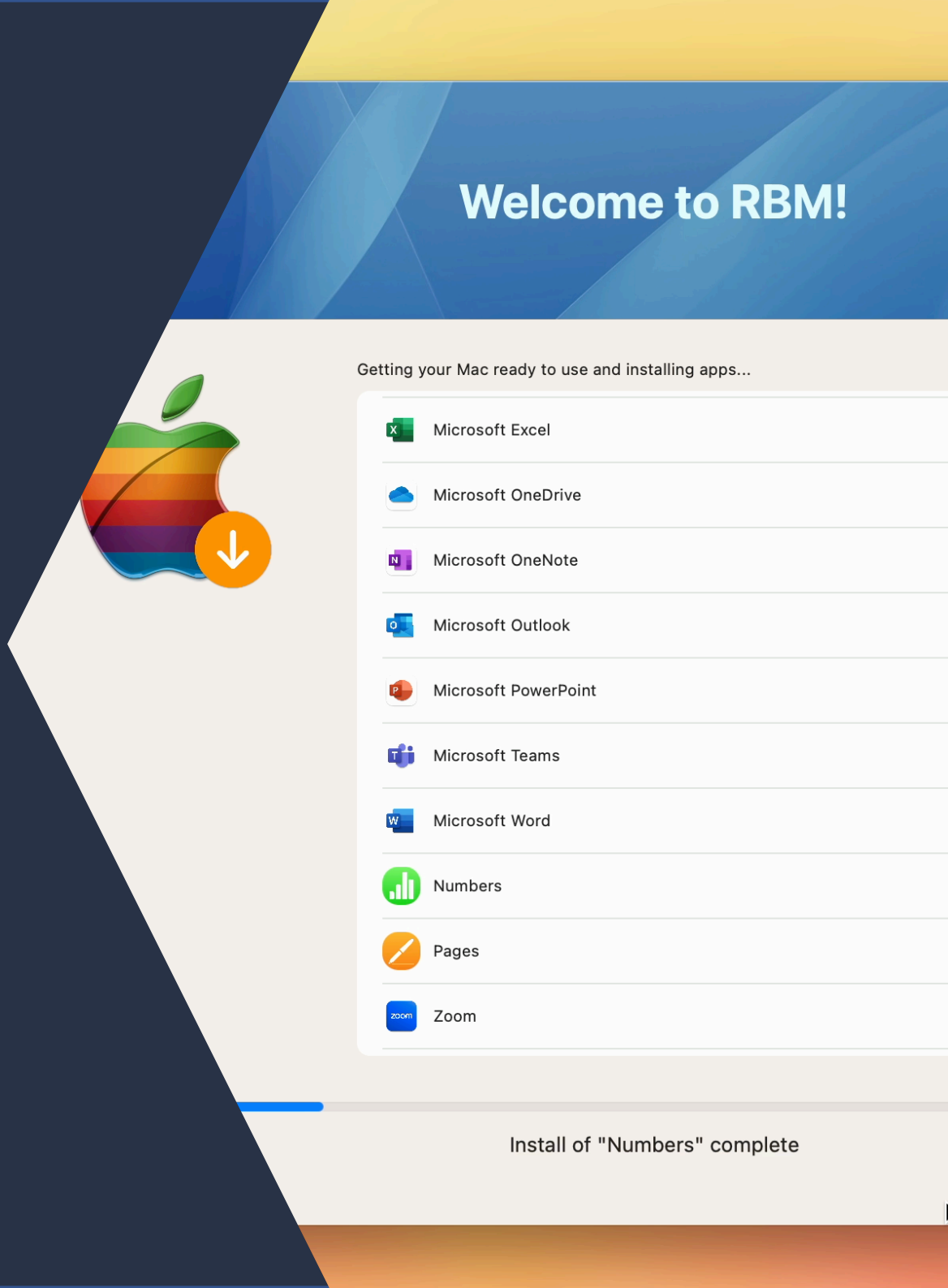
academia
the technology group

Swift Dialog Build

By Richard Brown-Martin

Friday, 24 March 2023



Swift Dialog Build

About this Document

This document outlines how to use the Swift Dialog Script by Richard Brown-Martin to quickly configure a graphical front end for Mac build processes.

Why Bother?

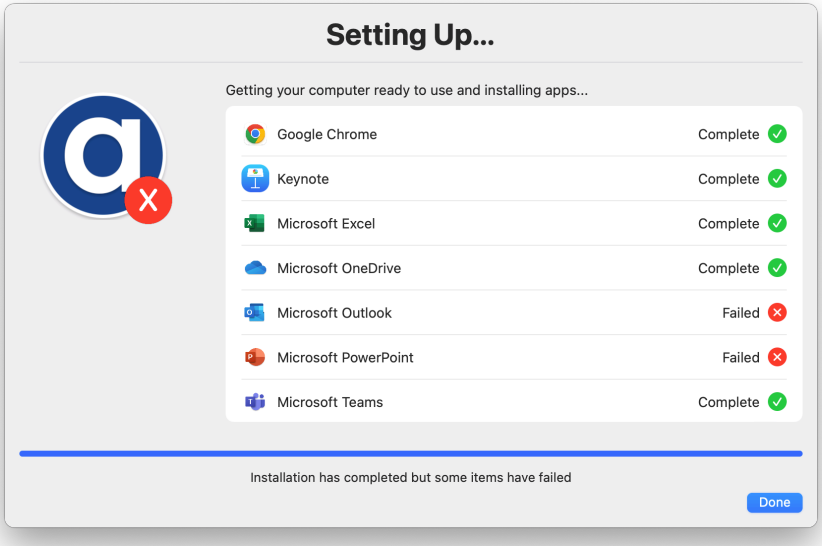
If you're the sort of person who'll be reading this document, you'll already be thinking
Why do we need another graphical front end for macOS builds? We have DEP Notify and its relatives NoMAD Notify and Jamf Connect Notify?
There are pretty good off-the-shelf scripts for getting those tools to run a bunch of Jamf policies in a certain order and give a nice interface for that. Why bother changing?

First up…

DEP Notify hasn't been graphically updated since macOS 10.10 Yosemite which still appears in its default banner graphic. That was released in 2014 and today in 2023, even hardware released in 2016 has made it to the scrap pile. In the interim, Apple have completely refreshed the appearance of macOS, implemented a whole new API for graphics (SwiftUI) and switched to a different CPU architecture. In that light, DEPNotify feels a bit old and rubbish.

Moreover…

When DEP Notify launched, the policy was king in the world of deployment. But Jamf policies have started to lose their place as premier method of app installation. While deploying from the Mac App Store using VPP has limitations, many Jamf admins got their start in primarily iOS focused environments and prefer the more automatic approach to App deployment and licensing afforded by that method - Microsoft Office being available via VPP also makes it an attractive approach. However, the real shake-up has come with Jamf's own App Catalog, introduced in March 2022 as part of Jamf Pro 10.37.
While the Jamf App Catalog has some major drawbacks and limitations, it has clear benefits for simplifying workflows and patch management and we are assured it is the future. So while the future of policies looks to move from packages to Installomator, workflows that are designed to work with all install methods and not just policies are going to become more important. That is what this script attempts to solve.



Finally…

There's a bit of a learning curve with build scripts. The curve goes from creating packages in a drag-and-drop interface or setting up a few smart groups to suddenly needing to understand long and complex bash scripts and set a whole bunch of settings. Even worse, if you have a few different "builds" to deploy - like the IT department, the Art department, that one lad in finance who insists he needs a Mac even though the rest of that department are Windows only - a plethora of clone scripts are required each with their own tweaks.

This might be somewhat inevitable, but the complexity can put some workflows out of reach of some administrators. There's always room to make things easier to work with.

Features and Benefits

- **Asynchronous Installation Tracking**
  Items are marked as complete as they install as soon as the script detects a specific file has appeared on the client machine (eg: /Applications/Keynote.app). It doesn't matter what order or method was used to install them
- **Jamf Policy Triggering**
  Custom Policy Triggers can be run sequentially from the script. As intended, each policy will install a separate item in a specific order - much like DEPNotify workflows.
- **Modern Configurable Interface**
  The script makes it simple to customise the look of the banner and icon displayed as well. It looks modern and adding app icons is simple.
- **Configuration Profile**
  The script can either be configured by changing the variables and arrays directly or by using a preference file - typically with a configuration profile. This makes it simple to deploy the same script to multiple Macs in an estate while installing different content simply by deploying a different profile.
- **Works with DEP and UIE workflows**
  Because the script runs GUI commands asynchronously to Jamf Policy commands, it is highly suitable for DEP based workflows. The lightweight supporting items (approximately 6MB) can be installed at enrolment allowing the enrolment policy to start before users reach the login window. Upon logging in, they are presented with the current status of their build progress. If any installations have already occurred, they will be able to see the state of the installation - rather than some workflows that wait for the DEPNotify window to launch before running any installations by policy.
- **Custom Schema**
  To allow easy setup of Configuration Profiles, a JSON Custom schema is available
- **Easy to Deploy**
  Deploying multiple builds follows these steps
  1. Setting up scoping for VPP and Jamf Catalog apps
  2. Creating policies with custom triggers for each item to be installed by policy.
  3. Creating and scoping a configuration profile with a list of what to display and policies to call for each build
  4. Optionally packaging graphical resources for logos and banners
  5. Creating a single enrolment policy to deploy the script

Known issues and limitations

- Swift Dialog will not run until a user has logged in, so cannot display build progress immediately after setup assistant and may require some user intervention
- No built-in functionality for naming a Mac, assigning to a user or manually picking a build is included making the workflow less useful for User-Initiated Workflows.
- In this version there is no option to run specific Jamf policy triggers without them being listed on the GUI - but there are numerous alternate methods for this
- If a configuration profile is used, all hard coded options in the script are overridden. If required settings are not configured, the script will not execute correctly.
- Currently no feedback is provided for the state of items being installed using policy other than "Installing" when the policy is first started.
- To determine if an item has installed a path must be specified and a file written. For an app or printer this will be the app's path or printer's PPD, but for a policy that does not write a standard file, the policy will need to write some file or other to monitor for completion.

Additional issues

- In Jamf Pro 10.43, if apps are installed using Jamf Catalog, they are not reinstalled after being deleted or the Mac is erased / re-enrolled. For testing, delete the Mac from Jamf Pro completely before re-enrolling

Version History and Release Notes

Version	Date	Notes
0.1	03/03/2023	Initial Release
0.2	17/03/2023	Improved window sizing when using banners
0.3	24/03/2023	Bug fixes to ensure non-policy items are not triggering policies or recorded as failed incrorrectly. Improvements to exiting Fixes to EAs

Deployment Overview

Component Overview

Items to add to Jamf Pro

Items marked with ✅ are required. Items marked with ⚠️ are typically required or are dependencies for other components. Other items are entirely optional.

Name	Description	Type	Required
Swift Dialog Build Script	Main script that performs build and updates GUI	Script	✅
Enrolment Policy	A policy to trigger the script when a Mac enrols to Jamf Pro. Optionally this can install Swift Dialog and graphical resources	Policy	✅
Swift Dialog pkg	The script can install Swift Dialog from a URL, but it may be prudent to install this with a policy or as an enrolment package to ensure components are ready	Package	⚠️
Graphical Resources	During the build process, a custom icon and banner image can be added to the dialog for branding. These can either be installed as local assets or a URL can be used to deploy web-hosted resources.	Package	
Build Configuration Profile	To customise items installed, a configuration profile can be deployed to configure which items are installed on each Mac. The profile can configure the appearance of the dialog as well as which items are installed	Configuration Profile	⚠️
Install Policies	For any items that should be installed with a Policy, a policy with a suitable custom trigger should be created. Each policy must	Policies	⚠️
Mac Apps / Jamf Catalog Apps	For any apps that should be installed by either Mac App Store (VPP) or Jamf Catalog, apps should be added and configured for automatic deployment	Mac Apps	⚠️
Scoping EA	If used with configuration profile(s), the Scoping EA reports which items should be scoped to the client machine	Extension Attribute	⚠️
Scoping Smart Groups	If using the Scoping EA, then each item installed can be scoped using a smart group to based on the results of the scoping EA	Smart Groups	⚠️
Build Status EA	Optional EA to report if each Mac was set up using this process and the outcome, Complete or Failed	Extension Attribute	
Build Name EA	Optional EA to report an arbitrary build name at build-time	Extension Attribute	

Items that may be added to client Macs

Name	Description	Required
Swift Dialog	The Swift Dialog binary must be installed to each Mac. This could be deployed as a pkg via Jamf or downloaded from the developer's GitHub and installed automatically by the script.	✅
Graphical Resources	Optionally, a custom icon and banner image can be added to the dialog for branding. These can either be installed as local assets to an accessible folder or a URL can be used to deploy web-hosted resources.	
Build Preferences	Optionally, settings for the Build can be stored on a client Mac as a Preference file. This would typically be a Managed Preference deployed by a Configuration Profile, but could be installed manually. This defines the look of the dialog and items installed	
Build Flag Preference	Optionally, a preference file can be written to the client Mac to record if the process completed successfully or failed and a date when the Mac completed the process. It can also record a "build name" to later identify which process was used.	

