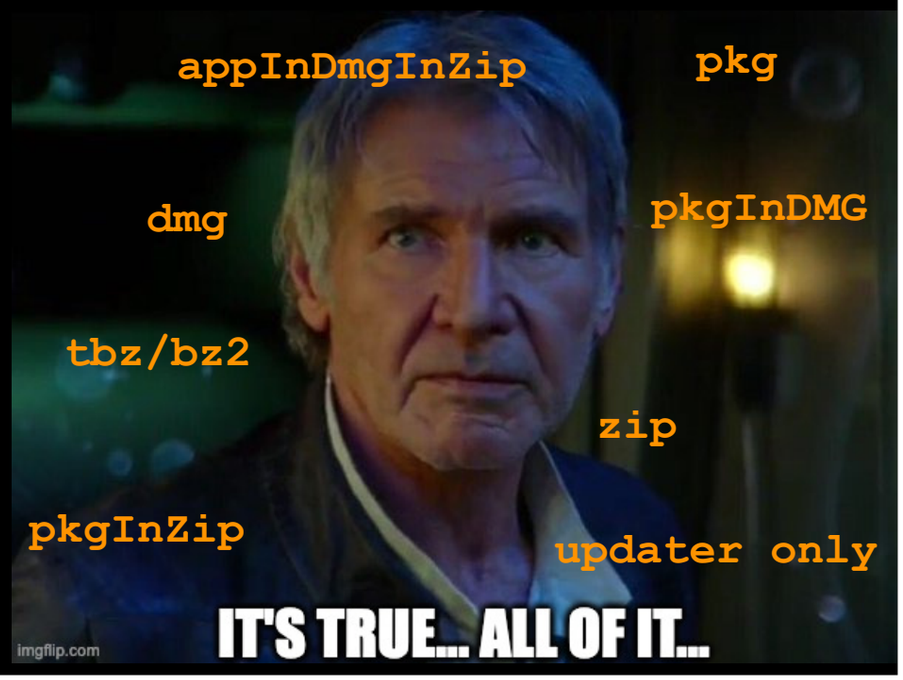
Core Script

Handles many kinds of macOS installer formats



Core Script

Handles many kinds of macOS installer formats



Labels

Name of the application - Usually thats what we see in our /Applications folder

Type of installer - Is it an app in a dmg? Is it a package? Is it a zipped dmg containing a package?
Installomator handles almost all of the crazy ways developers deliver their software.

downloadURL

TeamID: Which approved apple developer ID signs the software

There are additional optional label components, but one important one to speak about is app new version

This will be a string of code capable of determining the version number of the latest software offered by the developer.

Labels

- Fragments of a shell script with instructions

Labels

- Fragments of a shell script with instructions
- Required Label Components:
  - `name=` What is the software called

Labels

- Fragments of a shell script with instructions
- Required Label Components:
  - `name=` What is the software called
  - `type=` What kind of installer

Labels

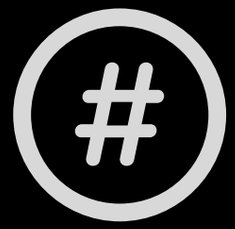
- Fragments of a shell script with instructions
- Required Label Components:
  - `name=` What is the software called
  - `type=` What kind of installer
  - `downloadURL=` Where do we find it

Labels

- Fragments of a shell script with instructions
- Required Label Components:
  - `name=` What is the software called
  - `type=` What kind of installer
  - `downloadURL=` Where do we find it
  - `expectedTeamID=` Who signed it

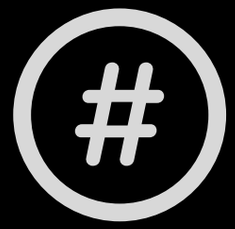
Labels

- Fragments of a shell script with instructions
- Required Label Components:
  - `name=` What is the software called
  - `type=` What kind of installer
  - `downloadURL=` Where do we find it
  - `expectedTeamID=` Who signed it
  - `appNewVersion=` What is the latest version



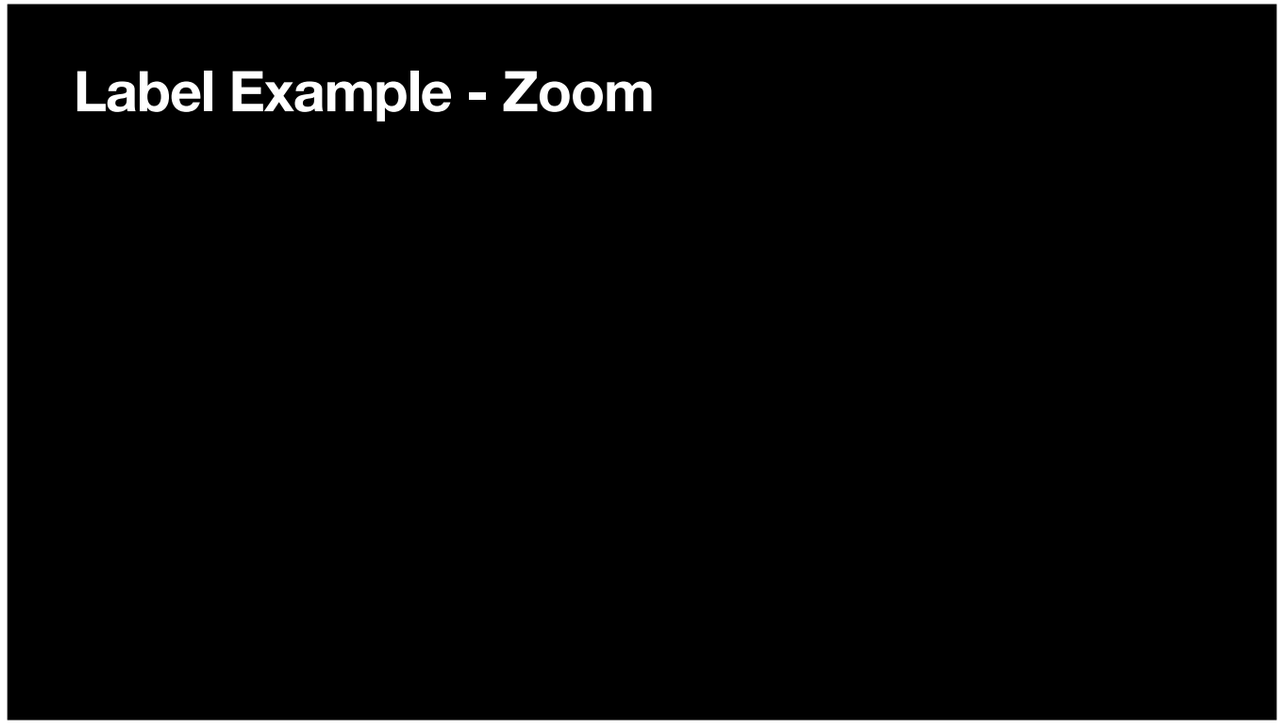
Labels

- Fragments of a shell script with instructions
- Required Label Components:
  - `name=` What is the software called
  - `type=` What kind of installer
  - `downloadURL=` Where do we find it
  - `expectedTeamID=` Who signed it
  - `appNewVersion=` What is the latest version



OPTIONAL

Label Example - Zoom



Zoom as an example

Kind enough to provide one single URL which always redirects to latest package

appNewVersion looks a little different. It's not a static value. We're going to drill in on that.

Label Example - Zoom

- `type="pkg"`

Label Example - Zoom

- `type="pkg"`
- `expectedTeamID="BJ4HAAB9B3"`

Label Example - Zoom

- type=“pkg”
- expectedTeamID="BJ4HAAB9B3"
- downloadURL=

  “https://zoom.us/client/latest/ZoomInstallerIT.pkg"

Label Example - Zoom

- type=“pkg”
- expectedTeamID="BJ4HAAB9B3"
- downloadURL=

  “https://zoom.us/client/latest/ZoomInstallerIT.pkg"
- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i ^location | cut -d "/" -f5)”

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)”`

First thing to notice $ and parenthesis

"This variable will be set to the result of the code thats contained within the parenthesis"

That code gets executed when we tell Installomator we want to install this software label

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)”`

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)”`

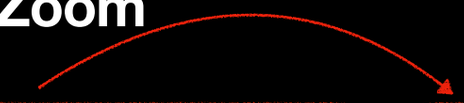
These vertical lines are called pipes, and they pass the results of the command to the left as input to the command to the right

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)”`

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"



In this case we're using curl to download something, and then passing the output of that command to "grep" which is a search tool.

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)”`

Curl is a tool used to download things at command line.

Here we're using specific options - not downloading the software

Get the HTTP headers only. The meta-data about the webpage.

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

-I, --head

Fetch the headers only! HTTP-servers feature the command HEAD which this uses to get nothing but the header of a document.

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

-L, --location

If the server reports that the requested page has moved to a different location… (follow it)

The L means "Hey if this is a redirect, follow it. I want the headers of the final destination of this URL"

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

```
HTTP/2 302

date: Mon, 27 May 2024 05:21:07 GMT

content-length: 0

location: https://cdn.zoom.us/prod/6.0.11.35001/ZoomInstallerIT.pkg

x-zm-trackingid: v=2.0;clid=aw1;rid=WEB_93ec12b95e09eebf8d71899f8d2cc0e8

x-content-type-options: nosniff

content-security-policy: upgrade-insecure-requests; default-src https://
*.zoom.us https://zoom.us blob: 'self'; img-src https: about: blob: data:
'self'; style-src https: safari-extension: chrome-extension: 'unsafe-inline'
… etc… etc… etc…
```

This is what the output looks like so far just running that curl command. You'll see we have a "location" line, which is the actual location of the latest zoom installer package

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

```
HTTP/2 302
date: Mon, 27 May 2024 05:21:07 GMT
content-length: 0
location: https://cdn.zoom.us/prod/6.0.11.35001/ZoomInstallerIT.pkg
x-zm-trackingid: v=2.0;clid=aw1;rid=WEB_95ec12b95e09eebf8d71899f8d2cc0e8
x-content-type-options: nosniff
content-security-policy: upgrade-insecure-requests; default-src https://
*.zoom.us https://zoom.us blob: 'self'; img-src https: about: blob: data:
'self'; style-src https: safari-extension: chrome-extension: 'unsafe-inline'
… etc… etc… etc…
```

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)”

```
location: https://cdn.zoom.us/prod/6.0.11.35001/
ZoomInstallerIT.pkg
```

using grep removes all lines except the one we're interested in

remember the end goal is to get to ONLY that version number you see in the URL

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

```
location: https://cdn.zoom.us/prod/6.0.11.35001/
ZoomInstallerIT.pkg
```

cut splits up this string of text based on a delimiter

forward slash is our delimiter which results in the string of text having 6 fields all separated by that forward slash

we want field 5, because hat contains our version number

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

```
location: https://cdn.zoom.us/prod/6.0.11.35001/
ZoomInstallerIT.pkg
```

Label Example - Zoom

- appNewVersion=
  "$(curl -fsIL ${downloadURL} | grep -i
  ^location | cut -d "/" -f5)"

1 2 3 4 5

location: https://cdn.zoom.us/prod/6.0.11.35001/
ZoomInstallerIT.pkg

6

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)"`

```
location: https://cdn.zoom.us/prod/6.0.11.35001/
ZoomInstallerIT.pkg
```

Label Example - Zoom

- `appNewVersion=`
  `"$(curl -fsIL ${downloadURL} | grep -i`
  `^location | cut -d "/" -f5)”`