Preparation

Before starting, some fundamental decisions should be made.

- Is there one build across the entire estate or several different setups?
  - If all / most Macs are likely to be set up the same, then this configuration could be embedded in the script.
  - If there are a variety of different setups, then a configuration profile may be a better option for deploying configurations
- Are some apps deployed to all Macs - this may affect scoping options
  - If all Mac App Store apps or apps deployed with the Jamf App Catalog are required on every Mac, then the Scoping EA is not needed. Apps should be scoped manually
  - If some apps from the Mac App Store or Jamf App Catalog are only installed on some Macs, then the Scoping EA may be required
- Should the dialog be customised with graphics for icons and banners? If so, can they be deployed from a web server or should they be installed locally?
- Which items to deploy by Mac App Store, Jamf App Catalog or Policy?
  - This is an increasingly complex question. Mac App Store is required for some apps (mostly apps from Apple) but some apps can be installed in mutliple ways. The Mac App Store provides some update flexibility while the Jamf App Catalog includes fully automated patching. Policies can be configured with custom packages if needed or apps can be deployed with a script to ensure the latest version is installed. If a custom package is needed then a policy is still the best option.
- Is there anything being installed / configured that shouldn't appear in the dialog?
  - While the dialog is a great way to show progress, some smaller apps and configuration might not need to be shown explicitly.

Script

The script should be uploaded to Jamf Pro and given a suitable name.
Preference Paths can be changed as required. The Pref Path must match the domain of the configuration profile(s) being deployed
If configuring manually, then the static variables in the script should be changed. See Script Configuration for more information.
If making other changes to the script, consult the script overview

Packages

The Swift Dialog Build itself only requires packages for the Swift Dialog binary which can be downloaded from GitHub, and optionally for graphical elements, see banner image for details. Icons for each item installed could also be included in the package or deployed from the internet as a URL.
Swift Dialog is signed and notarized, so suitable to deploy as an enrolment package. If graphical elements need to be deployed at enrolment, then the package would need to be signed.
Graphical elements installed locally can be installed to any globally accessible location such as /Users/Shared… or /Library/Management… or even /tmp

Configuration Profiles

If deploying multiple builds, then a configuration profile may be the best option to design and deploy variations. This is made simpler by using the custom schema to design the install process. The Configuration Profiles can use any domain, but this must match the PrefFile static variable defined in the Script.

📌 **Tip**
When designing the Configuration Profile, a suggested approach is to add all the possible items for every diferent build to one profile. This can then be deployed and tested to check that all policies and apps are installed correctly. After this, the configuration profile with every setting can be cloned and elements removed to create a profile for each build.

If multiple builds are needed, create a profile for each build. Then create a PreStage for each build along with a Smart Group for any Macs enrolled with that PreStage. Scope the profile to the appropriate Smart Group.

Install Policies

For each item to be displayed in the list that is installed by a policy, a policy needs to be created that:

1. Has a custom trigger
   Each policy is called by the build script using its custom trigger
2. Installs a file when complete
   A file is required by the script to determine if a policy succeeded or not
3. Appropriate Scoping
   This could be configured separately by the PreStage, or use the Scoping EA to create appropriate smart groups. As policies are called using a custom trigger, they could even be scoped to All Computers

File Install Requirment
For policies that install an app, printer or other files this, any file installed can be used as a reference.
However for scripted changes, binding to Active Directory, and other items that would not result in a file being written, the policy should be set to write a nominal completion file. This could be any flag file and even be written to /tmp so as to not persist. It could be added to the Files and Processes payload using *touch /tmp/somefile*
However, if it is possible to adapt a script or check that that the policy had the correct effect and to write a file only if it succeded, then that is preferred.

Enrolment Policy

The Enrolment policy should run the script. It should be scoped to any Mac that needs to use this process and unless variations of the script have been created per-build, the same policy can run for all Macs. Variation is more easily acheived using a Congiruation Profile.
As long as Preferences / Configuration Profiles are present or the script has the correct items hard-coded, that is technically all that is needed.

However, the Enrolment Policy could optionally install packages for Swift Dialog and any graphical resources.

Inventory Update is not needed in the policy as this is run as part of the script.

Frequency should generally not matter as the EnrolmentComplete trigger is only normally called once immediately after enrolment.

Scoping

The Scoping EA is used in conjuction with Configuration Profiles and returns a list of all items that are listed to be installed. From this, Smart Groups can be created
In designing scoping, the following reccomendations are suggested

- Any items that should be installed on every Mac should be scoped to every Mac rather than using the Scoping EA
- If deploying multiple builds and customising with a Configuration Profile, the Scoping EA can be used to create Smart Groups for each app / item
- If deploying items with a policy, the policy will be run using a custom trigger, so should generally be okay to scope to all computers
- Mac App Store Apps and Jamf Catalog Apps could use scoping derived from the PreStage rather than the Scoping EA.

The Scoping EA is intended to make it simpler to customise multiple builds with these principals

1. Each app typically need only be scoped to one group
2. Changing the profile for a build will automatically add / remove apps from the scope after Macs update their inventory
3. Monitoring which apps should be on each Mac becomes simplified

Installomator

Installomator is a popular tool and can be used in conjuction with this build process. However, newer versions of Installomator have integration with other popular Swift Dialog scripts. This can interfere with output from this build process and muck up the GUI.

It is therefore recommended to create a duplicate Installomator script to use for build policies and configure variables as outlined

Variable	Recommended Value	Note
DIALOG_CMD_FILE	“” (Blank)	Prevents Installomator changing the Swift Dialog interface
NOTIFY	silent	Reduces pop-ups and notifications
DEBUG	0	Production mode
REOPEN	no	Reduces likelihood of apps opening in the background
IGNORE_APP_STORE_APPS	yes	Reduces likelihood of app store apps being overwritten by accidentally scoped policies
BLOCKING_PROCESS_ACTION	kill	Reduces likelihood of pop-ups in the background

Hidden Items

The build script uses an array to control which items are shown in the dialog. However, additional configurations may be required that either do not need to be displayed or should not be shown.

For Mac App Store Apps and Jamf App Catalog apps, these can be scoped separately and deployed without displaying in the build dialog. This is useful for small utility apps and non-critical items.

For anything else deployed using a Jamf Pro policy, these can be handled in a variety of ways.

1. **Bundle with other items**
   To ensure items get installed during the build process, policies could have the same trigger as another item. Eg: While PowerPoint is shown in the list, the trigger installPowerPoint is called which runs more than one policy with that trigger, one being PowerPoint but a few other items are installed too.
2. **Generic list item**
   An item could be added to the list with a generic name and icon which triggers a variety of setup policies with the same custom trigger
3. **Deploy at Check-in**
   Items could be deployed separately from the build process at check-in. The cleanup section of the build process includes the option to run any policies that run at check-in
4. **Deploy as Enrolment Package**
   Packaged items or scripts embedded in packages could be deployed as an Enrolment Package if critical to the build process
5. **Deploy as Enrolment Policy**
   Additional policies could be called by the enrollmentComplete trigger - however if the build process is set to restart / logout / shutdown the Mac, it is possible this will prevent other policies from running. Therefore they should run before the Build Process and ideally be fast to prevent blocking the build, or part of the same enrolment policy.

End User Experience

For the user, once the Mac is enrolled with DEP, they will log in to their account and the Swift Dialog window will launch.

If the Swift Dialog window is set to fullscreen, users will see a blurred background and be unable to do anything until the Swift Dialog window closes.

As items install, users will see the list of items and symbols / descriptions to show progress

Description	Note	Symbol
Waiting	Items that are not installed by Jamf policies are listed as Waiting until they are detected and marked as completed	
Queued	Items installed by Jamf policy are initially marked as pending until the policy starts	
Installing	Items installed by Jamf policy are marked as installing once the corresponding policy trigger runs	
Failed	Items installed by Jamf policy are marked as failed if the policy trigger completes but no corresponding file is found, eg: policy installword completes but / Applications/Microsoft Word.app doesn't exist. Items installed by other methods are marked as failed only after a timeout	
Complete	Items installed by Jamf policy are marked as complete once the policy trigger completes and the corresponding file is found. Items installed by other methods are marked as complete as soon as their corresponding file is found. eg: /Applications/Pages.app	

The progress bar will also mark completion of each item until it is full. At which point, users will be notified if cleanup actions are being run - including running any other check-in policies and inventory updates before they see the final outcome.
At this point they will either be able to Quit, Restart, Shutdown or Logout. If the user ignores this for long enough, the action will complete automatically after a pause.

As the build progresses, the main icon will update to provide feedback

Phase	Note	Symbol
Install	Items in the list are downloading and installing	
Cleanup	Items in the list have completed and any other tasks are completing	
Finished (Complete)	Setup is complete, everything on the list installed	
Finished (Failed)	Setup is complete, but some items in the list did not complete	

Script Overview

The script follows the process outlined here to update the Swift Dialog UI and run Jamf policy triggers to complete the build

Startup

Upon initially running, the script records a start time, caffeinates the Mac to prevent display sleep during the build and unloads the Jamf Pro recurring checkin LaunchDaemon to prevent scheduled policies from running during the build. Start a Jamf recon to ensure

Get Prefs

If (managed) preferences are found at the specified path (/Library/Managed Preferences/com.rbm.swiftdialogbuild.plist as default), then these will be loaded **overriding** anything hard coded into the script itself.

Wait to Start

If Swift dialog is not yet installed or the Dock is not running, the script will loop - ensuring Swift Dialog is launched only after the user logs in. Jamf policies will continue to be run even while waiting for login.

Launch Dialog

Once Swift Dialog is installed and the user has logged in, the initial state of the dialog is created - including the icon, banner, text, appropriate window size, progress bar and initial list of items being installed. The basic list shows only the names of items with no progress symbol or icon.

Populate List

Updates the basic list by checking each item to be installed and determining if it is installed by policy (in which case it will be set as “pending”) or by another method (in which case it will be set as “waiting”). This causes the list to display icons and status information

Main Loop

The main loop continues until all items to be installed are set as complete using a counter variable. It will also exit if the timeout time is exceeded.
Steps in the main loop are

1. **Jamf Checker**
   This step checks if any Jamf policies should be run and if any triggered items are still running. Once one item finishes and is no longer running, it is marked as either complete or failed based and the next is executed until no custom Jamf policy triggers are left.
2. **Install Checker**
   This step checks if items not installed by policy triggers have installed and mark them as complete once they have installed.
3. **Refresh Dialog**
   Any items that have had their status changed (ie: items that have completed, policies that have just started or failed) are updated in the GUI
4. **Check Timeout**
   The running time is calculated in order for the main loop to abort if the timeout has been exceeded.

Cleanup

After the main loop has completed or aborted, the cleanup phase runs to
1. Update Dialog with “Performing Final Actions” message
2. Optionally perform any policies with a check-in trigger
3. Optionally write a preference file containing
   1. If all installed items completed or if any failed
   2. A name for the build
4. Optionally perform a final Inventory Update to Jamf Pro

Finalise

At this point, the dialog is updated allow the exit button to be clicked and if the main loop timed out, display a fail message or otherwise show a completion message.

Wait to End

The script loops for a defined period displaying the final install status of each item to allow a user to see. The time also ends once a user clicks to terminate the dialog window.

End Dialog

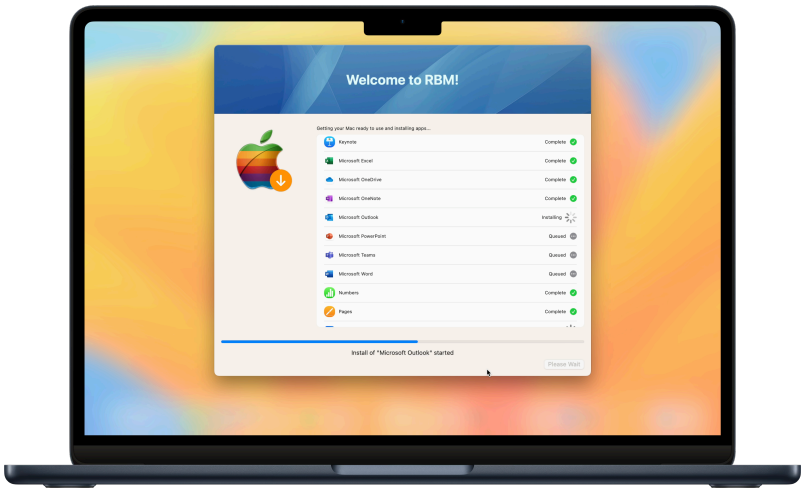
Once the wait to end time runs out, or the user clicks the button to close it, a 5 second countdown commences before the script exits and if necessary closes the dialog. If the computer is set to restart or shutdown or log out, this action is performed. If the script is set to quit, then the Jamf checkin LaunchDaemon is reloaded and caffeinate is killed to allow screen sleep to resume.

Configuration

The script can be configured in one of two ways - either by editing the values in the script, or by installing a configuration profile on the target computer to override script values.

Advantages of editing the script values are that it may be quicker to make changes, experiment and once set the options are hard-coded for all Macs.

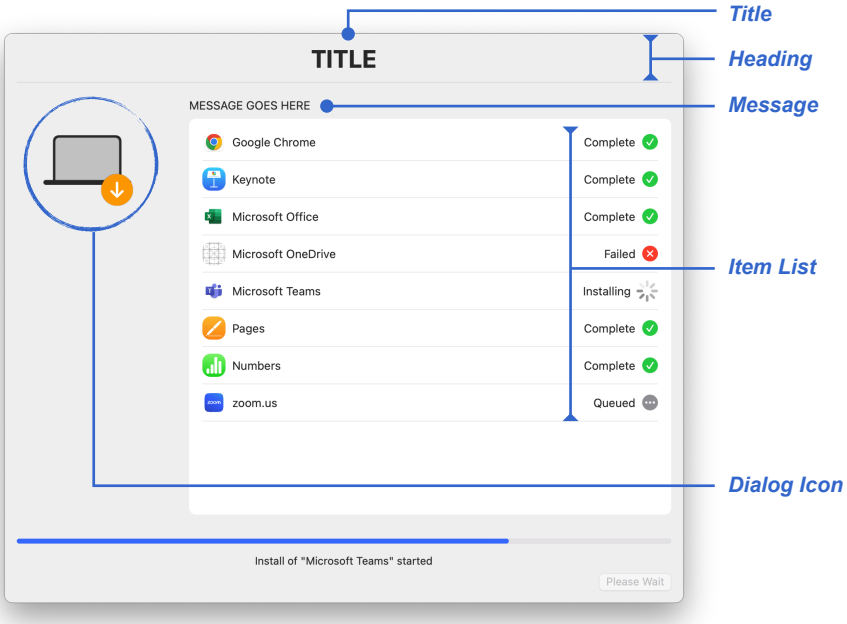
If deploying in an environment with multiple different configurations, the Configuration Profile method is more flexible and variations of the configuration profile can be deployed to Macs of each build to independently change what is installed or other behaviours.



Swift Dialog Appearance

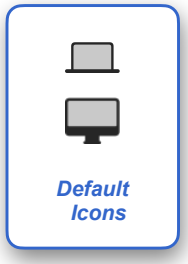
The diagram shows the configurable elements of the Swift Dialog

Dialog Overview



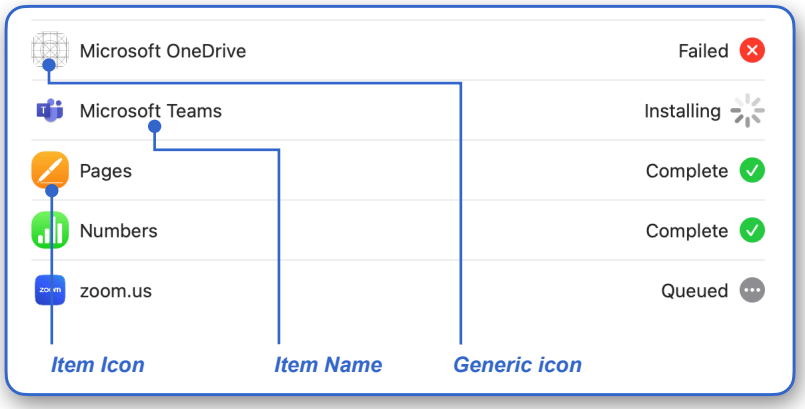
The Title and Message are mandatory parameters. Setting Title to “none” will cause it to be hidden.

If no icon is set, a generic SF Symbol for a Desktop or Laptop Mac is used with an overlaid progress indicator.



List Detail

Elements in the list are configured in the **Install List** array or using an array of item dictionaries if using a profile / preference file.



For each item, a Name is required. The icon can be defined or otherwise a generic icon can be defined and is used. The generic icon is replaced if an app is successfully installed using the icon of the app

Icons

The main icon is 170pt wide x 260pt high (340 x 520 pixels).
Icons will be scaled to fit in this box. JPG, PNG or ICNS can be used and will have rounded corners as shown.

For list item icons, a square format is recommended. Icons will vary in size depending on available screen space and will be cropped with rounded corners.

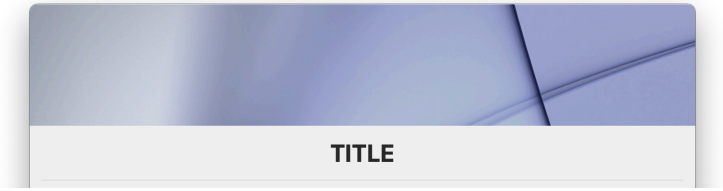
Icons in lists can typically use a URL for a Self Service app icon.

Banner Image

A Banner image can be used to add an image at the top of the dialog. Otherwise the title will appear at the top of the window

Example with no banner image.

When using a banner image, the Title can be layered over the image or positioned underneath, which is the default behaviour.



Example with banner image. Title not over background

Example with banner image. Title over background with custom colour (white)

The title text can also be set to a different colour if needed to stand out on top of the banner image. This can either be set using a colour name eg: “white, red, green” or a HEX value, eg: #1D97DE

Note: The title text could be changed in other ways such as size, font and adding a drop shadow, but this is currently not configurable without manually changing this in the script.

Example of image cropping

The banner image is scaled to fit the width of the dialog 820pt (typically 1640 pixels) and then cropped from the top up to 150pt (typically 300 pixels). If the banner has a wider ratio than 82:15, then the height of the banner is reduced. Eg: If an image 1800 x 224 pixels is used, the banner portion at the top of the banner will appear 1640 x 204 pixels (820 x 102 pt).

It is recommended to use an image of width 1640 and a height of 300 pixels or less to avoid unexpected cropping. Larger images can be used with the maximum height scaled proportionately

Method 1: Script Configuration

When configuring the script manually, copy the script and then edit the variables as outlined. Define the Install list with items to display on the dialog.
If a Preference File (typically deployed with a configuration profile) is found, all preferences will be overridden

Customisable Variables

Variable	Default / *Examples*	Note
prefFile	/Library/Managed Preferences/ com.rbm.swiftdialogbuild.plist	Path to Preference file to configure the script (if using a configuration profile)
title	“Setting Up…”	Text to display in the title area of the Swift Dialog box. If unset, falls back to “none" Can be set to “**none**” to hide
message	“Getting your computer ready to use and installing apps…	Text to display above list. If unset, falls back to “Starting setup…"
dialogIcon	*"/Users/Shared/images/academia-login-logo.png"* *“SF=car"*	Path, URL or SF Symbol to icon for Swift Dialog box. If left blank, a generic Mac desktop or laptop icon is used To use SF Symbol, prefix with SF= Icon files should ideally be PNG, ICNS or JPG.
fullScreen	blank *true* *false*	Run Swift Dialog with a blurred background to prevent users from doing other things while the build runs. Set to “**true**” or “**yes**” to run full screen. Set to **blank**, “**false**” or “**no**” to run in window
bannerImage	*“/Users/Shared/images/ academiabkgrnd.jpg”*	Image to use in banner section of Swift Dialog. If left blank, no image is used
titleColour	blank *orange* *#E5F3FF*	Colour for the title text. Leave blank for default colour (black) Pass either a colour name (eg: blue, red, etc) or a hex colour value (eg: #FFFFFF)
titleOnBanner	blank *true* *false*	If using a banner image, the title text can be below or superimposed over the banner image. Has no effect unless using a banner image. Set to “**true**” or “**yes**” to display text over banner. Set to **blank**, “**false**” or “**no**” display text below banner or if not using a banner.
buildFlagPath	blank */Library/Preferences/ com.rbm.swiftdialogbuild.plist*	If a record of the success or failure to complete the installation is required, set a path for that file which will be written as a property list (.plist). Date of build start and complete are also recorded. Leave blank to not write anything
buildName	blank “Art Department” “Room 101”	If buildFlagPath is set, a name for the build can also be recorded into the property list. This may be useful for recording what PreStage was originally used to enrol a device. Ignored if no buildFlagPath is set.

Variable	Default / *Examples*	Note
cleanUpPolicy	blank *true* *false*	Set to true to run any additional Check-in trigger Jamf Policies during the cleanup phase Set to "**yes**" or "**true**" to run 'sudo jamf policy' during cleanup phase Set to **blank**, "**false**" or "**no**" to skip this
cleanUpRecon	blank *true* *false*	Set to true to run a final Jamf Inventory Update during the cleanup phase Set to "**yes**" or "**true**" to run 'sudo jamf recon' during cleanup phase Set to **blank**, "**false**" or "**no**" to skip this
timeout	1800	Time (in seconds) from the script starting before the main loop exits. This should be enough time for the build to complete but is included to ensure unattended Macs do not get stuck waiting for items that have failed to install. If unset, defaults to 60 seconds
waitToClose	1800	Time (in seconds) from the build completing or timing for the dialog box to stay open. Falls back to 1 second minimum
installDialogURL	https://github.com/bartreardon/swiftDialog/releases/download/v2.1.0/dialog-2.1.0-4148.pkg	If SwiftDialog is not installed on the client Mac, it cannot be used to display messages and must be downloaded. If unset, the script will exit with an error if Swift Dialog is not already installed
exitBehaviour	blank *quit* *Restart* *logout*	Behaviour of script when script completes. Options are **quit**, **restart**, **logout** and **shutdown**. If Quit is set, then the SwiftDialog will quit and let the user start working immediately. If unset, falls back to Quit.
genericIcon	/System/Library/PrivateFrameworks/MobileIcons.framework/Versions/A/Resources/DefaultIcon-60@2x~iphone.png	Path or URL for a generic icon to use if no icon is set for an individual item to be installed. If no individual icon is configured for an app, its app icon is used once the app is installed. Until the app is installed, or if the item is not an app the generic icon is used. Icon should be in PNG, JPG or icns format.

Install List

The install list is declared as an array variable.

Each item to display in the dialog box must be defined as a line in the array and wrapped in quotes.
For each item, the following items must be defined in a comma separated line in order.
Colours are for illustration only.

Display Name, **Path of item**, **Policy Trigger**, **Icon path or URL**

Item	Notes	Required
Display Name	Name to use for item in the Dialog list	Required
Path of item	Path of installed item. This is required otherwise the dialog will timeout and item will be marked as failed	Required
Policy Trigger	If the item is installed using a Jamf Pro policy, the custom trigger name. For items not installed using a policy, leave blank	Only if installed by policy
Icon path or URL	Icon to use in list for item. Can be a path or URL. Icon should ideally be in PNG, ICNS or JPG format. If no icon is defined a generic icon can be set. If the item is an app, the generic icon will be replaced with the app's icon after installation.	No. Recommended for non-app items

Example:

```
installList=(
        "Google Chrome,/Applications/GoogleChrome.app,installGoogleChrome,/Users/Shared/icons/google.png"
        "Firefox,/Applications/Firefox.app,installFirefox,"
        "Pages,/Applications/Pages.app,,https://appstoreurl/pages.png"
        "Bind to Active Directory,/tmp/bind.txt,bindToAD,/Users/Shared/icons/bind.png"
)
```

In the example:
Google Chrome will install using the Jamf policy trigger installGoogleChrome.
Firefox will install using the Jamf policy trigger installFirefox and will be listed with a generic icon until the app is installed
Pages is installed from the App Store and should not have a policy trigger
Bind to Active Directory will install using the Jamf policy trigger bindToAD. The policy must write a file to /tmp/bind.txt once complete or the item will display as failed, even if it binds correctly.

The list in the interface will show items in the order that they are listed in the array.