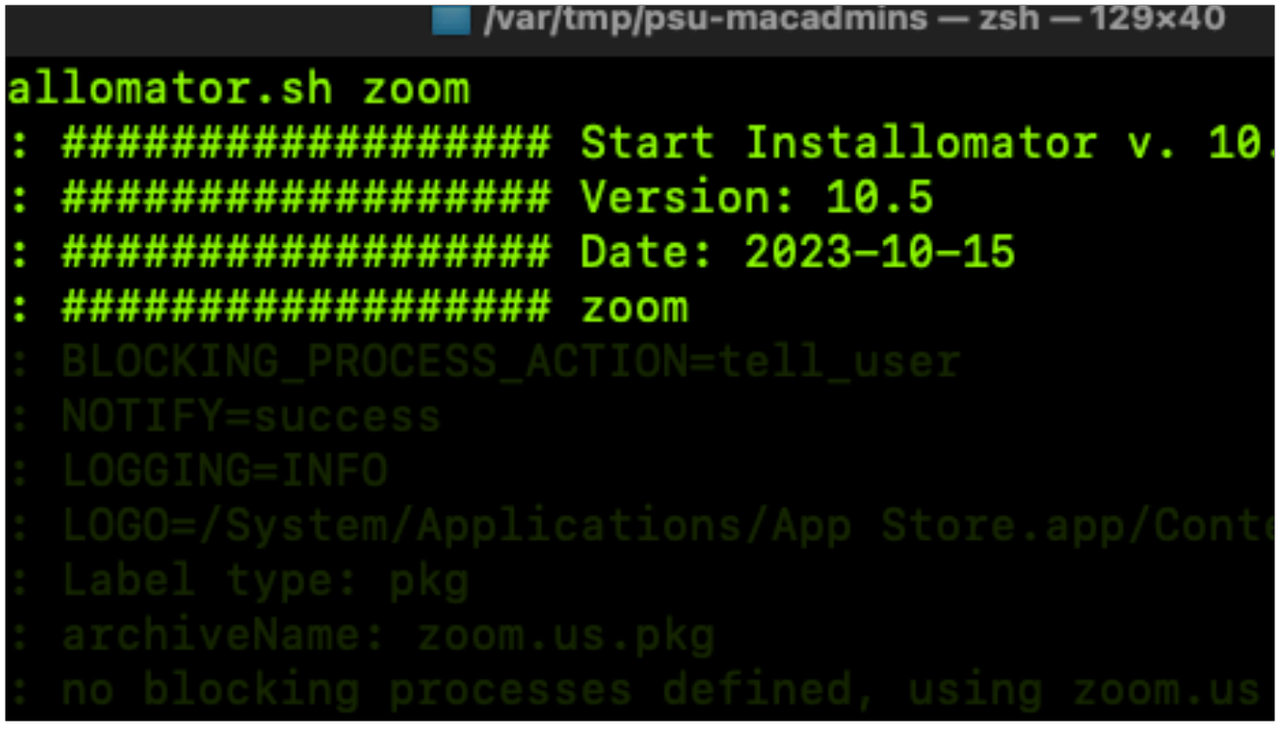
`appNewVersion=“6.0.11.35001”`

That makes the value of our variable equal to the latest version of zoom available.

```
/var/tmp/psu-macadmins — zsh — 129×40
psu % /usr/local/Installomator/Installomator.sh zoom
2024-05-24 21:39:53 : REQ   : zoom : ################## Start Installomator v. 10.5, date 2023-10-15
2024-05-24 21:39:53 : INFO  : zoom : ################## Version: 10.5
2024-05-24 21:39:53 : INFO  : zoom : ################## Date: 2023-10-15
2024-05-24 21:39:54 : INFO  : zoom : ################## zoom
2024-05-24 21:39:54 : INFO  : zoom : BLOCKING_PROCESS_ACTION=tell_user
2024-05-24 21:39:54 : INFO  : zoom : NOTIFY=success
2024-05-24 21:39:54 : INFO  : zoom : LOGGING=INFO
2024-05-24 21:39:54 : INFO  : zoom : LOGO=/System/Applications/App Store.app/Contents/Resources/AppIcon.icns
2024-05-24 21:39:54 : INFO  : zoom : Label type: pkg
2024-05-24 21:39:54 : INFO  : zoom : archiveName: zoom.us.pkg
2024-05-24 21:39:54 : INFO  : zoom : no blocking processes defined, using zoom.us as default
2024-05-24 21:39:55 : INFO  : zoom : name: zoom.us, appName: zoom.us.app
2024-05-24 21:39:55.025 mdfind[15635:335607] [UserQueryParser] Loading keywords and predicates for locale "en_US"
2024-05-24 21:39:55.025 mdfind[15635:335607] [UserQueryParser] Loading keywords and predicates for locale "en"
2024-05-24 21:39:55.108 mdfind[15635:335607] Couldn't determine the mapping between prefab keywords and predicates.
2024-05-24 21:39:55 : WARN  : zoom : No previous app found
2024-05-24 21:39:55 : WARN  : zoom : could not find zoom.us.app
2024-05-24 21:39:55 : INFO  : zoom : appversion:
2024-05-24 21:39:55 : INFO  : zoom : Latest version of zoom.us is 6.0.11.35001
2024-05-24 21:39:55 : REQ   : zoom : Downloading https://zoom.us/client/latest/ZoomInstallerIT.pkg to zoom.us.pkg
2024-05-24 21:40:15 : REQ   : zoom : no more blocking processes, continue with update
2024-05-24 21:40:15 : REQ   : zoom : Installing zoom.us
2024-05-24 21:40:15 : INFO  : zoom : Verifying: zoom.us.pkg
2024-05-24 21:40:16 : INFO  : zoom : Team ID: BJ4HAAB9B3 (expected: BJ4HAAB9B3 )
2024-05-24 21:40:16 : INFO  : zoom : Installing zoom.us.pkg to /
2024-05-24 21:40:32 : INFO  : zoom : Finishing...
2024-05-24 21:40:35 : INFO  : zoom : App(s) found: /Applications/zoom.us.app
2024-05-24 21:40:35 : INFO  : zoom : found app at /Applications/zoom.us.app, version 6.0.11.35001, on versionKey CFBundleVersion
2024-05-24 21:40:35 : REQ   : zoom : Installed zoom.us, version 6.0.11.35001
2024-05-24 21:40:35 : INFO  : zoom : notifying
2024-05-24 21:40:36 : INFO  : zoom : Installomator did not close any apps, so no need to reopen any apps.
2024-05-24 21:40:36 : REQ   : zoom : All done!
2024-05-24 21:40:36 : REQ   : zoom : ################## End Installomator, exit code 0

psu %
```

What the output looks like



Run our label. First block is metadata about the Installomator script and what label we've chosen to run

```
: ################### Date: 2023-10-15
: ################### zoom
: BLOCKING_PROCESS_ACTION=tell_user
: NOTIFY=success
: LOGGING=INFO
: LOGO=/System/Applications/App Store.app/Conte
: Label type: pkg
: archiveName: zoom.us.pkg
: no blocking processes defined, using zoom.us
: name: zoom.us, appName: zoom.us.app
5:335607] [UserQueryParser] Loading keywords ar
5:335607] [UserQueryParser] Loading keywords ar
5:335607] Couldn't determine the mapping betwe
```

Next we see options which are part of the label and options which we set at run time

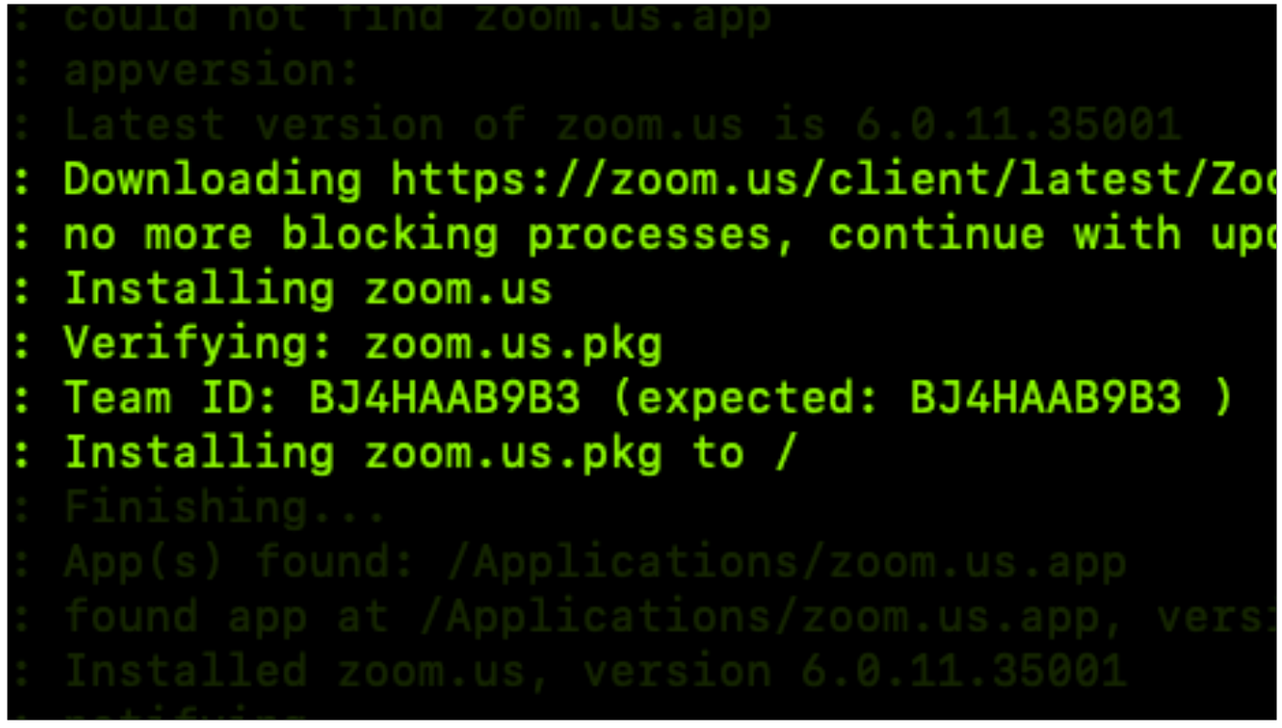
In this case, we didn't apply any options aside from the label, so this is confirming the default options that are applied

```
: no blocking processes defined, using zoom.us
: name: zoom.us, appName: zoom.us.app
5:335607] [UserQueryParser] Loading keywords ar
5:335607] [UserQueryParser] Loading keywords ar
5:335607] Couldn't determine the mapping betwee
: No previous app found
: could not find zoom.us.app
: appversion:
: Latest version of zoom.us is 6.0.11.35001
: Downloading https://zoom.us/client/latest/Zoo
: no more blocking processes, continue with upd
: Installing zoom.us
: Verifying: zoom.us.pkg
```

Installomator is looking on the device to see what version is currently installed.

If the latest version is already there, Installomator will exit at this point.

In this example, it's a fresh install, so no appversion was found.



Here we are downloading the pkg

Verifying the pkg and team ID

and then Installing the PKG

```
: Verifying: zoom.us.pkg
: Team ID: BJ4HAAB9B3 (expected: BJ4HAAB9B3 )
: Installing zoom.us.pkg to /
: Finishing...
: App(s) found: /Applications/zoom.us.app
: found app at /Applications/zoom.us.app, versi
: Installed zoom.us, version 6.0.11.35001
: notifying
: Installomator did not close any apps, so no r
: All done!
: #################### End Installomator, exit co
```

Finally last we're finishing up, reporting what happened, notifying the user, and cleaning up the installer pkg

```
.
d: /Applications/zoom.us.app
t /Applications/zoom.us.app, version 6.0.11.350
oom.us, version 6.0.11.35001

or did not close any apps, so no need to reoper

######## End Installomator, exit code 0
```

Finally last we're finishing up, reporting what happened, notifying the user, and cleaning up the installer pkg

```
d: /Applications/zoom.us.app
t /Applications/zoom.us.app, version 6.0.11.350
oom.us, version 6.0.11.35001

or did not close any apps, so no need to reoper

######## End Installomator, exit code 0
```

Deploying Installomator

- PKG release contains all community contributed labels
- `main` branch has latest approved labels, even if a full PKG release hasn't been made

The package is on the Releases page of the Github and includes all of the latest approved labels as of the time of that release

the `main` branch on GitHub has more up to date community submissions/fixes, you can go there if the PKG release is lagging behind

Deploying Installomator

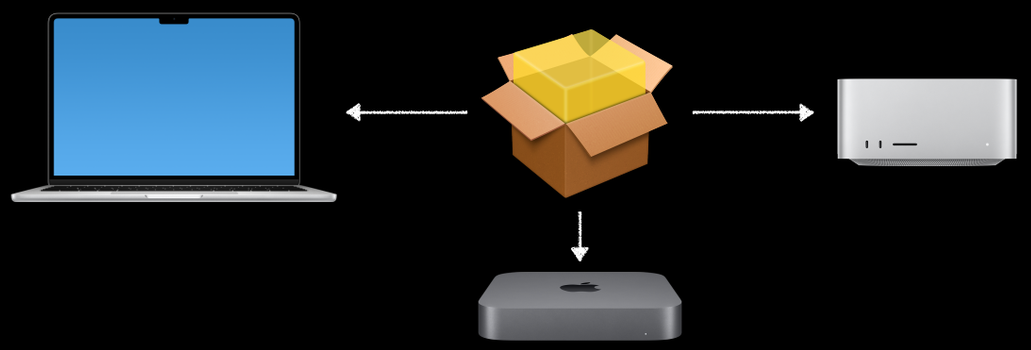
- The Github repo has tools to easily generate your own copy of Installomator.sh
- `/utils/assemble.sh`
  - Create your own Installomator.sh or PKG

You do not have to "YOLO" any community code you have not reviewed. You do not have to include all default labels in "your" version of Installomator.

The utilities folder on github has an assemble script which lets you build your own Installomator.sh or package and you can easily customize to include only the labels you have tested or written yourself.

Deploying Installomator

- Installomator.pkg will place the script onto a workstation

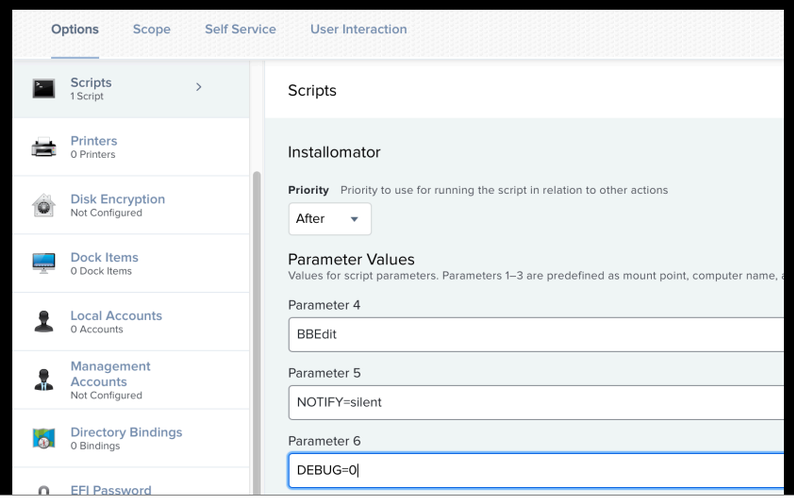


If you install the provided package, it will place the Installomator script onto your workstations.

No agents or daemons, no post installs, all it does it place the script so that it can be used on demand

Deploying Installomator

- With Jamf, the script lives on the server



If you're a jamf admin, your script lives in jamf and you can create policies with parameters that contain your options

Deploying Installomator

- Use scripting tools in your management platform to initiate runs of Installomator.sh

For other management systems you'll install the script to the device, and then use the scripting tools in your management system to call that script with your desired options.

this is an example of installing Nudge using mosyle

Options at Runtime

Command line options at runtime define how Installomator behaves, and what the user experience is

Options at Runtime

- User interactions and notifications
- What conditions to install or bail
- Overwrite (or ignore) App Store apps

Override Anything

Any label value can be passed as an argument when you run the script.

Override Anything

An older version of swiftDialog runs on macOS 11, you could override the downloadURL to deploy that old version on old machines

If you wanted to help test the latest version of Nudge, you can configure the URL for the latest beta release

Or if a label is broken or needs changing in any other way, you can pass these options while you wait for the Installomator team to update the recipe on the official repo

Override Anything

- “Pin” your download URL to an old version for OS compatibility
- Configure the URL for a beta release
- Fix labels that need updates between Installomator releases

Security and Verification

We are downloading code from the internet and running it, after all

Security and Verification

Installomator uses built in macOS Security Tools

Uses 2 key built in macOS security features

Security and Verification

Installomator uses built in macOS Security Tools

- **Signing**
  - **Verifies the identity of the developer that signed the app or pkg.**
  - **Verifies that the contents of the app/pkg is what the developer built.**
- **Notarization**
  - **The app or pkg has been submitted to Apple for malware scanning.**

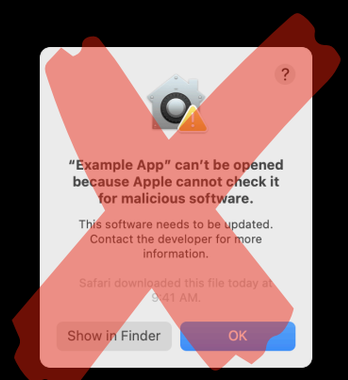
Security and Verification

Installomator uses built in macOS Security Tools

- Signing
  - Verifies the identity of the developer that signed the app or pkg.
  - Verifies that the contents of the app/pkg is what the developer built.
- Notarization
  - The app or pkg has been submitted to Apple for malware scanning.

Security and Verification

Installomator uses built in macOS Security Tools



If an app is not both signed and notarized, Installomator will not install it

Security and Verification

Installomator uses built in macOS Security Tools

		Gatekeeper	Installomator
Apple Developer ID Signature	Data Integrity	✅	✅
Notarization	Malware Scan	✅	✅
Expected Team ID	Origin		✅

You may say "but my users are admins, they can install stuff on their own and we already use gatekeeper"

Installomator adds two additional verification steps to ensure things go as planned

Expected Team ID means we know who signed the package. Any developer can sign a package and name it "Microsoft Office" and stick it up on a website somewhere

We're also providing the origin of the package. We know we're downloading the software from the locations defined in the label recipes.

Security and Verification

Installomator uses built in macOS Security Tools

		Gatekeeper	Installomator
Apple Developer ID Signature	Data Integrity	✅	✅
Notarization	Malware Scan	✅	✅
Expected Team ID	Origin		✅

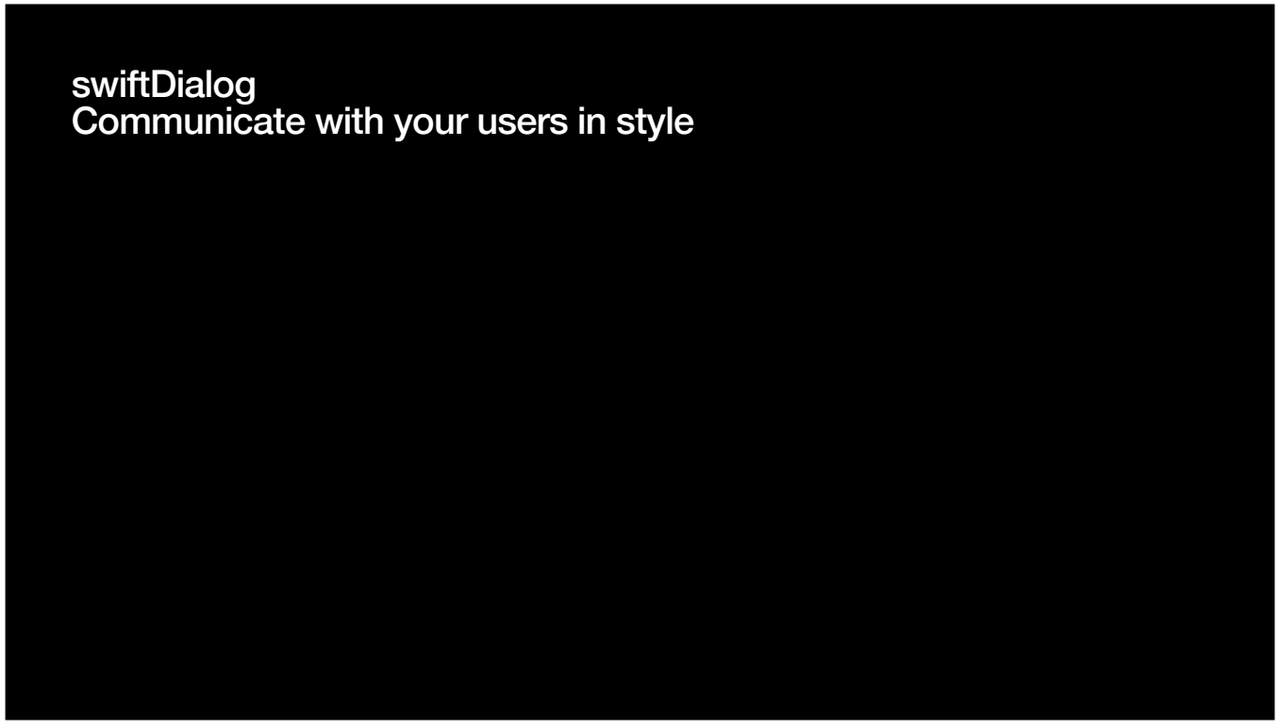
Q&A
Installomator

That is my overview of Installoamtor, I do want to pause at this point to see if anyone has questions about what we've covered so far.

swiftDialog

Communicate with
your users in style

Wipe all that from your brains for the moment as we pivot to the second tool today, SwiftDialog



Works for the CSIRO

Australian Government agency responsible for scientific research.

swiftDialog
Communicate with your users in style

The Man, the Myth, the Legend

Bart Reardon
@bartreardon

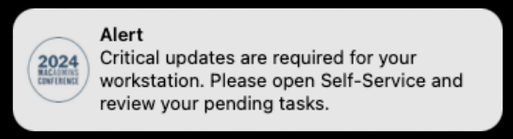
swiftDialog Use Cases

Native macOS notifications

gather information from your users, and use that information to fuel the logic of your management scripts

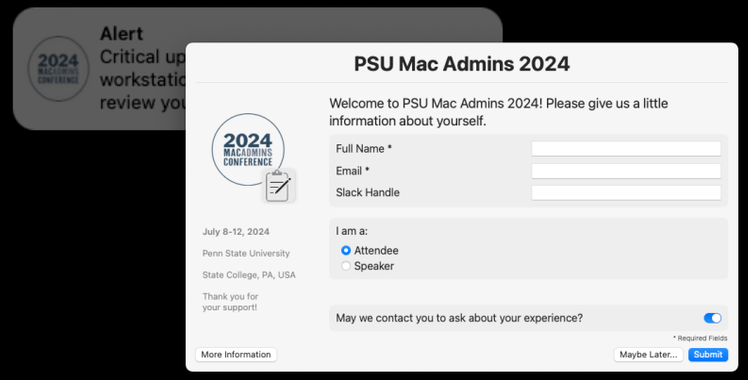
notify of ongoing processes
"This is what i'm doing to your computer and how it's going"

swiftDialog Use Cases



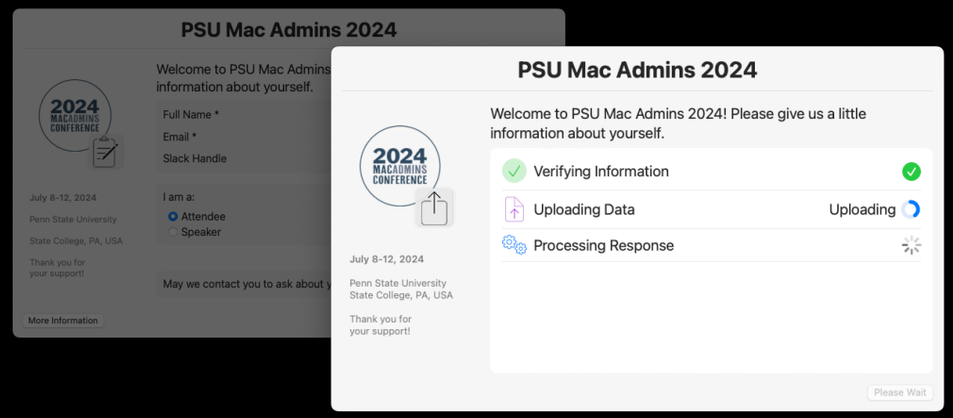
Native macOS Notifications

swiftDialog Use Cases



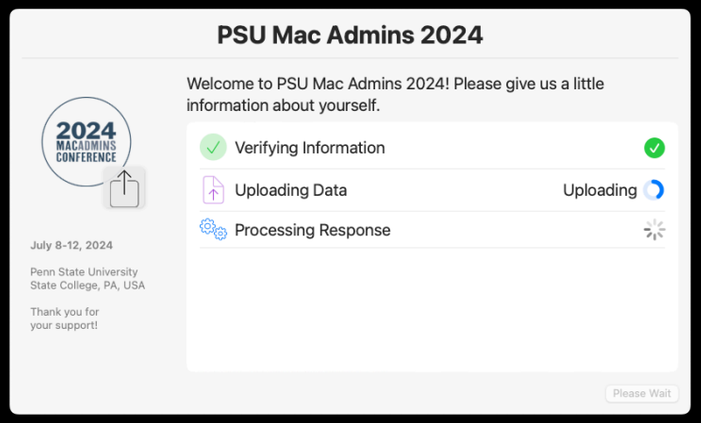
Gather information

swiftDialog Use Cases



Notify of ongoing processes

swiftDialog Use Cases



Notifying of ongoing processes is what the Installomator integration relies on

This is our focus, but we have some fundamentals we need to establish first

Deploy swiftDialog to Your Fleet

Custom App Icon is what shows in your notifications

Notifications will never be seen unless you also deliver a profile, unless the user goes and allows them

Unless you slept through the first part, you'll know about at least one useful tool to deploy swiftDialog

Deploy swiftDialog to Your Fleet

- Customize the App Icon
- MDM Profile for Notifications

OPTIONAL

Deploy swiftDialog to Your Fleet

- Customize the App Icon
- MDM Profile for Notifications
- **Deploy the PKG**

OPTIONAL

```
#!/bin/bash

/usr/local/Installomator/Installomator.sh \
    swiftdialog \
    BLOCKING_PROCESS_ACTION=ignore \
    NOTIFY=silent \
```

swiftDialog
Two Primary Components:

App Bundle

Binary

App Bundle

Binary

App bundle lives here in application support and you generally will not interact with it

The binary is the command you'll call in your scripts, and it does some clever things to run things as the logged in user even if your script is running as root

App Bundle

- /Library/Application Support/Dialog/Dialog.app
- Not typically used directly

Binary

App Bundle

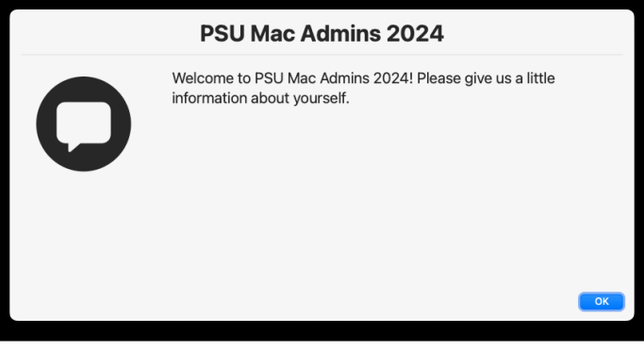
- /Library/Application Support/Dialog/Dialog.app
- Not typically used directly

Binary (it's actually a shell script, shhhhh…)

- /usr/local/bin/dialog
  - Use this path in your scripts
  - Runs Dialog "As User" even if run by sudo/root

Our First Dialog

```
/usr/local/bin/dialog \
    --title "PSU Mac Admins 2024" \
    --message "Welcome to PSU Mac Admins 2024! Please give us a little information about yourself." \
```



First basic dialog window

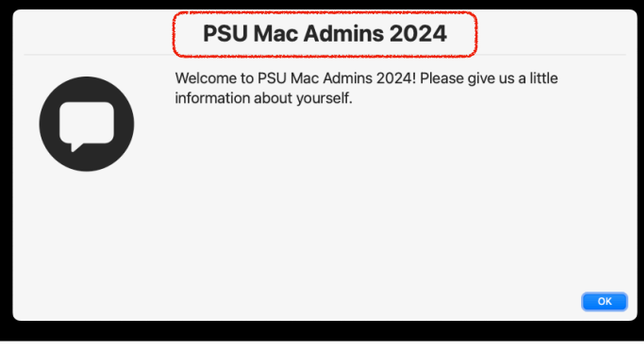
Have configured the title and the message only

You may also notice the forward slashes at the end of each line.

This is a shell script trick to make things more readable. It allows us to spread one single command across many lines. Without this, your scripts (or your keynote) can become very difficult to read.

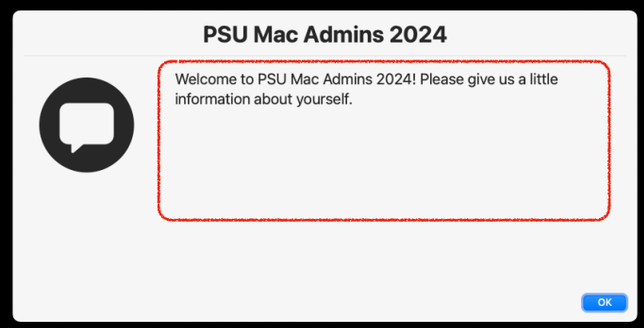
Our First Dialog

```
/usr/local/bin/dialog \
    --title "PSU Mac Admins 2024" \
    --message "Welcome to PSU Mac Admins 2024! Please give us a little information about yourself." \
```



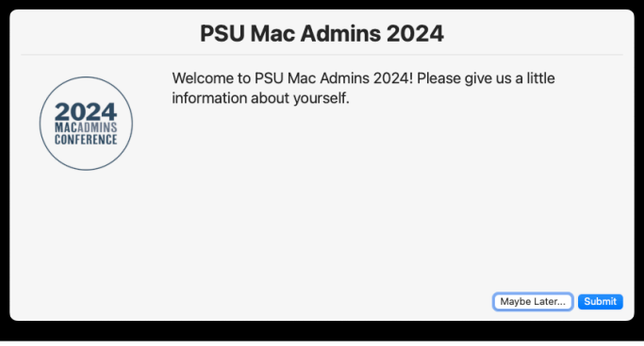
Our First Dialog

```
/usr/local/bin/dialog \
    --title "PSU Mac Admins 2024" \
    --message "Welcome to PSU Mac Admins 2024! Please give us a little information about yourself." \
```



Our First Dialog

```
--icon "./PSU.png" \
--button1text "Submit" \
--button2text "Maybe Later..." \
--ontop --moveable \
```



Now we're adding additional options here for the icon, button text

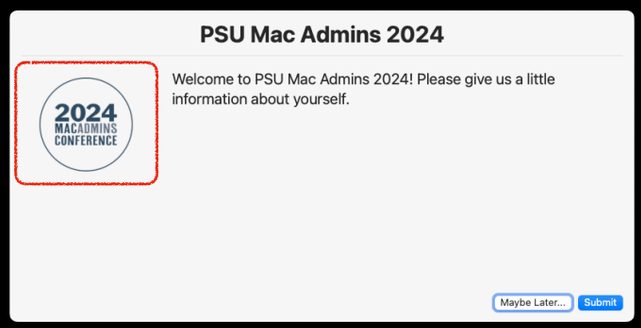
Notice that I have a second button now, since I defined that button text

Pretty much always use onto and moveable.