Variables that shouldn't need updating

Variable	Default / *Examples*	Note
dialogApp	"/usr/local/bin/dialog"	Path for SwiftDialog binary
dialogCMDFile	/var/tmp/dialog.log	Path for SwiftDialog's log, used to update content during script
jamfBinary	/usr/local/bin/jamf	Path for Jamf Binary

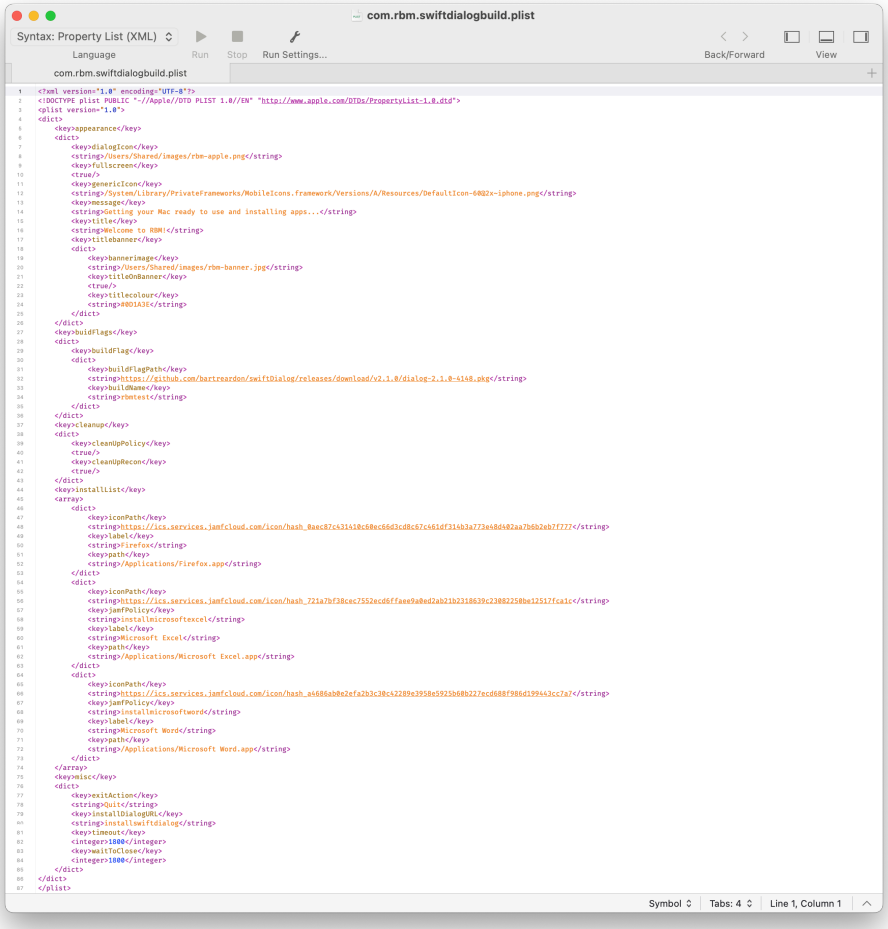
Method 2: Configuration Profile (Manual)

A preference file in XML Property List format can be created using the template in Appendix 2. This can be edited in a text editor or plist editing tool to customise values.

Preferences map to static variables in the script.

Key	Type	Value
Root	Dictionary	(5 items)
appearance	Dictionary	(6 items)
fullscreen	Boolean	YES
message	String	Getting your Mac ready to use and installing apps...
dialogIcon	String	/Users/Shared/images/rbm-apple.png
title	String	Welcome to RBM!
titlebanner	Dictionary	(3 items)
bannerimage	String	/Users/Shared/images/rbm-banner.jpg
titleOnBanner	Boolean	YES
titlecolour	String	#0D1A3E
genericIcon	String	/System/Library/PrivateFrameworks/MobileIcons.framewor
installList	Array	(3 items)
Item 0	Dictionary	(3 items)
label	String	Firefox
path	String	/Applications/Firefox.app
iconPath	String	https://ics.services.jamfcloud.com/icon/hash_0aec87c431
Item 1	Dictionary	(4 items)
jamfPolicy	String	installmicrosoftexcel
label	String	Microsoft Excel
iconPath	String	https://ics.services.jamfcloud.com/icon/hash_721a7bf38c
path	String	/Applications/Microsoft Excel.app
Item 2	Dictionary	(4 items)
jamfPolicy	String	installmicrosoftword
label	String	Microsoft Word
iconPath	String	https://ics.services.jamfcloud.com/icon/hash_a4686ab0e2
path	String	/Applications/Microsoft Word.app
buidFlags	Dictionary	(1 item)
buildFlag	Dictionary	(2 items)
buildFlagPath	String	https://github.com/bartreardon/swiftDialog/releases/down
buildName	String	rbmtest
misc	Dictionary	(4 items)
exitAction	String	Quit
waitToClose	Number	1800
timeout	Number	1800
installDialogURL	String	installswiftdialog
cleanup	Dictionary	(2 items)
cleanUpRecon	Boolean	YES
cleanUpPolicy	Boolean	YES

Example of editing in Xcode



Example of editing in CodeRunner

The plist can then be uploaded to Jamf Pro as a custom application profile to be deployed as a Configuration Profile

Key Paths

A plist can be generated manually using PlistBuddy to define the preference. Paths listed use PlistBuddy notation

Appearance Dictionary

Key	Path	Description	Type / Values	Required
title	:appearance:title	Text to display in the title area of the Swift Dialog box. If unset, falls back to “none" Can be set to “**none**” to hide	String	⚠️
Message	:appearance:message	Text to display above list. If unset, falls back to “Starting setup…"	String	⚠️
dialogIcon	:appearance:dialogIcon	Path, URL or SF Symbol to icon for Swift Dialog box. If left blank, a generic Mac desktop or laptop icon is used To use SF Symbol, prefix with SF= Icon files should ideally be PNG, ICNS or JPG.	String	
fullscreen	:appearance:fullscreen	Run Swift Dialog with a blurred background to prevent users from doing other things while the build runs. Set to “**true**” to run full screen. Set to “**false**” to run in window	Boolean	
genericIcon	:appearance:genericIcon	Path or URL for a generic icon to use if no icon is set for an individual item to be installed. If no individual icon is configured for an app, its app icon is used once the app is installed. Until the app is installed, or if the item is not an app the generic icon is used. Icon should be in PNG, JPG or icns format.	String	
bannerImage	:appearance:titlebanner:bannerImage	Path or URL for Image to use in banner section of Swift Dialog. If left blank, no image is used	String	
titleOnBanner	:appearance:titlebanner:titleOnBanner	If using a banner image, the title text can be below or superimposed over the banner image. Has no effect unless using a banner image. Set to “**true**”to display text over banner. Set to “**false**” display text below banner or if not using a banner.	Boolean	
titleColour	:appearance:titlebanner:titleColour	Colour for the title text. Leave blank for default colour (black) Pass either a colour name (eg: blue, red, etc) or a hex colour value (eg: #FFFFFF)	String	

Misc Dictionary

Key	Path	Description	Type / Values	Required
exitAction	:misc:exitAction	Behaviour of script when script completes. Options are **quit**, **restart**, **logout** and **shutdown**. If Quit is set, then the SwiftDialog will quit and let the user start working immediately. If unset, falls back to Quit.	Quit Restart Shut Down LogOut	⚠
timeout	:misc:timeout	Time (in seconds) from the script starting before the main loop exits. This should be enough time for the build to complete but is included to ensure unattended Macs do not get stuck waiting for items that have failed to install. If unset, defaults to 60 seconds	Integer	⚠
waitToClose	:misc:waitToClose	Time (in seconds) from the build completing or timing for the dialog box to stay open. Falls back to 1 second minimum	Integer	⚠
installDialog URL	:misc:installDialogURL	URL to download Swift Dialog pkg if not already installed. If not set and Swift Dialog is not installed, the script will execute with no GUI.	String	

Cleanup Dictionary

Key	Path	Description	Type / Values	Required
exitAction	:misc:exitAction	Behaviour of script when script completes. Options are **quit**, **restart**, **logout** and **shutdown**. If Quit is set, then the SwiftDialog will quit and let the user start working immediately. If unset, falls back to Quit.	Quit Restart Shut Down LogOut	⚠
timeout	:misc:timeout	Time (in seconds) from the script starting before the main loop exits. This should be enough time for the build to complete but is included to ensure unattended Macs do not get stuck waiting for items that have failed to install. If unset, defaults to 60 seconds	Integer	⚠
waitToClose	:misc:waitToClose	Time (in seconds) from the build completing or timing for the dialog box to stay open. Falls back to 1 second minimum	Integer	⚠

Build Flags Dictionary

Key	Path	Description	Type / Values	Required
buildFlagPath	:buildFlag:buildFlagPath	If a record of the success or failure to complete the installation is required, set a path for that file which will be written as a property list (.plist). Date of build start and complete are also recorded. Leave blank to not write anything	String	
buildName	:buildFlag:buildName	If buildFlagPath is set, a name for the build can also be recorded into the property list. This may be useful for recording what PreStage was originally used to enrol a device. Ignored if no buildFlagPath is set.	String	

Install List

The install list array defines which items will appear in the list. The list can contain as many items as needed - adding more items to the array.

For each item in the list the following should be defined

Key	Path	Description	Type / Values	Required
label	:installList:{index}:label	Name to use for item in the list, eg; Google Chrome	String	⚠
path	:installList:{index}:path	Path of installed item. This is required otherwise the dialog will timeout and item will be marked as failed	String	⚠
jamfPolicy	:installList: {index}:jamfPolicy	If the item is installed using a Jamf Pro policy, set the custom trigger name. For items not installed using a policy, leave blank or do not include this key	String	⁉
iconPath	:installList: {index}:iconPath	Icon to use in list for item. Can be a path or URL. Icon should ideally be in PNG, ICNS or JPG format. If no icon is defined a generic icon can be set. If the item is an app, the generic icon will be replaced with the app's icon after installation.	String	

Deployment

Once created, the preference can be installed at the path defined in the prefFile variable in the script, or added as an upload to a Custom and Application profile in Jamf Pro.

Method 3: Configuration Profile (Custom Schema)

The configuration profile can also be generated in Jamf Pro using the Custom Schema in Appendix 3

Adding the Custom Schema

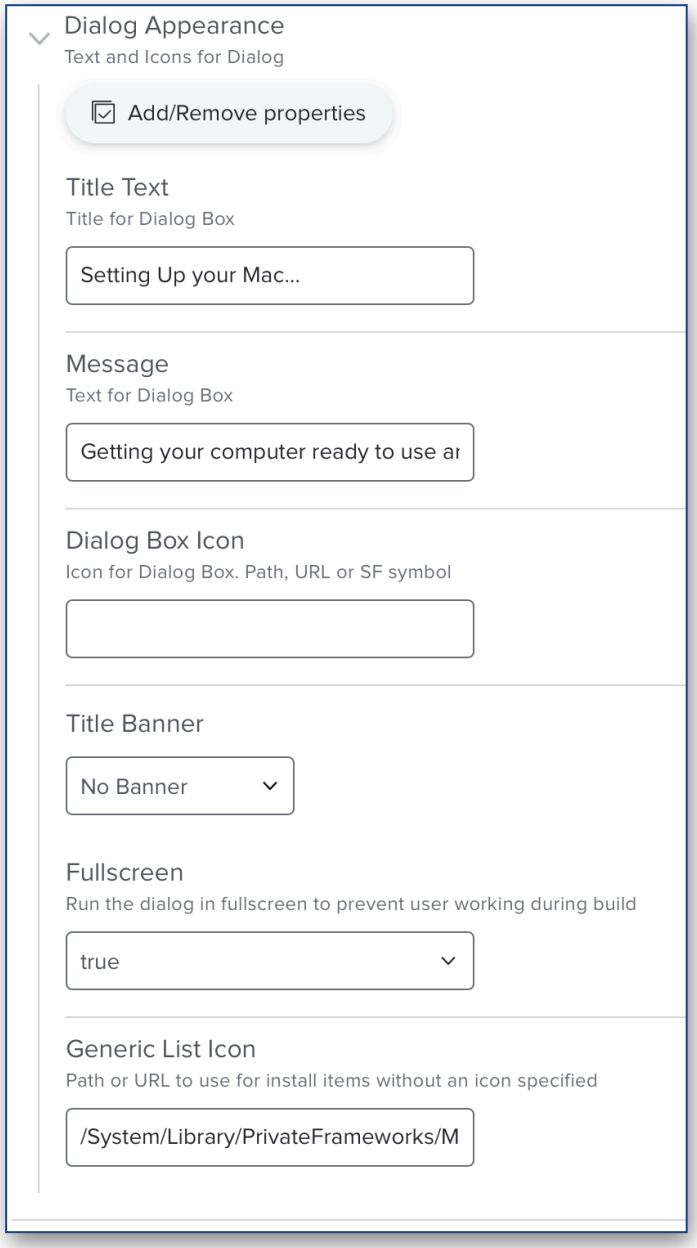
1. In Jamf Pro, create a Configuration Profile. Configure the Name, Category, etc as normal
2. Add the Custom and Application Payload > External Applications
3. From the Source drop down, select Custom Schema
4. Set a preference domain that matches the prefFile parameter set in the script
5. Click Add Schema
6. Add the Custom schema using a method below
    1. Click Upload, choose the json file for the schema. Click Save
    2. Paste the custom schema into the box. Click Upload. Click Save
7. Complete the form and deploy the configuration profile.