You don't want your window getting lost behind chrome or MS word, it can tie up your management agents waiting for user interaction

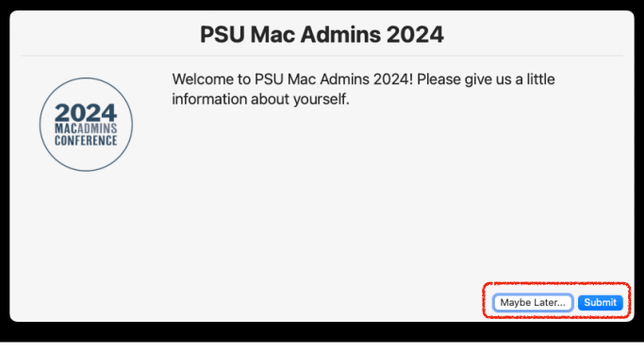
Our First Dialog

```
--icon "./PSU.png" \
--button1text "Submit" \
--button2text "Maybe Later..." \
--ontop --moveable \
```



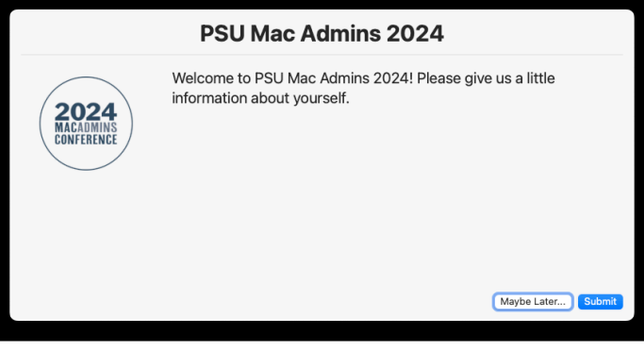
Our First Dialog

```
--icon "./PSU.png" \
--button1text "Submit" \
--button2text "Maybe Later..." \
--ontop --moveable \
```



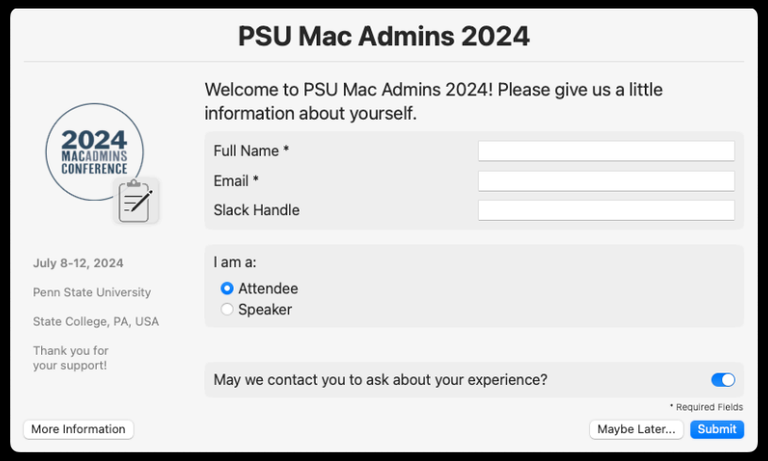
Our First Dialog

```
--icon "./PSU.png" \
--button1text "Submit" \
--button2text "Maybe Later..." \
--ontop --moveable \
```



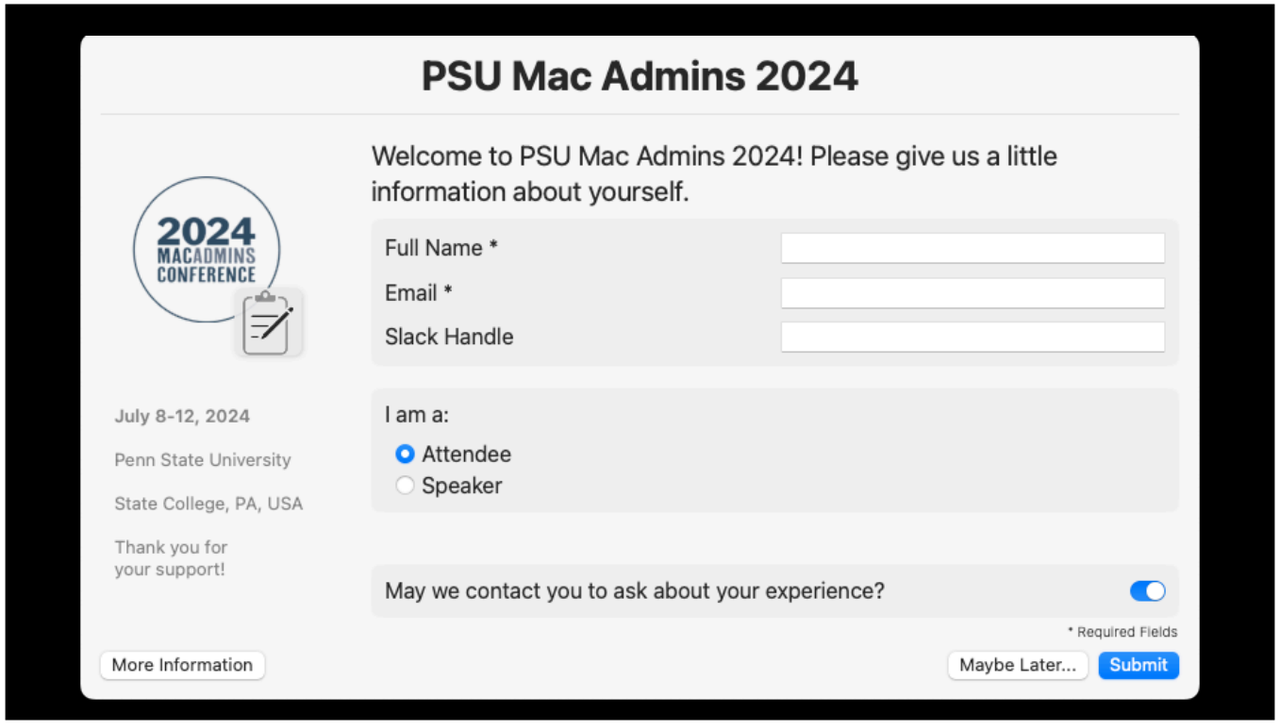
Our First Dialog

`sudo "Finish My Script"`



Using abra cadabra magíc, I can finish making a very robust and professional looking customized dialog using many of the options at once

Each field of input could be used later in your script.


PSU Mac Admins 2024
2024 MACADMINS CONFERENCE
Welcome to PSU Mac Admins 2024! Please give us a little information about yourself.
Full Name *
Email *
Slack Handle
July 8-12, 2024
Penn State University
State College, PA, USA
Thank you for your support!
I am a:
Attendee
Speaker
May we contact you to ask about your experience?
* Required Fields
More Information
Maybe Later...
Submit

Json Support

```
{
    "title": "PSU Mac Admins 2024",
    "message": "Welcome to PSU Mac Admins 2024! Please give us a little information about yourself.",
    "icon": "2024 MacAdmins Logo-Circle-Blue.png",
    "overlayicon": "SF=pencil.and.list.clipboard",
    "ontop": true,
    "moveable": true,
    "height": 460,
    "button1text": "Submit",
    "button2text": "Maybe Later...",
    "infobutton": true,
    "infobox": "infobox.md",
    "vieworder": "textfield,radiobutton,checkbox",
    "textfield" : [
        { "title" : "Full Name","required" : true },
        { "title" : "Email","required" : true },
        { "title" : "Slack Handle","required" : false }
    ],
    "checkbox": [
        {
            "label": "May we contact you to ask about your experience?",
```

When you get as complex as that previous window, a command line argument with all of those options becomes unmanageable even with the forward slash trick

For those cases, you can create a json text file and include all options in that file

When you do, your final command can end up very simple.

Json Support

```
    ],
    "checkbox": [
        {
            "label": "May we contact you to ask about your experience?",
            "checked": true,
            "disabled": false
        },
    ],
    "checkboxstyle": {
        "style": "switch",
        "size": "regular"
    },
    "selectitems" : [
        {"title" : "I am a:", "values" : ["Attendee","Speaker"], "style" : "radio"}
    ]
}

```

```
/usr/local/bin/dialog --jsonfile PSU-Registration.json
```

Command Files

Modify Existing swiftDialog Windows With Updated Information

So far we've looked at static message window and information gathering, but as i said our primary focus is on "real time ongoing updates"

Command Files

First we draw our initial Dialog window

notice this time, i'm still using the icon argument but I have a URL instead of a file path. SwiftDialog supports this

For this example, I have a process which I know consists of 4 steps. I'm going to call a progress bar and use the number 4 to declare this

I'm also using "mini" mode, which is a swiftdialog feature that draws a smaller window than what we saw earlier and changes the layout

Last, you will want to see that I'm using the ampersand. This is not a swiftDialog feature, this is a shell scripting technique

By default, scripts do one thing, wait for that to complete, and then do the next thing, and so on.

This ampersand tells your script "Hey do this thing (in this case, draw our dialog window), and let it keep running in the background while we move on to other stuff"

We start with a bouncing progress bar until we tell it to update as we complete steps.

Command Files

First we draw our initial Dialog window

```
/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --moveable --ontop --mini &
```

Command Files

First we draw our initial Dialog window

```
/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --moveable --ontop --mini &
```

Command Files

First we draw our initial Dialog window

```
/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --moveable --ontop --mini &
```

Command Files

First we draw our initial Dialog window

```
/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --moveable --ontop --mini &
```

Command Files

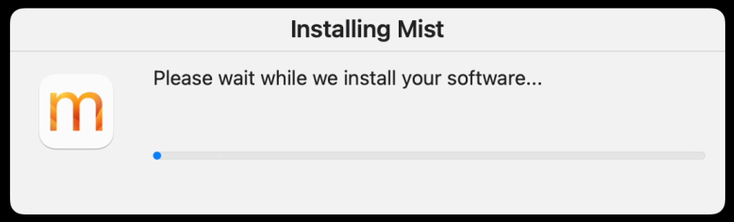
First we draw our initial Dialog window

```
/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --moveable --ontop --mini &
```

Command Files

First we draw our initial Dialog window

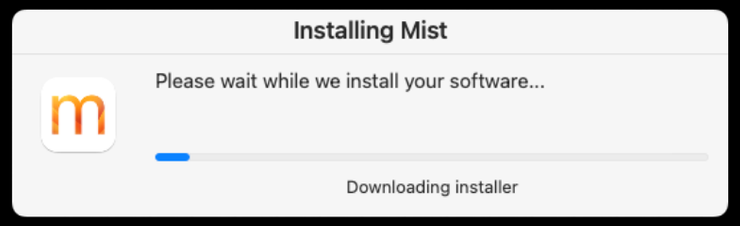
```
/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --moveable --ontop --mini &
```



Command Files

Update the progress text

```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
```



So how do we tell it?

echo is a basic command line tool that just prints whatever text you provide it.

Here we're going too use echo to print a specially formatted line of text, and add that line of text to a special designated file

We're sending "progress text" here to update the window to show that we're in the download phase

This double arrow means "Send the output of this command to a file.

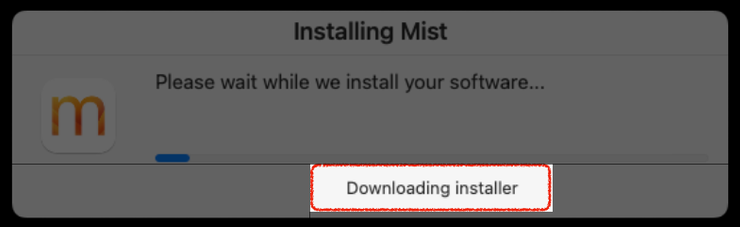
One arrow would mean overwrite the file, two means add it to the bottom of the file as new lines.

The file name after those arrows is the destination for those new lines. In this case, we're sending it to the special default command file that all swiftDialog windows watch for updates.

Command Files

Update the progress text

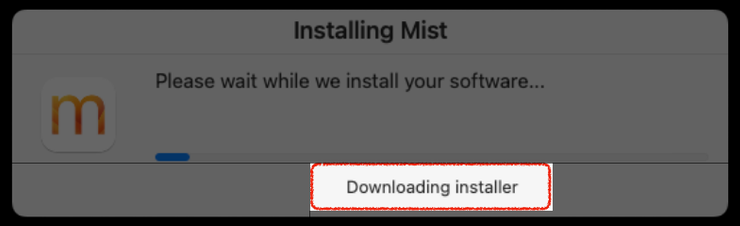
```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
```



Command Files

Update the progress text

```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
```

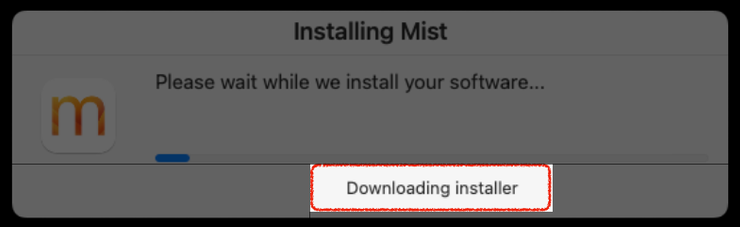


Command Files

Update the progress text

Default Command File

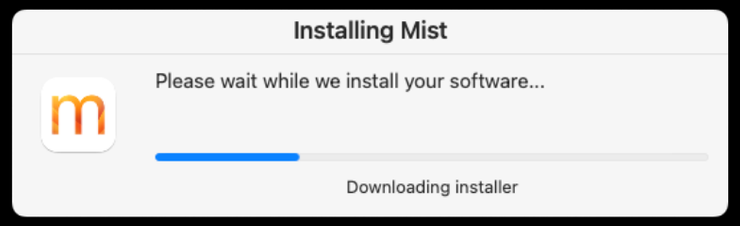
```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
```



Command Files

Increment our progress bar

```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1
```

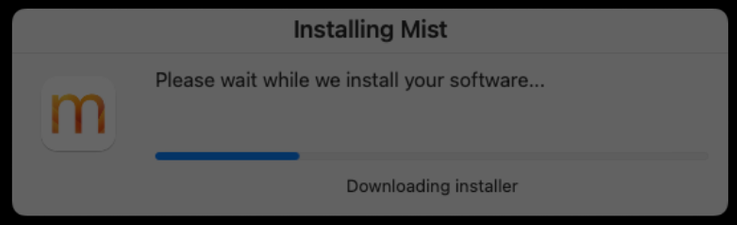


Next we're going to wait just .1 seconds using the sleep command. This allows swiftDialog just enough time to process the previous command before we add yet another

Command Files

Increment our progress bar

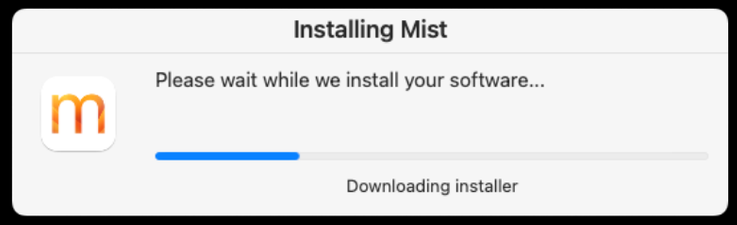
```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1
```



Command Files

Increment our progress bar

```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1
```



This next command is to increment the progress bar

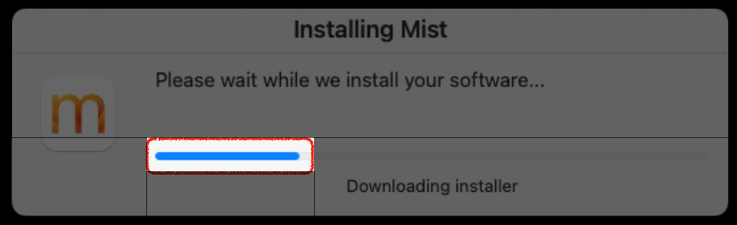
you'll remember that we declared 4 steps when we called the window, so increment moves the progress to 25%

You can also move the progress bar to any specific number you may want

Command Files

Increment our progress bar

```
echo "progresstext: Downloading installer" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment" >> /var/tmp/dialog.log
sleep .1
```

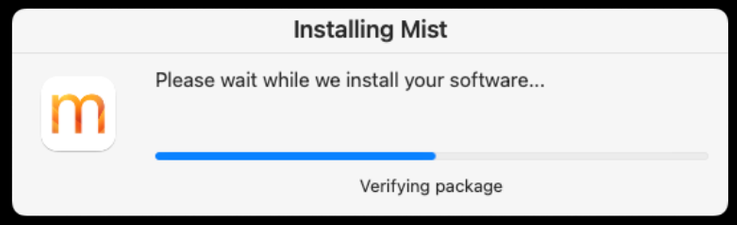


Command Files

Continue to update our text and our progress bar

```
echo "progresstext: Verifying package" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1

# Put your commands for this step here, between where you're updating
the progress
```



Now we're continuing our script. you'll want to imaging we're putting commands that do the actual business in between these updates to the command file.

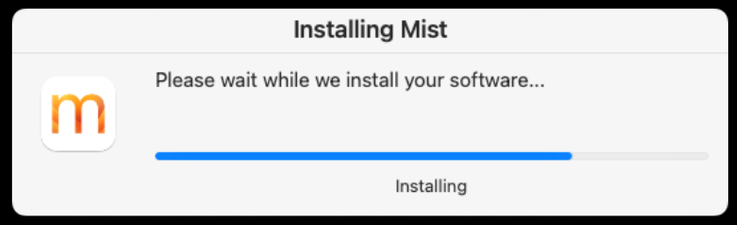
We're verifying the download

Command Files

Continue to update our text and our progress bar

```
echo "progresstext: Installing" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1

# Put your commands for this step here, between where you're updating
the progress
```



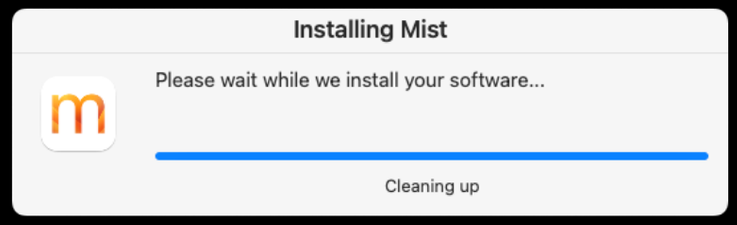
Now we're Installing what we downloaded

Command Files

Complete our process bar and close the Dialog

```
echo "progresstext: Cleaning up" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1

echo "quit:" >> /var/tmp/dialog.log
```



and now we're cleaning up after ourselves, which is the 4th and last step

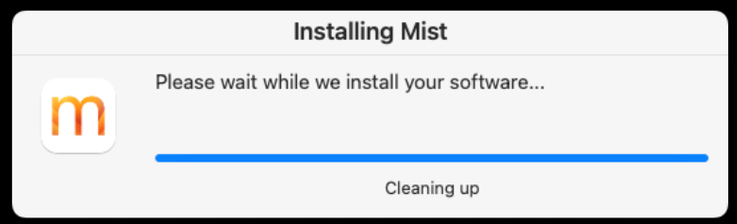
I can now close the dialog window which we sent to the background previously by sending a quit command to the dialog

Command Files

Complete our process bar and close the Dialog

```
echo "progresstext: Cleaning up" >> /var/tmp/dialog.log
sleep .1
echo "progress: increment” >> /var/tmp/dialog.log
sleep .1

echo "quit:" >> /var/tmp/dialog.log
```



Command Files

Avoid repeating code by using functions

```
function dialog_command(){
    # $1 is the command we want to send
    # to our running dialog window
    echo "${1}" >> "$dialogCommandFile"
    sleep .1
}
```

That was a lot of repeating code. We were using echo and redirecting the output and then using sleep to wait .1 seconds.

In shell scripting, any time you're repeating code like that you should consider using a function instead. This allows us to write that code once and call it as many times as we need.

Command Files

Avoid repeating code by using functions

```
dialog_command "progresstext: Installing"

dialog_command "progress: increment”

# Do the install things here...

dialog_command "progresstext: Cleaning up"

dialog_command "progress: increment"

# Do the cleanup things here...

dialog_command "quit:"
```

This function greatly simplifies sending updates to a running dialog window.

Command Files

Make unique temporary command files

```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"
```

The previous example used the default special file that all swiftDialog windows watch for updates.

This can cause issues in some edge cases, and I prefer to use a custom path for every run of a script.

Here we're going to generate a temporary text file with a file path unique to each run

just like earlier when we were building the instlalomator label, we have a dollar sign and parenthesis which means the variable takes the value of the output of the command within

the mktemp command creates an empty text file. the X's become randomly generated characters, and the command output is the name of the temporary file

Command Files

Make unique temporary command files

```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"
```

Command Files

Make unique temporary command files

```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"
```

Command Files

Make unique temporary command files

```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"


dialogCommandFile="/var/tmp/Example.JAqbR"
```

Command Files

Make unique temporary command files

```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"
```

Going back to our previous example, here's the same initial dialog command we used except this time we're telling dialog to use this custom file path as the command file

Command Files

Make unique temporary command files

```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"

/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --commandfile “$dialogCommandFile” \
    --moveable --ontop --mini &
```

Command Files

Make unique temporary command files

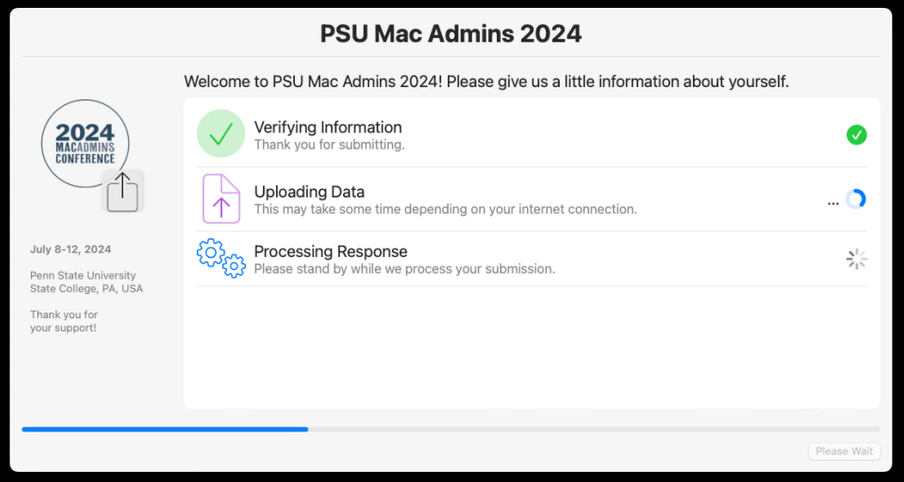
```
dialogCommandFile="$(mktemp /var/tmp/Example.XXXXX)"

/usr/local/bin/dialog \
    --title "Installing Mist" \
    --message "Please wait while we install your software..." \
    --icon "https://raw.githubusercontent.com/ninxsoft/Mist/main/
README%20Resources/App%20Icon.png" \
    --button1disabled \
    --progress 4 \
    --commandfile “$dialogCommandFile” \
    --moveable --ontop --mini &
```

Dialog Lists

Dialog Lists

Each item has it's own icon, text, and progress indicator



Dialog Lists

Add list items in shell or using a json

```
/usr/local/bin/dialog \
     --listitem "Verifying Information"
```

Dialog Lists

Add list items in shell or using a json

```
/usr/local/bin/dialog \

      --listitem "Verifying Information"
```

```
/usr/local/bin/dialog \

     --listitem "Verifying
Information",icon=SF=checkmark.circle.fill,color=green,subtitle="
```

Dialog Lists

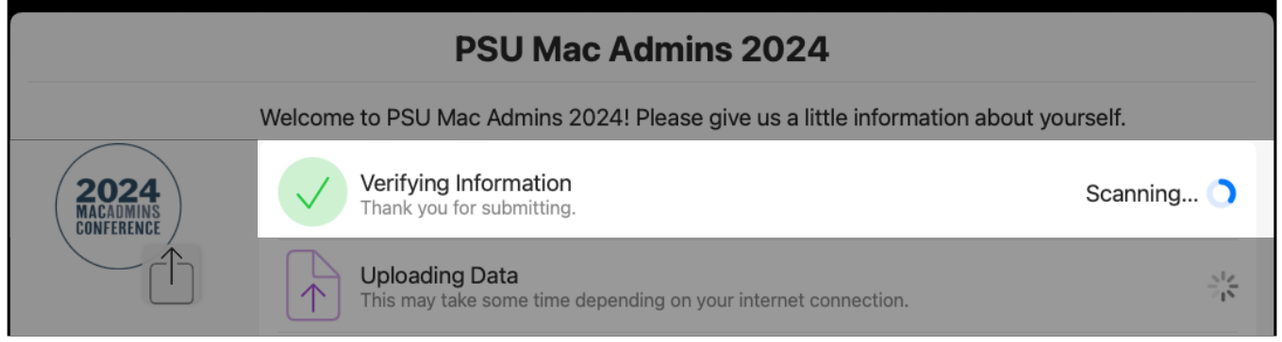
Add list items in shell or using a json

```
{
    "title": "PSU Mac Admins 2024",
    "message": "Welcome to PSU Mac Admins 2024! Please give us a little information about yourself.",
    "icon": "2024 MacAdmins Logo-Circle-Blue.png",
    "overlayicon": "SF=square.and.arrow.up",
    "ontop": true,
    "moveable": true,
    "height": 540,
    "width": 1080,
    "button1text": "Please Wait",
    "button1disabled": true,
    "infobox": "infobox.md",
    "progress": 100,
    "listitem" : [
        {"title" : "Verifying Information", "icon" : "SF=checkmark.circle.fill,color=green", "status" : "wait",
        {"title" : "Uploading Data", "icon" : "SF=arrow.up.doc,color=purple", "status" : "wait", "subtitle": "T
        {"title" : "Processing Response", "icon" : "SF=gearshape.2,color=blue", "status" : "wait", "subtitle":
    ]
}
```

Dialog Lists

Updating Individual List Items

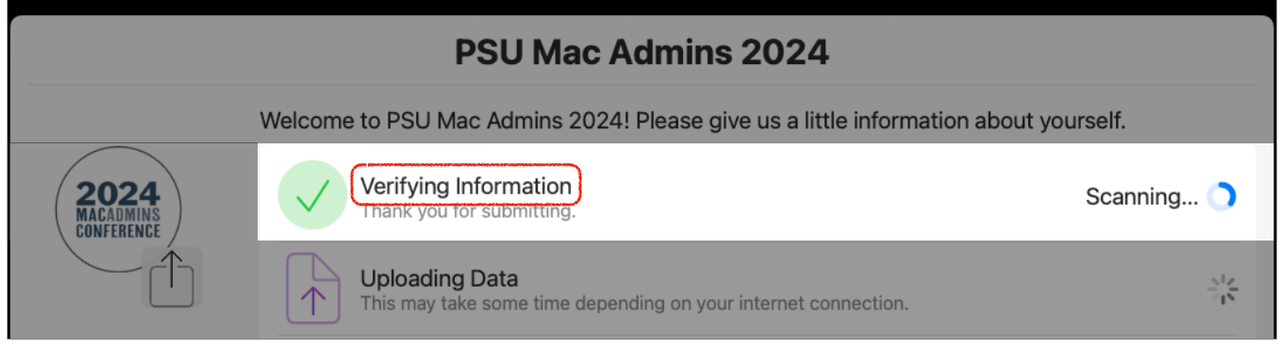
```
displayName="Verifying Information"
dialog_command "listitem: title: ${displayName}, statustext : Scanning..."
```



Dialog Lists

Updating Individual List Items

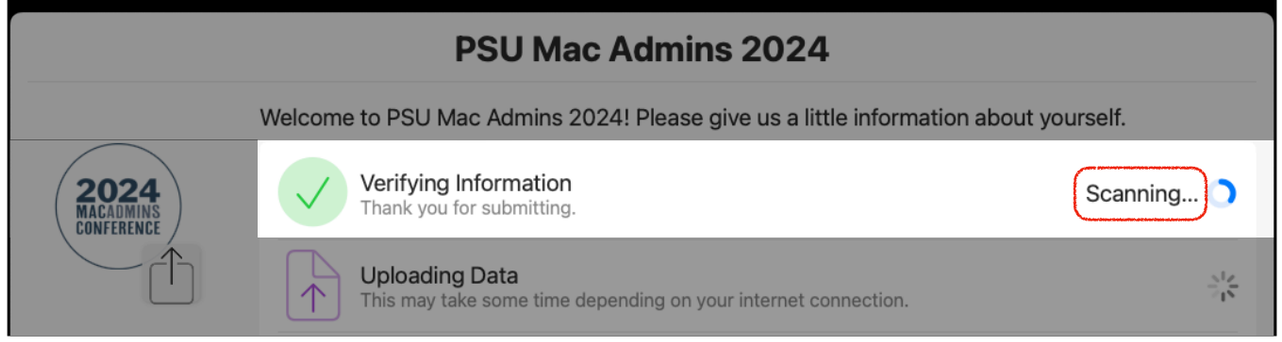
```
displayName="Verifying Information"
dialog_command "listitem: title: ${displayName}, statustext : Scanning..."
```



Dialog Lists

Updating Individual List Items

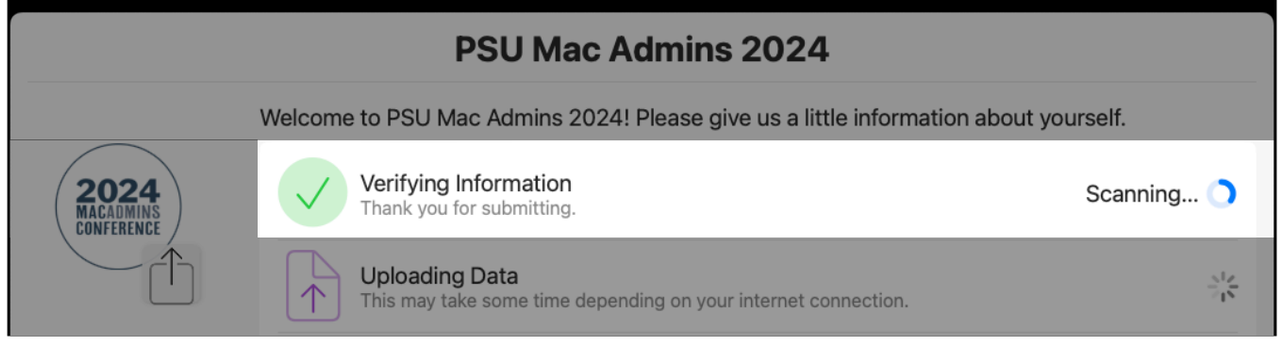
```
displayName="Verifying Information"
dialog_command "listitem: title: ${displayName}, statustext : Scanning...'
```



Dialog Lists

Updating Individual List Items

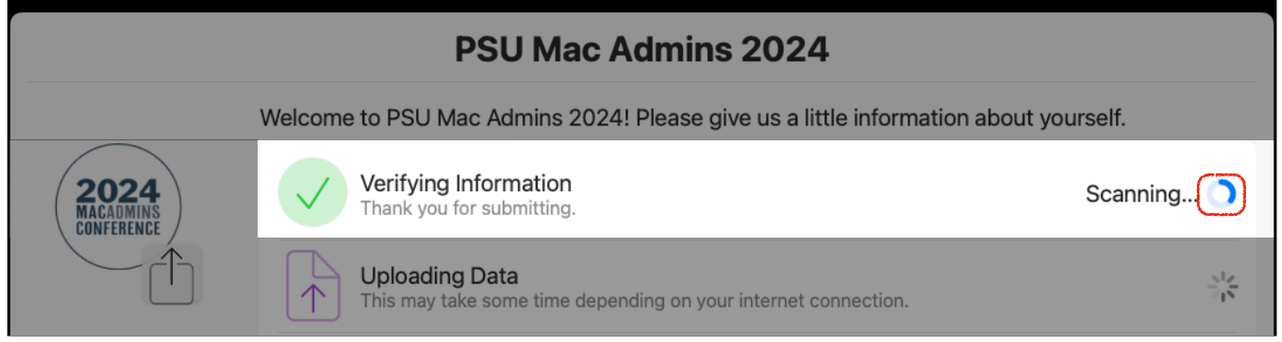
```
progress=40
dialog_command "listitem: title: ${displayName}, progress: $progress"
```