The form will be available in the Jamf Pro interface. Default values will be populated for items.

Configuring the Custom Schema

The form is split into 4 sections to configure elements of the dialog and build process

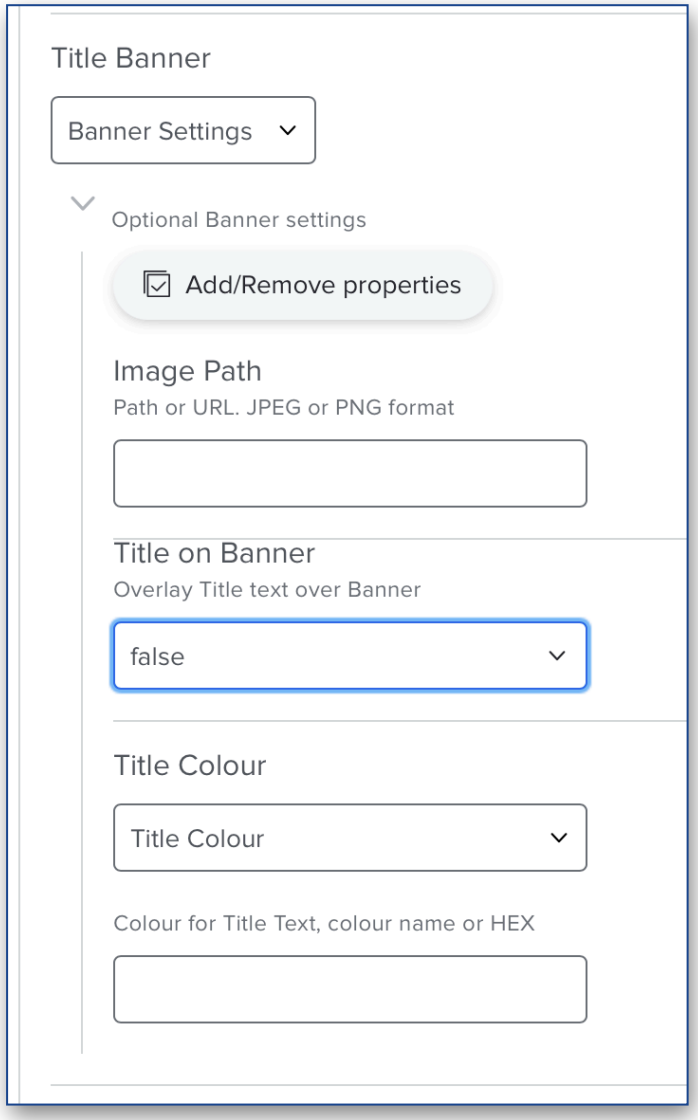
Dialog Appearance



Item	Decription	Default Provided	Required
Title Text	Title Text to display on dialog box. Can be set to “none” to hide	✅	✅
Message	Text shown at top of dialog box	✅	✅
Dialog Box Icon	Path, URL or SF symbol to use for dialog box		❌
Title Banner	Additional options shown if a banner image needs to be configured		❌
Full Screen	True or Balse	✅	✅
Generic List Icon	Path or URL for icon to use if an item in the list does not have an icon defined. PNG, JPG or ICNS	✅	✅

Selecting Banner Settings for the Title Banner shows additional configuration for the banner

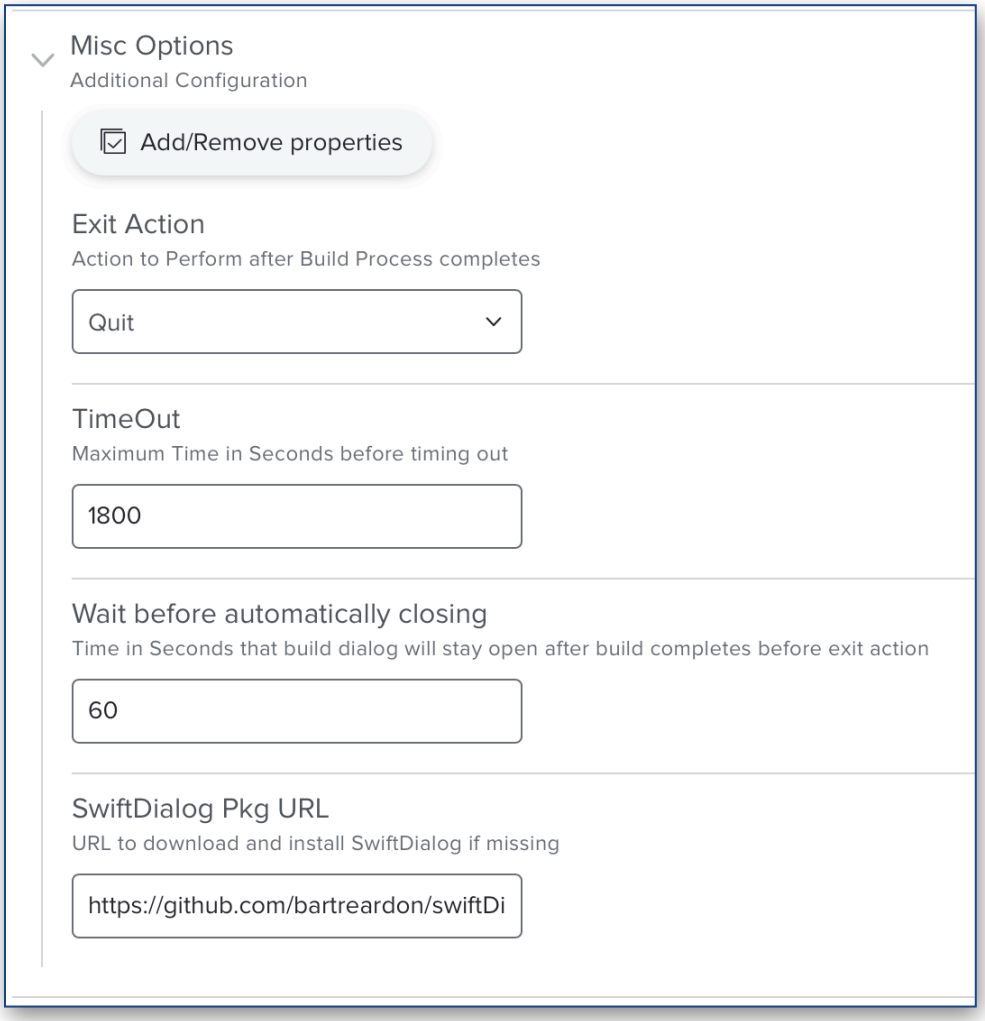
Item	Decription	Default Provided	Required
Image Path	Path or URL to image to display as banner at the top of the dialog	❌	⚠️
Title on Banner	If true, the title text is layered over the image. If false (default) the title text appears underneath the banner	✅	❌
Title Colour	Title Text will appear Black as default. This can be overriden using either a colour name (blue, green, red, etc) or a HEX colour value (#1D97DE)	✅	❌



Misc Options

These are non-UI configuration items

Item	Decription	Default Provided	Required
Exit Action	Action after the Build Process completes. Quit, Shut Down, Restart or Log Out.	✅	✅
TimeOut	Time in seconds from start of script before main loop times out. Any items not installed will be marked as failed.	✅	❌
Wait before automatically closing	Time in seconds for the dialog to stay open after all items in the build complete (or time out) before exit action	✅	❌
SwiftDialog Pkg URL	URL to download a SwiftDialog pkg if the binary is missing. If not provided and not installed, no dialog will be displayed	✅	❌



Cleanup

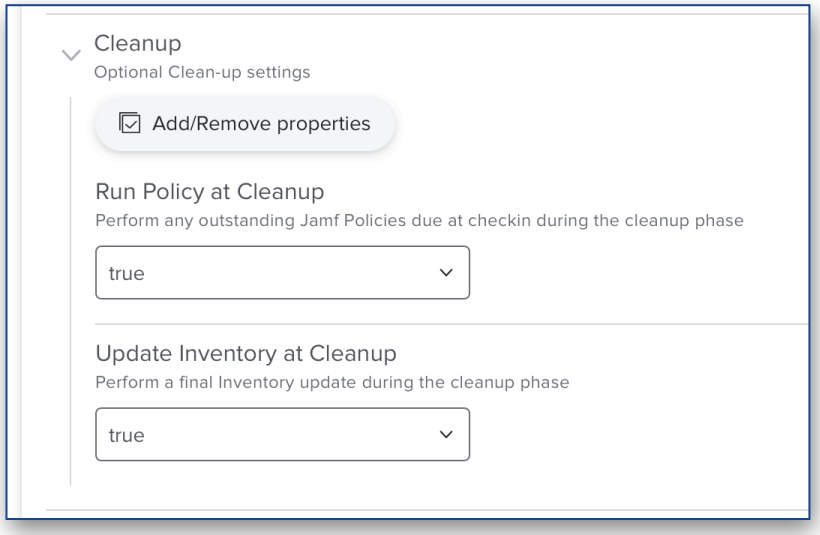
Cleanup options are run after the main list has either completed or timed out.

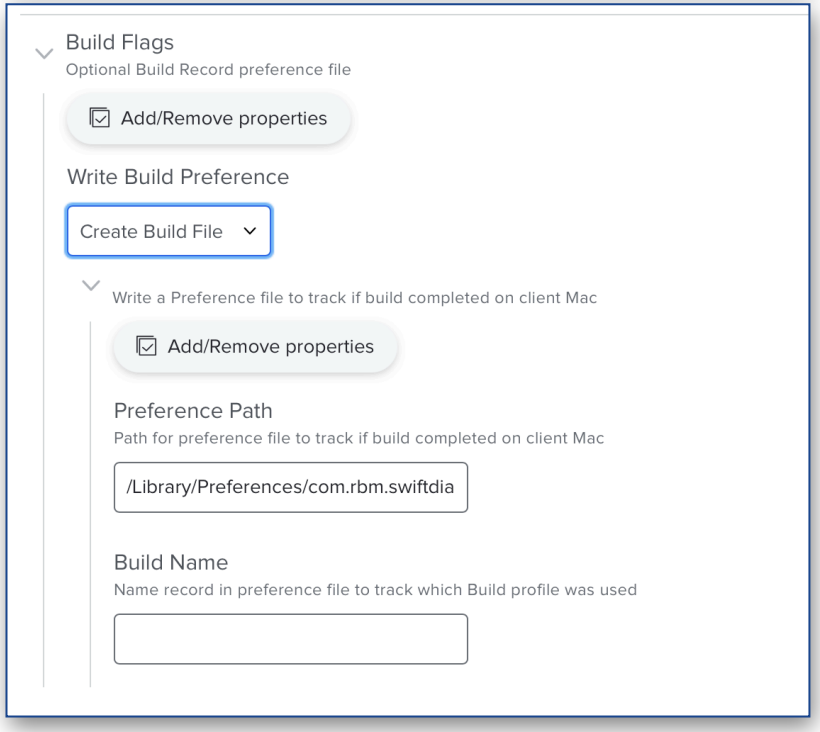
Item	Decription	Default Provided	Required
Run Policy at Cleanup	If true, any Jamf Policies triggered by check-in will run after the list / timeout	✅	❌
Update Inventory at Cleanup	If true, an inventory update will be submitted after any policies are run and any flags are written (if defined)	✅	❌

Build Flags

Build Flags are written during the cleanup. This creates a preference file (plist) at the defined path as a record of the build process. If written, the preference will contain a key "**buildStatus**" displaying either "Complete" or "Failed" to track whether the build worked or not.
Additionally a Build Name can written to the preference using the key "**buildType**" with an arbitrary name for the build for future reference

Item	Decription	Default Provided	Required
Preference Path	Path to record build status preference	✅	❌
Build Name	Name of Build used	✅	❌