Dialog Lists

Updating Individual List Items

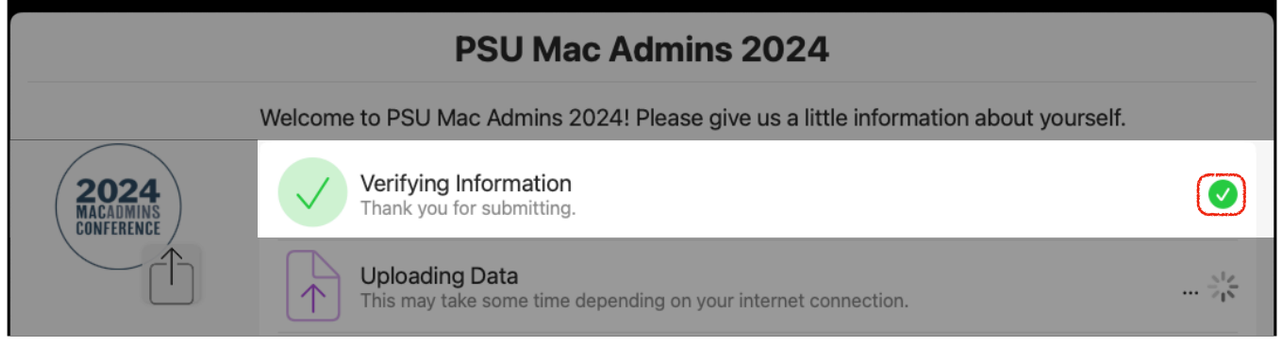
```
progress=40
dialog_command "listitem: title: ${displayName}, progress: $progress"
```



Dialog Lists

Updating Individual List Items

```
status="success"
dialog_command "listitem: title: ${displayName}, status: ${status}"
```



Q&A
swiftDialog

Better Together

Better Together

Step One

- Initiate your Dialog window

We can make an awesome and easy self-service workflow with just 4 steps in a script

Step one is to draw a dialog window

Better Together

Step One

- **Initiate your Dialog window**

```
/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &
```

Better Together

Step One

- **Initiate your Dialog window**

```
/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \ <-- Use custom Command File
    --moveable --ontop --mini &
```

Better Together

Step One

- **Initiate your Dialog window**

```
/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \ <-- Use custom Command File
    --moveable --ontop --mini & <-- Use "&" to send to the background
```

Better Together

Step Two

Step 2 is to run the installomator label

We will tell installomator that we want it to utilize the built in swiftDialog integration by passing an argument telling Installomator where our dialog command file is

Better Together

Step Two

- **Run your Installomator label**

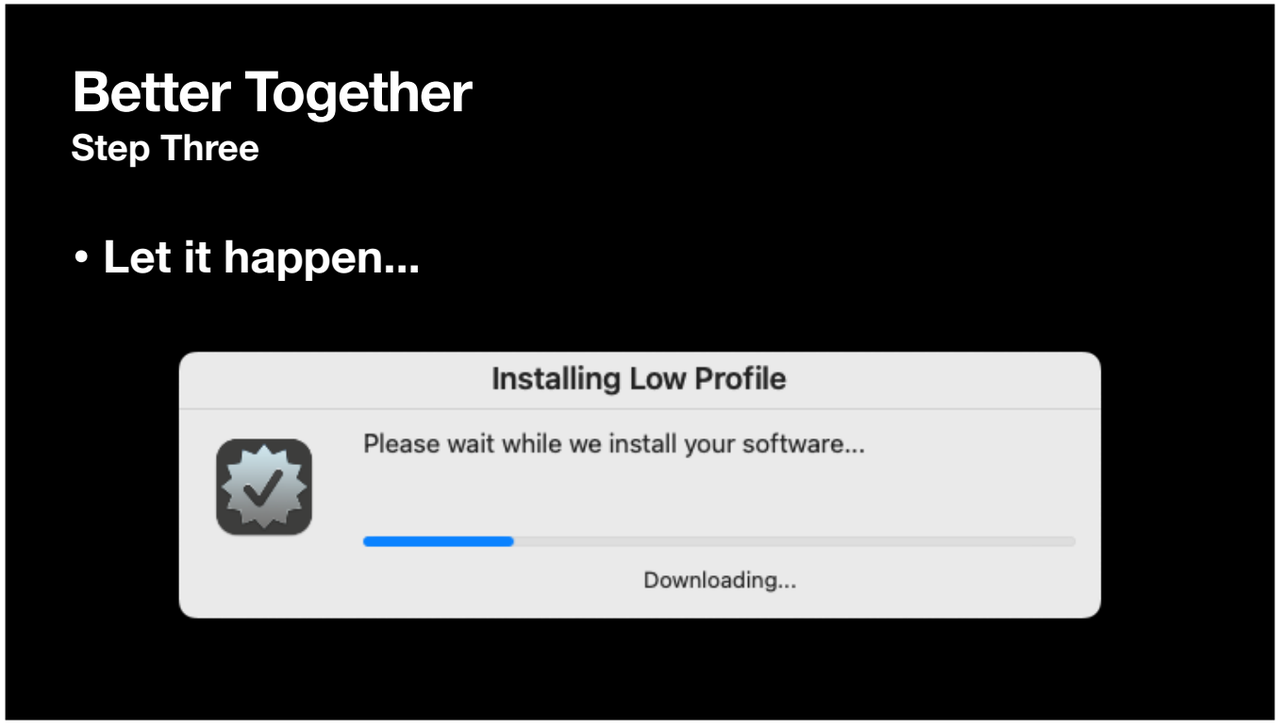
```
/usr/local/Installomator/Installomator.sh \
   lowprofile \
   DIALOG_CMD_FILE="$dialogCommandFile"
```

Better Together

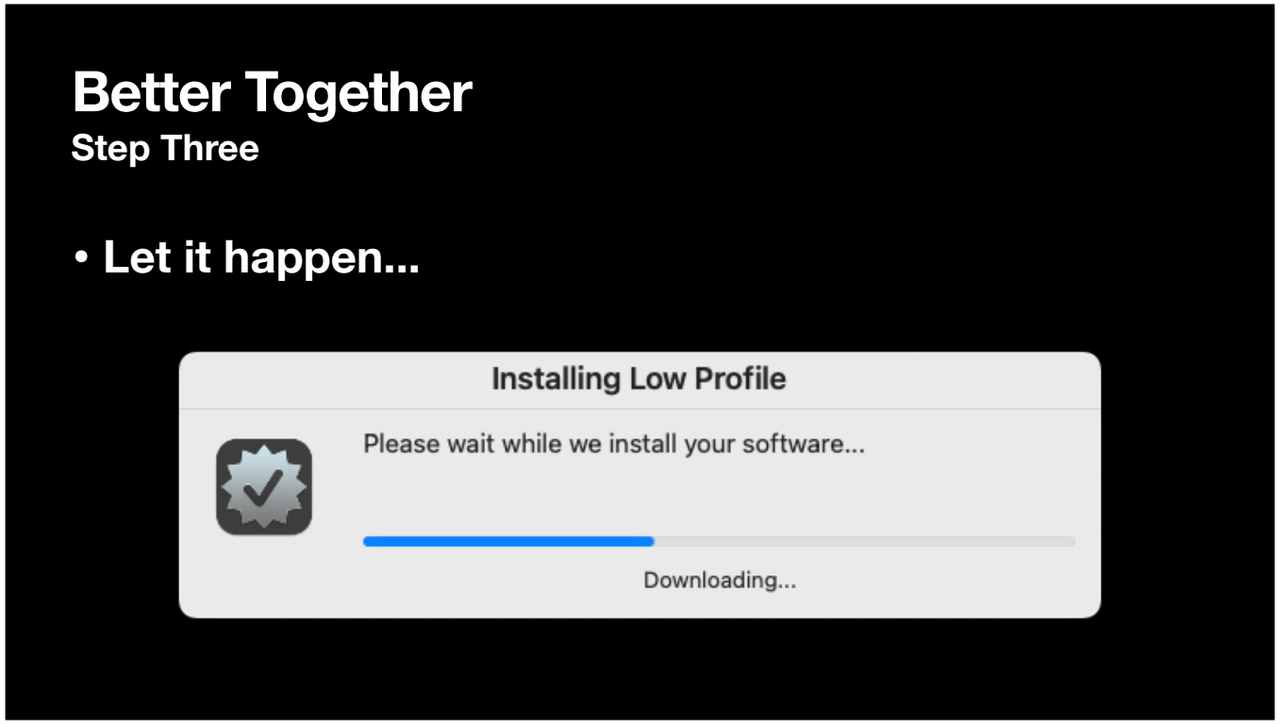
Step Two

- Run your Installomator label
  - Tell Installomator where your Command File is

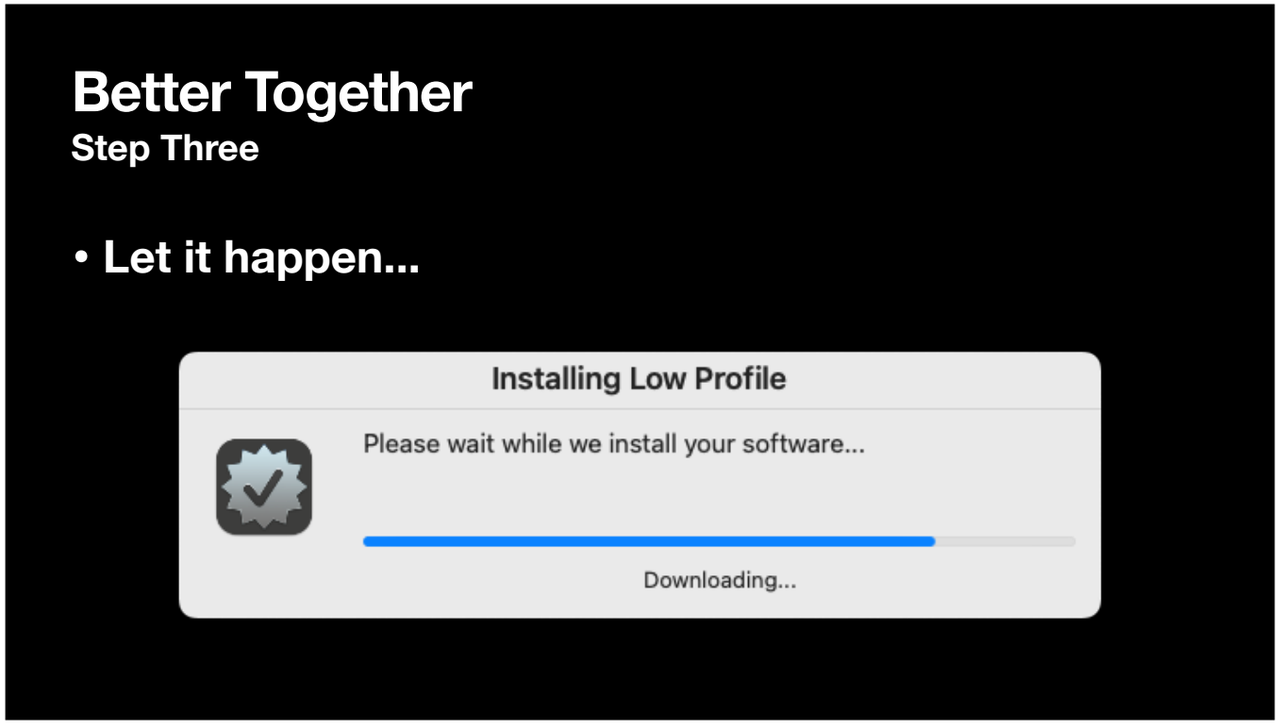
```
/usr/local/Installomator/Installomator.sh \
   lowprofile \
   DIALOG_CMD_FILE="$dialogCommandFile"
```



Step three we let it happen. This isn't even really a step.



Step three we let it happen. This isn't even really a step.

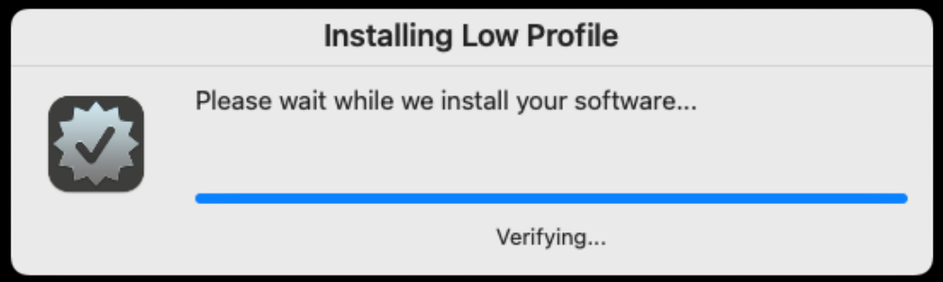


Step three we let it happen. This isn't even really a step.

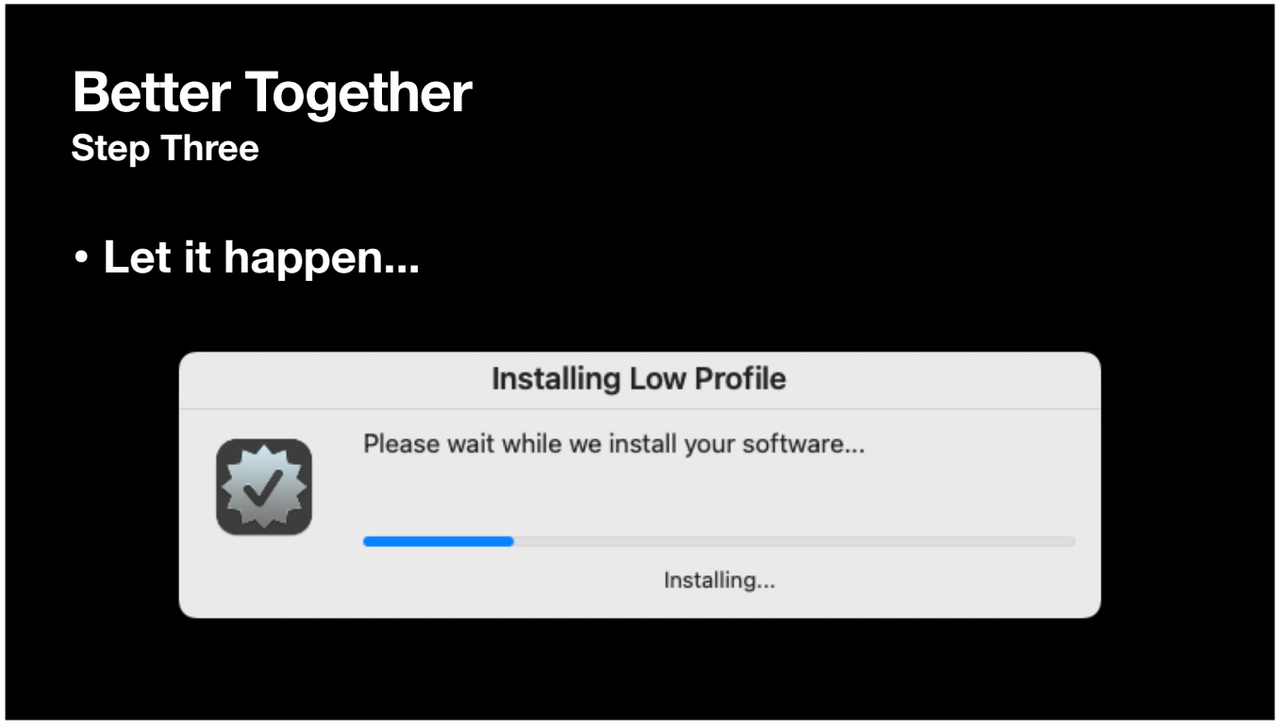
Better Together

Step Three

- Let it happen...



Step three we let it happen. This isn't even really a step.

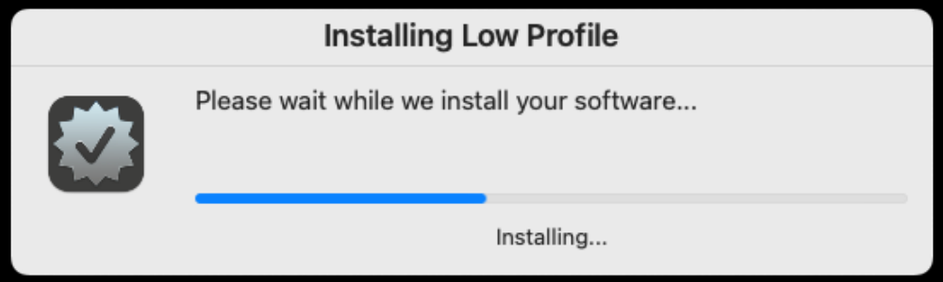


Step three we let it happen. This isn't even really a step.

Better Together

Step Three

- Let it happen...

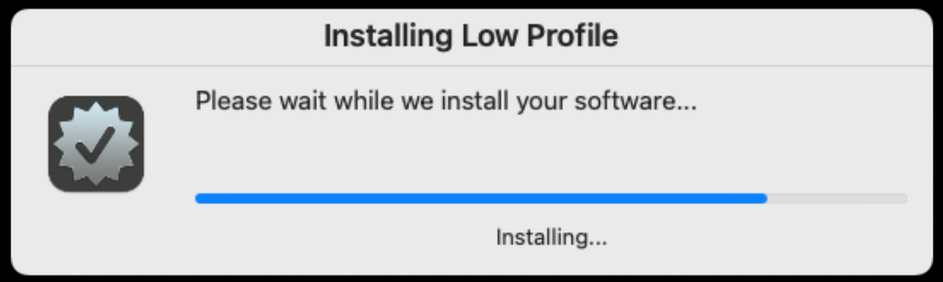


Step three we let it happen. This isn't even really a step.

Better Together

Step Three

- Let it happen...

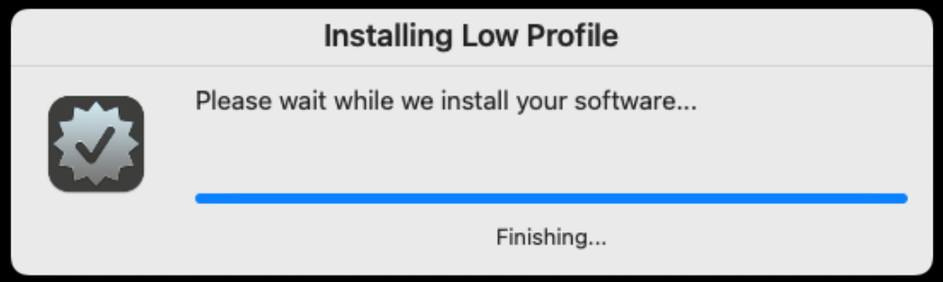


Step three we let it happen. This isn't even really a step.

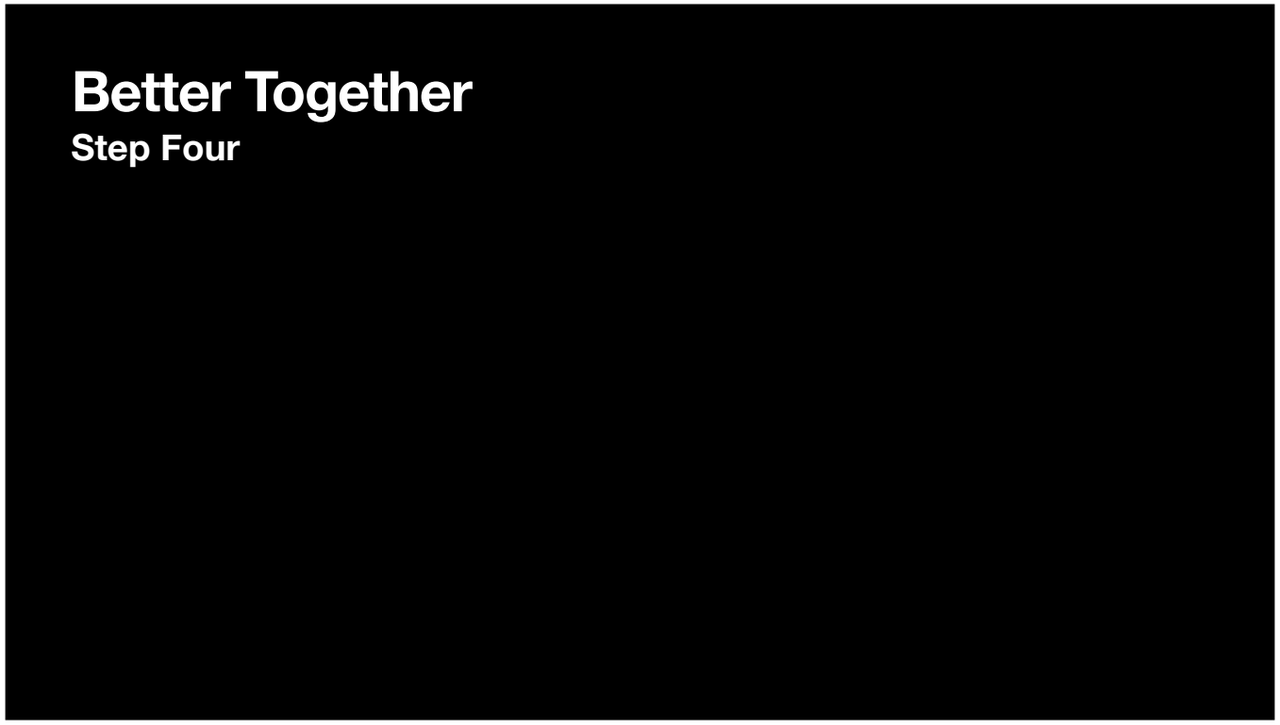
Better Together

Step Three

- Let it happen... (Done)



Step three we let it happen. This isn't even really a step.



Installomator won't close our dialog window for us, so we do that ourselves

Better Together

Step Four

- **Quit your dialog**

```
dialog_command "quit:"
```

Better Together

Full Script

```
#!/bin/bash
# set -x

# FourStepSelfService.sh by: Trevor Sysock
# 2024-06-05

dialogCommandFile=$(mktemp /var/tmp/Example.XXXXX)

function dialog_command(){
    # $1 is the command we want to send to our running dialog window
    echo "${1}" >> "$dialogCommandFile"
    sleep .1
}

/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
```

Thats it, thats the four steps we need for a fully functional self service script.

Better Together

Setup Command File Stuff

```
#!/bin/zsh --no-rcs
# shellcheck shell=bash
# set -x

# Combined.sh by: Trevor Sysock
# 2024-06-05

dialogCommandFile=$(mktemp /var/tmp/Example.XXXXX)

function dialog_command(){
    # $1 is the command we want to send to our running dialog window
    echo "${1}" >> "$dialogCommandFile"
    sleep .1
}

/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
```

Define our command file stuff

Better Together

Call Dialog Window

```
function dialog_command(){
    # $1 is the command we want to send to our running dialog window
    echo "${1}" >> "$dialogCommandFile"
    sleep .1
}

/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```

Call our dialog window

Better Together

Call Dialog Window

```
function dialog_command(){
    # $1 is the command we want to send to our running dialog window
    echo "${1}" >> "$dialogCommandFile"
    sleep .1
}

/usr/local/bin/dialog \
    --title "Installing Low Profile" \
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```

Better Together

Call Installomator with Label and Command File

```
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```

Call Installomator with the label and options we want (including the custom command file)

Better Together

Call Installomator with Label and Command File

```
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```

Better Together

Call Installomator with Label and Command File

```
    --message "Please wait while we install your software..." \
    --icon "/Applications/Low Profile.app" \
    --button1disabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```

script path

label

options (command file)

Better Together

Close the Dialog

```
    --buttondisabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```

Close our dialog window

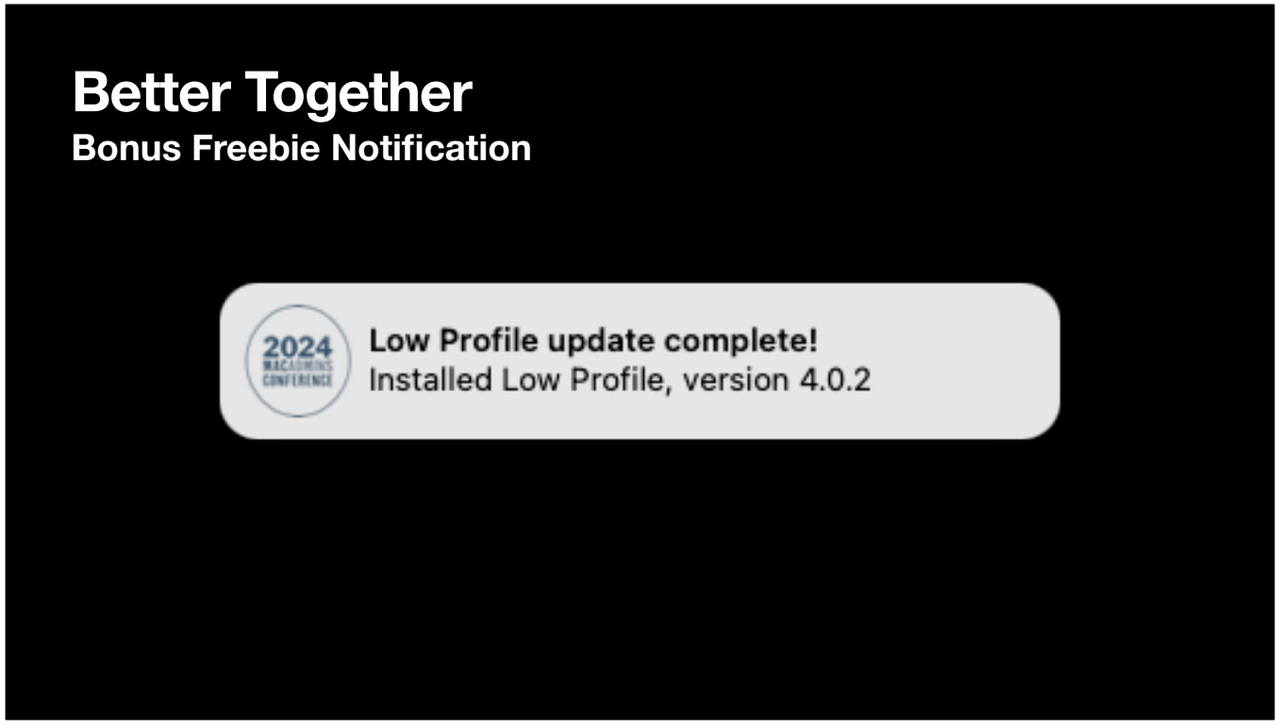
Better Together

Close the Dialog

```
    --buttondisabled \
    --progress \
    --commandfile "$dialogCommandFile" \
    --moveable --ontop --mini &

/usr/local/Installomator/Installomator.sh lowprofile DIALOG_CMD_FILE="$dialogCommandFile"

dialog_command "quit:"
```



As a bonus you're going to get a macOS Notification center event to let your user know that the application was successful or failed

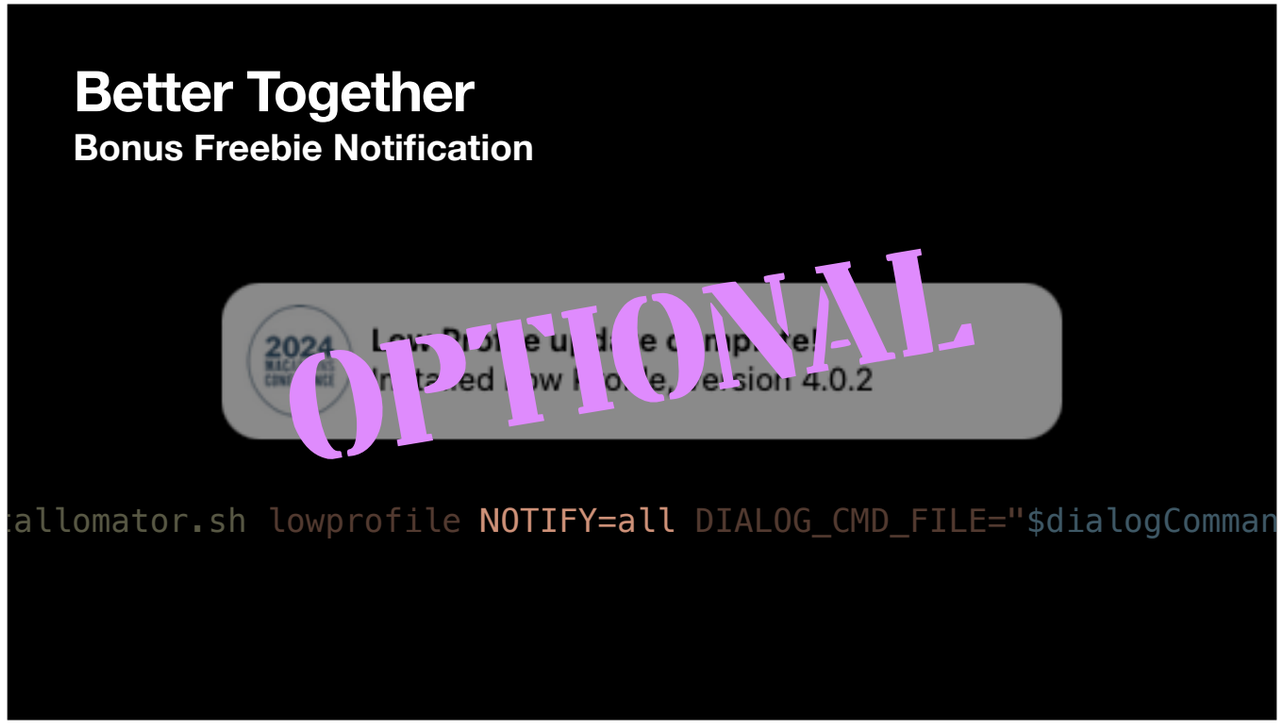
Better Together

Bonus Freebie Notification


```
/usr/local/Installomator/Installomator.sh lowprofile NOTIFY=silent DIALOG_CMD_FILE="$dialogCommandFile"
```

This is on by default, but we can skip it by passing an option to Installomator

We can silence notifications



We can be extra verbose and notify of each individual step along the way, I would't use this in production

Or we can notify upon successful installs.