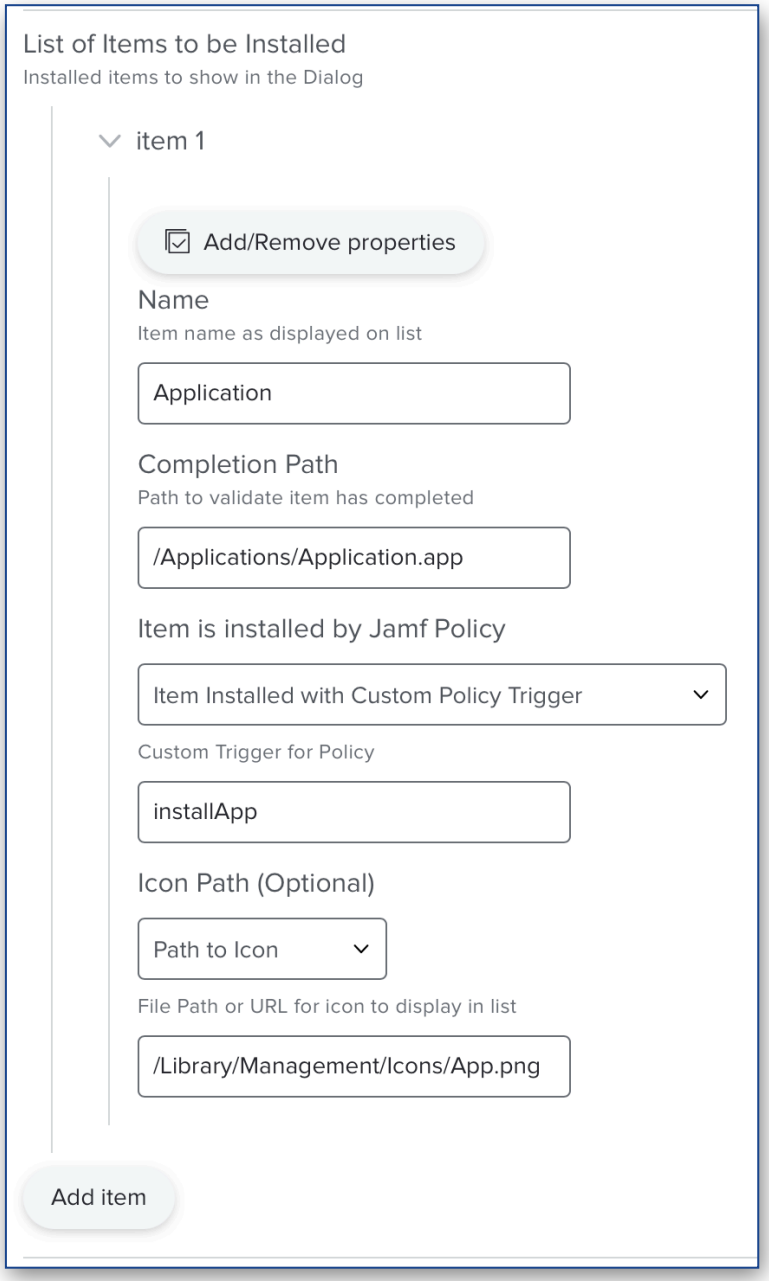
List of Items to be Installed

This list defines each item in the list that is installed

As many items as required can be added by clicking “Add item” at the bottom of the list

Item	Decription	Default Provided	Required
Name	Name displayed in list		✅
Completion Path	Path of a file that should exist if the item installs correctly. Typically the path for an app. If not defined or item is missing, item will be marked as failed.		✅
Item is installed by Jamf Policy	Either **Item is installed with Custom Policy Trigger** or **Item is installed via Jamf Catalog or Mac App Store** This displays / hides the Custom Trigger box		
Custom Trigger for Policy	Jamf Policy custom trigger to install item. Must be defined for any item installed with a custom trigger		⚠️
Icon Path	Either **Path to Icon** or **Use Default Icon** This displays / hides the URL box. If no path is defined, the generic icon is used		
File Path or URL for icon to display in list	Path or URL for icon to display for item.		⚠️

The list is displayed in the order of the configuration profile - so using the up and down buttons can set the order either by priority or alphabetically as desired.



Extension Attributes

Each EA will need to have Plist paths updated to match the Configuration Profile Domain or Build Preference path.

Scoping EA

The Scoping EA is used to extract a list of every item expected to be installed during the build process.

It works by reading the Install List from the preference file or configuration profile to determine a list of items.

This can be used to create scoping Smart Groups to deploy apps from the Jamf Catalog or Mac App Store.

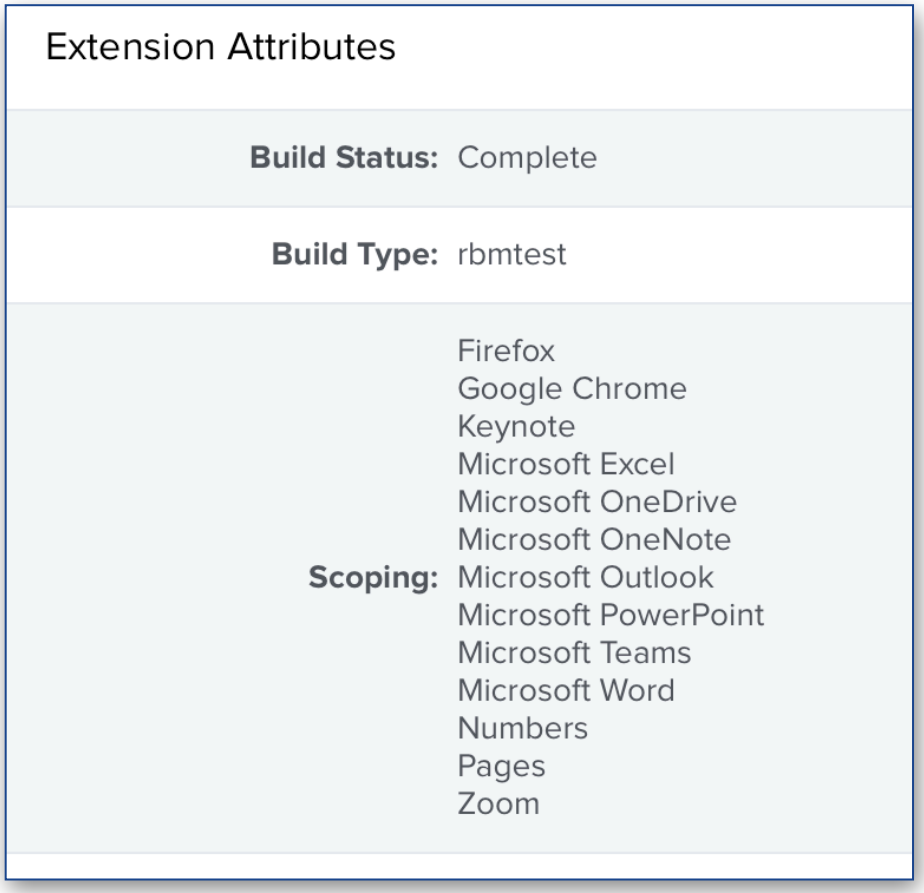
As the script runs a recon near the start of execution, this can be used to trigger the items configured in the preference file / Configuration profile.

Build Status EA

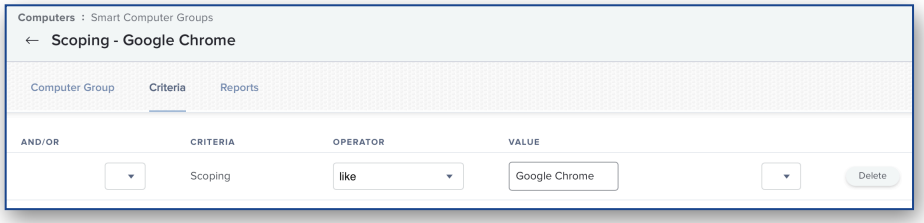
If the Build flag is configured when running the build script, a preference file can be written to the client Mac.
This EA will read the build key in that file and provide a result of either Complete or Failed.

Build Name EA

If the Build flag is configured when running the build script, a build name can also be written to the preference file
This EA will read the buildtype key in that file and provide a result of whatever arbitrary string was used as the build name.



Example results of each Extension Attribute



Example Smart Group for scoping Google Chrome based on the result of the EA

Appendix

Appendix 1: Swift Dialog Build Script

```
#!/bin/bash

###### Swift Dialog Build Dialog
###### By Richard Brown-Martin 2023

#### Static Variables
# All Static Variables can be overridden by the Configuration Profile
prefFile="/Library/Managed Preferences/com.rbm.swiftdialogbuild.plist"

## Basic Dialog Settings
# Title for top of dialog
title="Setting Up..."
# Main message for dialog
message="Getting your computer ready to use and installing apps..."
# Icon for the for the Dialog - replaced with a computer if left blank
dialogIcon="/Users/Shared/images/academia-login-logo.png"

## Dialog Appearance
# Blur the background of the screen. Pass anything other than false or no to
    activate.
fullScreen=""
# Banner Image - Path or URL for an image to use in the banner
bannerImage="/Users/Shared/images/rbm-strip.jpg"
#bannerImage=""
# Colour for Title Text - either a colour name or HEX
titleColour=""
# Title Text superimposed over banner image. Pass anything other than false or
    no to activate.
titleOnBanner="false"

### Build Information
# Optional Path for a "build flag" preference file to write at the end of installation
buildFlagPath=""
# Optional Build Name to write as part of "build flag"
buildName=""

### Cleanup Tasks
# Run any outstanding checkin policies during cleanup. Pass anything to
    activate or leave blank to skip.
cleanUpPolicy=""
# Run final inventory update during cleanup. Pass anything to activate or leave
    blank to skip.
cleanUpRecon=""

### Additional configuration
# Time Out (in seconds) to abort the main loop
timeout=1800
# Waiting time before dialog automatically closes (in seconds)
waitToClose=1800
# Jamf Policy to run if Dialog is not already installed. Completely optional.
installDialogURL="https://github.com/bartreardon/swiftDialog/releases/
    download/v2.1.0/dialog-2.1.0-4148.pkg"
# Exit behaviour - Reboot, Logout or Exit (default exit)
exitBehaviour=""

# Generic Icon path
genericIcon="/System/Library/PrivateFrameworks/MobileIcons.framework/
    Versions/A/Resources/DefaultIcon-60@2x~iphone.png"

### List of install items in format...
# "Label, Path for completed item, Policy Trigger (If necessary), Icon path or
    URL"

# Testing Install Array 1
#installList=(
# "Google Chrome,/Applications/Google Chrome.app,,"
# "Keynote,/Applications/Keynote.app,,https://ics.services.jamfcloud.com/icon/
    hash_f0048723c3075d745f4104f7f95ccb059689f4523ad4411aab3f2754061929f3"
# "Microsoft Excel,/Applications/Microsoft Excel.app,installexcel,https://
    ics.services.jamfcloud.com/icon/
    hash_721a7bf38cec7552ecd6ffaee9a0ed2ab21b2318639c23082250be12517fca1c"
# "Microsoft OneDrive,/Applications/OneDrive.app,,"
# "Microsoft Outlook,/Applications/Microsoft Outlook.app,installOutlook,https://
    ics.services.jamfcloud.com/icon/
    hash_b96ae8bdcb09597bff8b2e82ec3b64d0a2d17f33414dbd7d9a48e5186de7fd93"
# "Microsoft PowerPoint,/Applications/Microsoft
    Powerpoint.app,installPowerpoint,"
# "Microsoft Teams,/Applications/Microsoft Teams.app,,https://
    ics.services.jamfcloud.com/icon/
    hash_623505d45ca9c2a1bd26f733306e30cd3fcc1cc0fd59ffc89ee0bfcbfbd0b37e"
# "Microsoft Word,/Applications/Microsoft Word.app,,https://
    ics.services.jamfcloud.com/icon/
    hash_a4686ab0e2efa2b3c30c42289e3958e5925b60b227ecd688f986d199443cc7a7"
```

```
# "Pages,/Applications/Pages.app,,https://ics.services.jamfcloud.com/icon/
    hash_897f79c537a547a999cad36652d81cb34a23ff4fa8f028d061e40e26f6
    ad38a7"
# "Numbers,/Applications/Numbers.app,,https://ics.services.jamfcloud.com/
    icon/
    hash_97266d57fd2811e4dd3a0668bcd2c5c80bcbfbb29362dd5781015d219
    0844f38"
# "zoom.us,/Applications/zoom.us.app,,https://ics.services.jamfcloud.com/icon/
    hash_ad4aa8f819b365a848b798b32f3d42438cbfef2d5c62dd02283132673
    ec23871"
#)

# Testing Install Array 2
installList=(
 "Google Chrome,/Applications/Google Chrome.app,,"
 "Keynote,/Applications/Keynote.app,,https://ics.services.jamfcloud.com/icon/
    hash_f0048723c3075d745f4104f7f95ccb059689f4523ad4411aab3f275406
    1929f3"
 "Pages,/Applications/Pages.app,,https://ics.services.jamfcloud.com/icon/
    hash_897f79c537a547a999cad36652d81cb34a23ff4fa8f028d061e40e26f6
    ad38a7"
 "Numbers,/Applications/Numbers.app,,https://ics.services.jamfcloud.com/icon/
    hash_97266d57fd2811e4dd3a0668bcd2c5c80bcbfbb29362dd5781015d219
    0844f38"
 "zoom.us,/Applications/zoom.us.app,,https://ics.services.jamfcloud.com/icon/
    hash_ad4aa8f819b365a848b798b32f3d42438cbfef2d5c62dd02283132673
    ec23871"
)


## Static variables that are unlikely to need to be changed

# Location of dialog, dialog command file and Jamf binary
dialogApp="/usr/local/bin/dialog"
dialogCMDFile="/var/tmp/dialog.log"
jamfBinary="/usr/local/bin/jamf"


#### Functions

### Dialog Commands
dialog_command(){
 echo "$(date): $1"
 echo "$1" >> $dialogCMDFile
}

###
startUp(){
 # Record Start Time for Timeouts
 startTime=$(date +%s)
 # Stop the screen from sleeping while building
 caffeinate -dimsu &
 # Stop Jamf from doing anything unexpected during setup
 launchctl unload /Library/LaunchDaemons/com.jamfsoftware.task.1.plist
 #Perform an initial recon
 $jamfBinary recon &
 jamfPID=$!
}

### Get a specific key from preferences
readPref(){
   /usr/libexec/PlistBuddy -c "Print $1" "$prefFile" 2>/dev/null
 }

### Load in all Preferences
getPrefs(){
 if [  -z "$exitBehaviour" ]; then
  exitBehaviour="Quit"
 fi
 if [ -e "$prefFile" ] && [ -n "$prefFile" ] ; then
  prefIndex=0
  unset installList
  while [ -n "$prefItemTitle" ] || (( $prefIndex < 1 )); do
   prefItemTitle="$(readPref "installList:$prefIndex:label")"
   prefItemIcon="$(readPref "installList:$prefIndex:iconPath")"
   prefItemPath="$(readPref "installList:$prefIndex:path")"
   prefItemPolicy="$(readPref "installList:$prefIndex:jamfPolicy")"
   if [ -n "$prefItemTitle" ]; then
    installList+=("$prefItemTitle,$prefItemPath,$prefItemPolicy,
    $prefItemIcon,")
   fi
   prefIndex=$(( $prefIndex + 1 ))
  done

  title="$(readPref "appearance:title")"
  message=$(readPref "appearance:message")
  dialogIcon="$(readPref "appearance:dialogIcon")"

  fullScreen="$(readPref "appearance:fullscreen")"
  bannerImage="$(readPref "appearance:titlebanner:bannerimage")"
  titleColour="$(readPref "appearance:titlebanner:titlecolour")"
  titleOnBanner="$(readPref "appearance:titlebanner:titleOnBanner")"

  buildFlagPath="$(readPref "buidFlags:buildFlag:buildFlagPath")"
  buildName="$(readPref "buidFlags:buildFlag:buildName")"
```

```
    cleanUpPolicy="$(readPref "cleanup:cleanUpPolicy")"
    cleanUpRecon="$(readPref "cleanup:cleanUpRecon")"

    installDialogURL="$(readPref "misc:installDialogURL")"
    exitBehaviour="$(readPref "misc:exitAction")"
    timeout=$(( $(readPref "misc:timeout") ))
    waitToClose=$(( $(readPref "misc:waitToClose") ))

  fi

  # Fix issues
  if (( $timeout < 60 )) || [ -z $timeout ]; then
    echo "Timeout should not be less than 1 minute, resetting to 5 minutes"
    timeout=300
  fi

  if (( $waitToClose < 1 )); then
    echo "Wait to close must be at least 1 second"
    waitToClose=1
  fi

  if [  -z "$title" ]; then
    title=none
  fi

  if [ -z "$message" ]; then
    message="Starting setup..."
  fi

  if [ -z "$bannerImage" ]; then
    unset $titleOnBanner
  fi

  # Count the items in the array
  arrayLength=${#installList[@]}
  completionLength=$(($arrayLength + 1))
  # Initialise counters
  completion=0
  jamfIndex=0
  dialogUpdates=( )
}

### Launch Dialog
launchDialog(){

  # Set an icon
  if [ -z "$dialogIcon" ]; then
    if system_profiler SPPowerDataType | grep -q "Battery Power" ; then
      dialogIcon="SF=laptopcomputer"
    else
      dialogIcon="SF=desktopcomputer"
    fi
  fi

  # Calculate height of the dialog, maxing out at 90% of screen height
  pixHeight=$(( ( 90 * $arrayLength ) + 455 ))

  # Dialog optional items
  if [ -z "$fullScreen" ] || [ "$fullScreen" = "false" ] || [ "$fullScreen" = "no" ]; then
    blurscreen=""
  else
    blurscreen="--blurscreen"
  fi

  if [  -n "$bannerImage" ]; then
    banner="--bannerimage \"$bannerImage\""
    fullBannerHeight=$(sips -g pixelHeight "$bannerImage"| awk -F : '{print $2}')
    fullBannerWidth=$(sips -g pixelWidth "$bannerImage" | awk -F : '{print $2}')
    correctedBannerHeight=$(( $fullBannerHeight * 1640 / $fullBannerWidth ))
    if (( $correctedBannerHeight > 300 )); then
      correctedBannerHeight=300
    fi
    pixHeight=$(( $pixHeight + $correctedBannerHeight ))
  fi

  if [  -n "$titleColour" ]; then
    titleFont="--titlefont colour=$titleColour"
  fi

  if [ -z "$titleOnBanner" ] || [ "$titleOnBanner" = "false" ] || [ "$titleOnBanner" =
      "no" ]; then
    bannerTitle=""
  else
    if [ -n "$bannerImage" ]; then
      bannerTitle="--bannertitle"
      pixHeight=$(( $pixHeight - 100 ))
    else
      bannerTitle=""
    fi
  fi



  screenheights=$(system_profiler SPDisplaysDataType | grep Resolution | awk
      '{print $4}')
  for height in $screenheights; do
```

```
 if [ -z "$maxPixHeight" ] || (( $height < $maxPixHeight )); then
  maxPixHeight=$height
 fi
done
maxPixHeight=$(( $maxPixHeight * 9 / 10)) # Adjusted Max height is 90% of
    smallest display

if (( $maxPixHeight < $pixHeight )); then
 pointHeight=$(( ( $maxPixHeight - 55 ) / 2 ))
 fontsize=$(( $maxPixHeight / $pixHeight * 14))
 if (( $fontsize < 10 )); then
  fontsize=10
 fi
else
 pointHeight=$(( ( $pixHeight - 55 ) / 2 ))
 fontsize=14
fi


# dialogIcon="car"
# Configure the dialog
 dialogCMD="$dialogApp -p -o \
--title \"$title\" \
--message \"$message\" \
--messagefont \"size=$fontsize\" \
--icon \"$dialogIcon\" \
--overlayicon SF=arrow.down.circle.fill,palette=white,white,orange,bgcolor=none
    \
--progress $arrayLength \
--button1text \"Please Wait\" \
--button1disabled \
--height $pointHeight"

 # create the list of apps
 listitems=""
 for app in "${installList[@]}"; do
  listitems="$listitems --listitem '$(echo "$app" | cut -d ',' -f1)'"
 done


 # final command to execute
 dialogCMD="$dialogCMD $listitems $blurscreen $banner $bannerTitle
    $titleFont"

 # Launch dialog and run it in the background sleep for a second to let thing
    initialise
 eval "$dialogCMD" &
 dialogPID=$!
 running=2
 sleep 1
}

updateArrayItem(){
 updateArrayIndex=$1
 updateArrayNewState=$2
 # Breakdown the existing item
 updateArrayName="$(echo "${installList[$updateArrayIndex]}" | cut -d ',' -f1)"
 updateArrayPath="$(echo "${installList[$updateArrayIndex]}" | cut -d ',' -f2)"
 updateArrayPolicy="$(echo "${installList[$updateArrayIndex]}" | cut -d ',' -f3
    | xargs )"
 updateArrayAppIcon="$(echo "${installList[$updateArrayIndex]}" | cut -d ','
    -f4)"
 updateArrayOldState="$(echo "${installList[$updateArrayIndex]}" | cut -d ','
    -f5)"
 if [ "$updateArrayOldState" != "$updateArrayNewState" ]; then
 # Update array with the new state
 installList[$updateArrayIndex]="$updateArrayName,$updateArrayPath,
    $updateArrayPolicy,$updateArrayAppIcon,$updateArrayNewState"
 # Add item to list that need their dialogs updated
 dialogUpdates+=($updateArrayIndex)
 # If something has completed or failed then add a tally to the completion
    counter
  if [ "$updateArrayNewState" = "complete" ] || [ "$updateArrayNewState" =
    "failed" ]; then
  completion=$(($completion + 1 ))
  fi
 fi
}

updateDialog(){
 updateDialogIndex=$1
 removeIndex=0
 removeLength=${#dialogUpdates[@]}
 newDialogUpdates=( )
 while (( $removeIndex < $removeLength )); do
  itemAtRemoveIndex="${dialogUpdates[$removeIndex]}"
  if [ "$updateDialogIndex" != "$itemAtRemoveIndex" ]; then
   newDialogUpdates+=($itemAtRemoveIndex)
  fi
  removeIndex=$(( $removeIndex + 1 ))
 done
 dialogUpdates=( "${newDialogUpdates[@]}" )
 # Get the relevant item info
 updateDialogName="$(echo "${installList[$updateDialogIndex]}" | cut -d ','
    -f1)"
```

```
updateDialogAppIcon="$(echo "${installList[$updateDialogIndex]}" | cut -d ',' -f4)"
updateDialogState="$(echo "${installList[$updateDialogIndex]}" | cut -d ',' -f5)"
# If no icon set, either use the generic item or if the install is complete, try the app icon
if [ -z "$updateDialogAppIcon" ]; then
 if [ "$updateDialogState" = "complete" ]; then
  updateDialogArrayPath="$(echo "${installList[$updateDialogIndex]}" | cut -d ',' -f2)"
  if [[ "$updateDialogArrayPath" = *.app ]]; then
  updateDialogAppIcon="$updateDialogArrayPath"
  else
   echo "updateDialogName is using $genericIcon"
   updateDialogAppIcon="$genericIcon"
  fi
 else
  echo "updateDialogName is using $genericIcon"
  updateDialogAppIcon="$genericIcon"
 fi
fi
# From the status set the icon and message
case $updateDialogState in
 complete)
  updateDialogStatus="success"
  updateDialogText="Complete"
  dialog_command "progresstext: Install of \"$updateDialogName\" complete"
 ;;
 failed)
  updateDialogStatus="fail"
  updateDialogText="Failed"
  dialog_command "progresstext: Install of \"$updateDialogName\" failed"
 ;;
 installing)
  updateDialogStatus="wait"
  updateDialogText="Installing"
  dialog_command "progresstext: Install of \"$updateDialogName\" started"
 ;;
 pending)
  updateDialogStatus="pending"
  updateDialogText="Queued"
 ;;
 waiting)
  updateDialogStatus="wait"
  updateDialogText="Waiting"
 ;;
 *)
  echo "Something has gone wrong, $updateDialogState"
 ;;
 esac
 dialog_command "listitem: title: $updateDialogName, status: $updateDialogStatus, statustext: $updateDialogText, icon: $updateDialogAppIcon"
 dialog_command "progress: $completion"
}

# Once the dialog is up, populate the list with icons and statuses
populateList(){
 dialog_command "progresstext: Waiting to Start..."
 populateIndex=0
 while (( $populateIndex < $arrayLength )); do
  populatePolicy="$(echo "${installList[$populateIndex]}" | cut -d ',' -f3 | xargs )"
  populateState="$(echo "${installList[$populateIndex]}" | cut -d ',' -f5)"
  # Only update state if not already set
  if [ -z "$populateState" ]; then
   # Determine if waiting or pending based on if item is a policy or not
   if [ -z "$populatePolicy" ]; then
    updateArrayItem $populateIndex "waiting"
   else
    updateArrayItem $populateIndex "pending"
   fi
#   updateDialog $populateIndex
  fi
  populateIndex=$(( $populateIndex + 1 ))
 done
}


## Jamf Checker - Checks if Jamf is running and starts it if necessary

jamfChecker(){
 # Check if all Jamf Policies have been run
 if (( $jamfIndex < $arrayLength )); then # Continue only if the Jamf Counter is less than the total items in the array
  # Check if Jamf is already running or not
  if [ -z "$jamfPID" ]; then # If no PID then start straight away
   jamfStarter
  else
   jamfRunningStatus=$(ps aux | grep -c "$jamfPID")
   if (( $jamfRunningStatus < 2 )); then #If Jamf is no longer then the previous policy has finished
    # Check if the last policy worked
    jamfCheckerInstallCheckState=$(echo "${installList[$jamfIndex]}" | cut -d ',' -f5)
    if [ "$jamfCheckerInstallCheckState" != "complete" ] &&
    [ "$jamfCheckerInstallCheckState" != "failed" ]; then
```

```
        # Check if the criteria is met, mark as complete once it is
        jamfCheckerInstallCheckPath="$(echo "${installList[$jamfIndex]}" | cut -d
    ',' -f2)"
        jamfCheckerInstallpolicyPath="$(echo "${installList[$jamfIndex]}" | cut -d
    ',' -f3 | xargs)"
        if [  -e "$jamfCheckerInstallCheckPath" ]; then
          if [  -n "$jamfCheckerInstallpolicyPath" ]; then
          updateArrayItem $jamfIndex "complete"
          fi
        else
          if [  -n "$jamfCheckerInstallpolicyPath" ]; then
            updateArrayItem $jamfIndex "failed"
          fi
        fi
      fi
      # Iterate the policy loop and unset the PID ready for the next policy, then
     get the next policy going.
       jamfIndex=$(( $jamfIndex + 1 ))
       unset jamfPID
       if (( $jamfIndex < $arrayLength )); then
         jamfStarter
       fi
     fi
   fi
 fi
}


## Jamf Starter - Starts the next Jamf policy
jamfStarter(){
  # Find the next item that has a policy trigger
  if (( $jamfIndex < $completionLength )); then
  unset jamfPolicy
  while [ -z "$jamfPolicy" ] && (( $jamfIndex < $completionLength )); do
    jamfPolicy="$(echo "${installList[$jamfIndex]}" | cut -d ',' -f3 | xargs )"
    if [ -z "$jamfPolicy" ]; then
      jamfIndex=$((jamfIndex + 1))
    else
      echo "$(date): Starting Policy: $jamfPolicy"
      updateArrayItem $jamfIndex "installing"
      $jamfBinary policy -event $jamfPolicy  -forceNoRecon &
      jamfPID=$!
      sleep 1
    fi
  done
  else
    echo "$(date): Jamf policies complete!"
  fi
}

# Wait for Dock before launching DEP Nofity. Important for DEP builds
waitToStart() {

 until [ -e "$dialogApp" ] && [ -n "$(pgrep -l "Dock")" ]; do
   jamfChecker
   sleep 3
 done

}

# Finalise Dialog if everything worked
finaliseGood(){
 dialog_command "overlayicon:
    SF=checkmark.circle.fill,palette=white,white,green,bgcolor=none"
 dialog_command "progresstext: Installation complete"
 dialog_command "progress: complete"
 dialog_command "button1text: $exitBehaviour"
 dialog_command "button1: enable"
}

# Finalise Dialog if anything failed
finaliseBad(){
 finaliseIndex=0
 while (( $finaliseIndex < $arrayLength )); do
     finaliseCheckState="$(echo "${installList[$finaliseIndex]}" | cut -d ',' -f5)"
     if [ "$finaliseCheckState" != "complete" ] && [ "$finaliseCheckState" !=
    "failed" ]; then
       updateArrayItem $finaliseIndex failed
     fi
  finaliseIndex=$(( $finaliseIndex + 1))
 done

 refreshDialog
 dialog_command "overlayicon:
    SF=x.circle.fill,palette=white,white,red,bgcolor=none"
 dialog_command "progresstext: Installation has completed but some items
    have failed"
 dialog_command "progress: complete"
 dialog_command "button1text: $exitBehaviour"
 dialog_command "button1: enable"

 exitCode=1
}

# Install Checker
installChecker(){
```

```
 installCheckIndex=0
 while (( $installCheckIndex < $arrayLength )); do
  # Find the State and if it's a policy installed item
  installCheckState="$(echo "${installList[$installCheckIndex]}" | cut -d ',' -f5)"
  installCheckSkipJamf="$(echo "${installList[$installCheckIndex]}" | cut -d ','
    -f3 | xargs)"
  # Skip checking items that are already installed or run from a Jamf policy
  if [ "$installCheckState" != "complete" ] && [ "$installCheckState" != "failed" ]
    && [ -z "$installCheckSkipJamf"  ]; then
    # Check if the criteria is met, mark as complete once it is
    installCheckPath="$(echo "${installList[$installCheckIndex]}" | cut -d ',' -f2)"
    if [  -e "$installCheckPath" ]; then
      updateArrayItem $installCheckIndex "complete"
    fi
  fi
  installCheckIndex=$(( $installCheckIndex + 1 ))
 done
}

# Check time running
checkTimeout(){
 timeNow=$(date +%s)
 timeRunning=$(( $timeNow - $startTime ))
}

# Refresh all items in the dialog
refreshDialog(){
 for indexToDraw in ${dialogUpdates[@]}; do
  updateDialog $indexToDraw
 done
}

# Cleanup
cleanUp(){
 dialog_command "overlayicon:
    SF=arrow.triangle.2.circlepath.circle.fill,palette=white,white,blue,bgcolor=no
    ne"
 dialog_command "progresstext: Performing final actions"
 dialog_command "progress: "
 if [ -n "$cleanUpPolicy" ] && [ "$cleanUpPolicy" != "false" ] &&
    [ "$cleanUpPolicy" != "no" ]; then
  dialog_command "progresstext: Performing final management actions..."
  $jamfBinary policy -forceNoRecon
 fi
 # If Build Flag set, write build flag
 if [ -n "$buildFlagPath" ]; then
  # Write Status depending on completion status
  if (( $completion < $arrayLength )); then
   defaults write "$buildFlagPath" buildStatus "Failed"
  else
   defaults write "$buildFlagPath" buildStatus "Complete"
  fi
  # Write name of build if set
  if [ -n "$buildName" ]; then
   defaults write "$buildFlagPath" buildType "$buildName"
  fi
  # Write date of completeion
  buildCompleteDate=$(date '+%Y-%m-%d %H:%M:%S %z')
  buildStartDate=$(date -j -f '%s' "$startTime" '+%Y-%m-%d %H:%M:%S %z' )
  defaults write "$buildFlagPath" buildCompleteDate -date "$buildCompleteDate"
  defaults write "$buildFlagPath" buildCompleteDate -date "$buildStartDate"
 fi
 if [ -n "$cleanUpRecon" ] && [ "$cleanUpRecon" != "false" ] &&
    [ "$cleanUpRecon" != "no" ] ; then
  dialog_command "progresstext: Performing Inventory Update..."
  $jamfBinary recon
 fi
}

waitToEnd(){
 timer=0
 while (( $(($running)) > 1 )) && (( $timer < $waitToClose )); do
  running=$(ps aux | grep -c $dialogPID)
  timer=$(( $timer + 1))
  sleep 1
 done
}

countdownWarning(){
 running=$(ps aux | grep -c $dialogPID)
 countdown=5
 while (( $(($running)) > 1 )) && (( $countdown > -1 )); do
  dialog_command "progresstext: $1 in $countdown..."
  sleep 1
  countdown=$(( $countdown - 1))
  running=$(ps aux | grep -c $dialogPID)
 done
}

endDialog(){
 case $exitBehaviour in
  "Restart" | "restart")
   countdownWarning "Restarting"
   shutdown -r now
  ;;
  "Shut Down" | "shutdown")
```

```
    countdownWarning "Shutting Down"
    shutdown -h now
   ;;
   "Log Out" | "logout")
    countdownWarning "Logging Out"
    killall loginwindow
   ;;
   *)
    countdownWarning "Closing Dialog"
    launchctl load "/Library/LaunchDaemons/com.jamfsoftware.task.1.plist"
    killall caffeinate 2> /dev/null
    exit $exitCode
   ;;
  esac
}

#### Start of Script ####

## Startup
startUp

# Get Prefs
getPrefs

if [ ! -e "$dialogApp" ] && [ -n "$installDialogURL" ]; then
  curl -L "$installDialogURL" --output /tmp/swiftDialog.pkg
  installer -pkg /tmp/swiftDialog.pkg -target /
fi

if [ ! -e "$dialogApp" ]; then
  dialog_command "No SwiftDialog! Quitting"
  exit 1
fi

# Wait for the Dock to by running before launching any GUI
# Jamf policies will be started from here if required
waitToStart

# Launch the Dialog
launchDialog

# Launch the initial List
populateList
refreshDialog
checkTimeout

# Main build loop
while (( $completion < $arrayLength )) && (( $timeRunning < $timeout )); do
  # Check if Jamf is running or has completed a trigger, then start new policies if
    needed
  jamfChecker
  ## Check if app is installed
  installChecker
  ## Refresh the interface
  refreshDialog
  checkTimeout
done

# One last check of everything
jamfChecker
installChecker
refreshDialog

# Cleanup tasks
cleanUp

# Check if all items installed or if things timed out
if (( $completion < $arrayLength )); then
  finaliseBad
else
  finaliseGood
fi

# Pause for a while so the person doing the build can see the results
waitToEnd

# Close out, Quit the script or reboot, shutdown or logout.
endDialog
```