Installomator does not notify upon failures, so you'll need to program that into your script based on the Installomator exit code if you want it

Better Together

Bonus Freebie Notification

OPTIONAL

```
allomator.sh lowprofile NOTIFY=success DIALOG_CMD_FILE="$dialogCo
```

default

That was too easy...

That was way too easy, and we have time left, so lets do something even better!

That was too easy...

We can do more...

Deploy Multiple Apps

Lets go through an example script which deploys multiple apps

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

This script is available on my GitHub, i'll have a QR code and a link at the end of the presentation.

This is not meant to be a fully developed production quality script, this could be the foundation of one but is meant for use as an example of the technologies and how to use them together

This script is only 49 lines of code when all of the comments and empty lines are stripped out

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
#!/bin/bash
#set -x

# Installomator-Dialog-List-Integration.sh by: Trevor
Sysock
# 2024-07-02

# This script will:
#   - Take a CSV formatted array of items to install
#   - Create a swiftDialog list view with all items
#   - Run Installomator to install all items
#   - Create a macOS Notification Center Event upon
completion
```

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# An array containing any options we want for our
Dialog list view window
dialogOptions=(
    --title "PSU Fancy Example"
    --message "Thanks for sticking around this long..."
    --icon "./Icons/PSU.png"
    --height 420
    --ontop --moveable
)

# An array containing the apps we want to install
# Comma separated values: "label,DisplayName,Path to
Icon"
# Each line should be encased in "quotes" so that it is
```

When I write scripts, I like to keep things in arrays.

This is an array that contains all of the basic options I want to pass to swiftDialog to make the window look how i want to look

You'll notice that my message and title text are surrounded by quotes, this is to ensure that the string is ONE array element, and not split at the spaces

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# An array containing the apps we want to install
# Comma separated values: "label,DisplayName,Path to
Icon"
# Each line should be encased in "quotes" so that it is
one single array entry
apps=(
    "firefoxpkg,Firefox,./Icons/Firefox.png"
    "nudgesuite,Nudge,./Icons/Nudge.png"
    "lowprofile,Low Profile,./Icons/Low Profile.png"
    "mist,Mist,./Icons/Mist.png"
    "renew,Renew,./Icons/Renew.png"
)

# An array containing the options we want to use for
```

This is another array containing details on which specific apps I want to install. We're going to install 5 things.

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# An array containing the apps we want to install
# Comma separated values: "label,DisplayName,Path to
Icon"
# Each line should be encased in "quotes" so that it is
apps=(
    "firefoxpkg,Firefox,./Icons/Firefox.png"
    "nudgesuite,Nudge,./Icons/Nudge.png"
    "lowprofile,Low Profile,./Icons/Low Profile.png"
    "mist,Mist,./Icons/Mist.png"
    "renew,Renew,./Icons/Renew.png"
)

# An array containing the options we want to use for
```

"firefoxpkg,Firefox,./Icons/Firefox.png"

You'll notice that this entire line is surrounded by quotes. I want this whole line to be one single array element. Then using a trick I learned from Adam Codega, we'll use the cut command to split this line up by it's comma separation later on.

I need each of these elements to be related, so i make them one array entry which I can split.

The Icon file paths are relative, which means the Icons folder needs to be in the same folder the script is running

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# An array containing the apps we want to install
# Comma separated values: Label,Name,PlayName,Path to
Icon"
# Each line should be encased in "quotes" so that it is
apps=(
    "firefoxpkg,Firefox,./Icons/Firefox.png"
    "nudgesuite,Nudge,./Icons/Nudge.png"
    "lowprofile,Low Profile,./Icons/Low Profile.png"
    "mist,Mist,./Icons/Mist.png"
    "renew,Renew,./Icons/Renew.png"
)

# An array containing the options we want to use for
```

cut -d ','

1 2 3

"firefoxpkg,Firefox,./Icons/Firefox.png"

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# An array containing the options we want to use for
Installomator
installomatorOptions=(
    NOTIFY=silent
    BLOCKING_PROCESS_ACTION=ignore
    DEBUG=1
)

# An array containing the options we want to use for
our final notification
notificationOptions=(
    --title "That's all, folks..."
    --message "Thank you for coming!"
)
```

Here i have another array containing any options I want to pass to Installomator.

Silenced notifications. Ignore if the application is already running.

DEBUG downloads but doesn't install

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# An array containing the options we want to use for
our final notification
notificationOptions=(
    --title "That's all, folks..."
    --message "Thank you for coming!"
)

####################################
# DO NOT EDIT BELOW FOR NORMAL USE #
####################################
# Syntax:
#   Variables and arrays are "camel case":
$thisIsAVariable
#   Functions are "snake case": this_is_a_function
```

One final configuration array for my macOS notification I want to send at the end of this process

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# Paths to our tools
dialogPath="/usr/local/bin/dialog"
installomatorPath="/usr/local/Installomator/
Installomator.sh"

# Create a tmp Dialog command file, and add it to our
array of options
dialogCommandFile=$(mktemp /var/tmp/
harrisonFord.XXXXXX)
dialogOptions+=(--commandfile "${dialogCommandFile}")
#################
#   Functions   #
#################
# execute a dialog command
```

Next I've got the path to my tools defined as variables, I just prefer to do things this way.

I'm also creating my command file, and appending the option to use that command file to my swiftDialog options array.

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
for app in "${apps[@]}"; do
    # Get the display name for this item
    displayName=$(echo "${app}" | cut -d ',' -f2)
    # Get the Icon path for this item
    displayIcon=$(echo "${app}" | cut -d ',' -f3)
    # Add to our final Dialog command
    dialogListView+=( --listitem "$
{displayName}",icon="${displayIcon}")
done

# Create our initial dialog utilizing the arrays of
options we have created
# Send it to the background with &
"$dialogPath" "${dialogOptions[@]}" "$
```

Now we get a little more scripty. This block of code goes one by one through the array of apps I want to install, and creates a new array with the proper formatting to get swiftDialog to show my Display Names and Icons on each dialog list item

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# Create our initial dialog utilizing the arrays of
options we have created
# Send it to the background with &
"$dialogPath" "${dialogOptions[@]}" "$
{dialogListView[@]}" &
sleep 1

# For every item in our apps array, run the
installomator label
# Include the swiftDialog options for integration
for app in "${apps[@]}"; do
    # Get our Label and DisplayName from our CSV
    label=$(echo "${app}" | cut -d ',' -f1)
    displayName=$(echo "${app}" | cut -d ',' -f2)
```

Since all of my options are neatly contained within arrays, my actual command to create the dialog list view is very simple.

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
for app in "${apps[@]}"; do
    # Get our Label and DisplayName from our CSV
    label=$(echo "${app}" | cut -d ',' -f1)
    displayName=$(echo "${app}" | cut -d ',' -f2)

    # Make this list item pending and sleep 1 because
it feels good
    dialog_command "listitem: ${displayName}: wait"
    sleep 1

    # Call installomator with all of our options
    "$installomatorPath" "${label}" \
        "${installomatorOptions[@]}" \
        DIALOG_CMD_FILE="${dialogCommandFile}" \
```

Now once again I iterate through the apps array, but this time I'm running Installomator with each loop and remembering to include the dialog command file and the name of each item i'm updating and passing them to Installomator

Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
    label=$(echo "${app}" | cut -d ',' -f1)
    displayName=$(echo "${app}" | cut -d ',' -f2)

    # Make this list item pending and sleep 1 because
it feels good
    dialog_command "listitem: ${displayName}: wait"
    sleep 1

    # Call installomator with all of our options
    "$installomatorPath" "${label}" \
        "${installomatorOptions[@]}" \
        DIALOG_CMD_FILE="${dialogCommandFile}" \
        DIALOG_LIST_ITEM_NAME=\'"${displayName}"\'
done
```

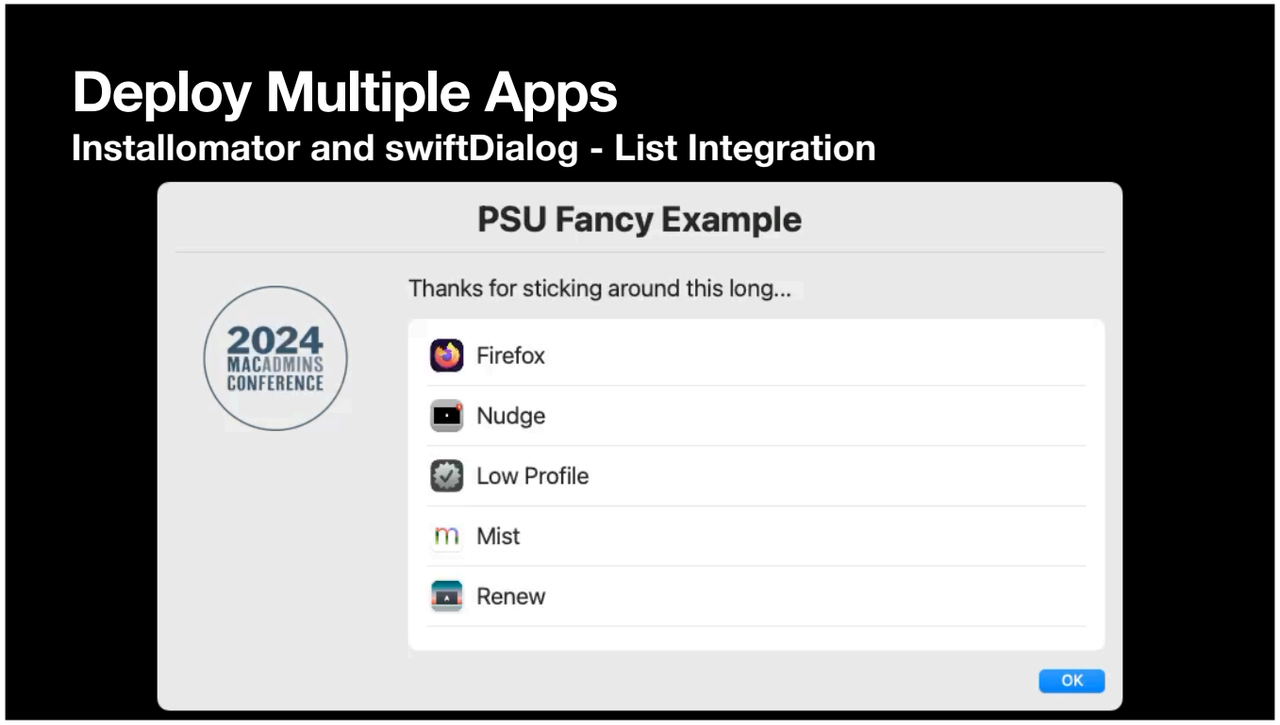
Deploy Multiple Apps

Installomator and swiftDialog - List Integration

```
# Give a notification that we're done
"$dialogPath" --notification "$
{notificationOptions[@]}"

# That's it!!!
dialog_command "quit:"
rm "${dialogCommandFile}"
```

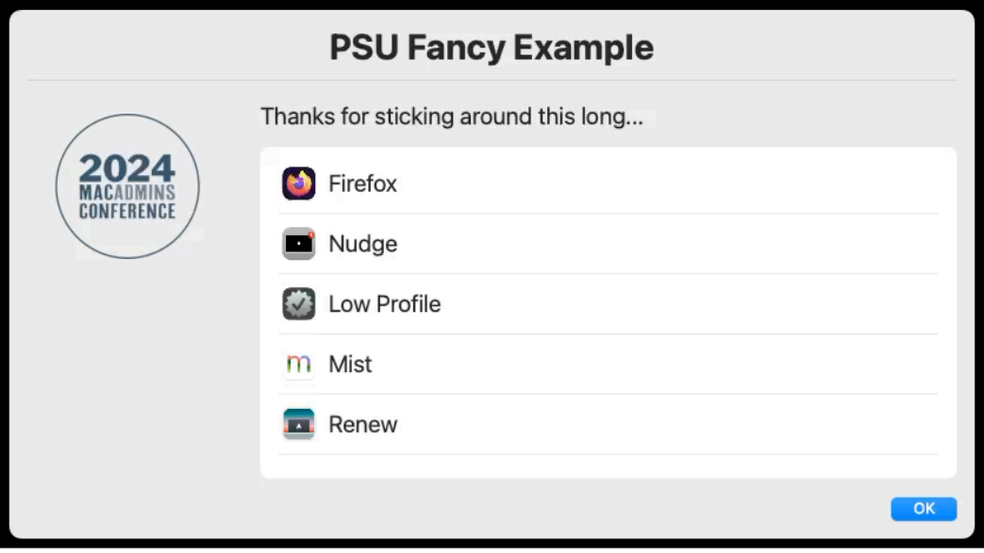
And thats it! I'm going to make a notification center event, close my dialog window, and delete my command file. we are done



This is a real-time video of the script running, but this is running labels on DEBUG mode which means it's downloading but not actually installing

Deploy Multiple Apps

Installomator and swiftDialog - List Integration



Deploy Multiple Apps

Installomator and swiftDialog - List Integration

Baseline by Second Son Consulting

An MDM agnostic workstation setup tool

- **Use a configuration file as an instruction set**
- **Run Installomator labels, pkgs, or scripts**
- **Configure all swiftDialog options from the configuration**

Sources and References

- https://github.com/Installomator/Installomator
- https://github.com/swiftDialog/swiftDialog

Sources and References

- https://github.com/SecondSonConsulting
  - Baseline (#baseline on Slack)
  - Renew (#renew on Slack)
  - swiftDialogExamples
- https://github.com/BigMacAdmin/macOS-Stuff
  - Various utilities and fun stuff

Sources and References

- Projects which use swiftDialog
  - Setup Your Mac
  - Erase Install
  - App Auto Patch
  - Patchomator
  - More and more and more...

For inspiration or just to steal the code

There are more, and i'm not trying to play favorites!

App Auto Patch and Patchomator both use Installomator along with swiftDialog

Sources and References

https://github.com/BigMacAdmin/PSU-MacAdmins-2024

BeRate Me!

2024
MACADMINS
CONFERENCE

https://bit.ly/psumac-2024-41

~~Be~~Rate Me!

2024
MACADMINS
CONFERENCE

https://bit.ly/psumac-2024-41