Appendix 2: Swift Dialog Example Preference

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>appearance</key>
    <dict>
        <key>dialogIcon</key>
        <string>/PATH/TO/ICON.png</string>
        <key>fullscreen</key>
        <false/>
        <key>genericIcon</key>
        <string>/System/Library/PrivateFrameworks/MobileIcons.framework/Versions/A/Resources/DefaultIcon-60@2x~iphone.png</string>
        <key>message</key>
        <string>MAIN MESSAGE TEXT</string>
        <key>title</key>
        <string>TITLE TEXT</string>
        <key>titlebanner</key>
        <dict>
            <key>bannerimage</key>
            <string>/PATH/TO/BANNER.jpg</string>
            <key>titleOnBanner</key>
            <true/>
            <key>titlecolour</key>
            <string>#000000</string>
        </dict>
    </dict>
    <key>buidFlags</key>
    <dict>
        <key>buildFlag</key>
        <dict>
            <key>buildFlagPath</key>
            <string>/Library/Preferences/COM.ORGANISATION.plist</string>
            <key>buildName</key>
            <string>BUILD</string>
        </dict>
    </dict>
    <key>cleanup</key>
    <dict>
        <key>cleanUpPolicy</key>
        <true/>
        <key>cleanUpRecon</key>
        <true/>
    </dict>
    <key>installList</key>
    <array>
        <dict>
            <key>iconPath</key>
            <string>/PATH/TO/PAGES.png</string>
            <key>label</key>
            <string>Pages</string>
            <key>path</key>
            <string>/Applications/Pages.app</string>
        </dict>
        <dict>
            <key>iconPath</key>
            <string>/PATH/TO/EXCEL.png</string>
            <key>jamfPolicy</key>
            <string>installmicrosoftexcel</string>
            <key>label</key>
            <string>Microsoft Excel</string>
            <key>path</key>
            <string>/Applications/Microsoft Excel.app</string>
        </dict>
        <dict>
            <key>iconPath</key>
            <string>/PATH/TO/WORD.png</string>
            <key>jamfPolicy</key>
            <string>installmicrosoftword</string>
            <key>label</key>
            <string>Microsoft Word</string>
            <key>path</key>
            <string>/Applications/Microsoft Word.app</string>
        </dict>
    </array>
    <key>misc</key>
    <dict>
        <key>exitAction</key>
        <string>Quit</string>
        <key>installDialogURL</key>
        <string>https://github.com/bartreardon/swiftDialog/releases/download/v2.1.0/dialog-2.1.0-4148.pkg</string>
        <key>timeout</key>
        <integer>1800</integer>
        <key>waitToClose</key>
        <integer>1800</integer>
    </dict>
</dict>
</plist>
```

Appendix 3: Swift Dialog Custom Schema


```
{
 "title": "com.rbm.swiftdialogbuild",
 "description": "Swift Dialog Build",
 "__version": "1.0",
 "type": "object",
 "options": {
  "remove_empty_properties": true
 },
 "definitions": {
  "policy_group": {
   "type": "object",
   "format": "grid",
   "options": {
    "collapsed": true,
    "disable_properties": true
   }
  }
 },
 "properties": {
  "appearance": {
   "title": "Dialog Appearance",
   "description": "Text and Icons for Dialog",
   "property_order": 10,
   "type": "object",
   "properties": {
    "title": {
     "type": "string",
     "title": "Title Text",
     "description": "Title for Dialog Box",
     "default": "Setting Up your Mac..."
    },
    "message": {
     "type": "string",
     "title": "Message",
     "description": "Text for Dialog Box",
     "default": "Getting your computer ready to use and installing apps..."
    },
    "dialogIcon": {
     "type": "string",
     "title": "Dialog Box Icon",
     "description": "Icon for Dialog Box. Path, URL or SF symbol"
    },
    "titlebanner": {
     "title": "Title Banner",
     "description": "Optional Banner settings",
     "anyOf": [
      {
       "type": "null",
       "title": "No Banner"
      },
      {
       "type": "object",
       "title": "Banner Settings",
       "properties": {
        "bannerimage": {
         "title": "Image Path",
         "description": "Path or URL. JPEG or PNG format",
         "type": "string"
        },
        "titleOnBanner": {
         "title": "Title on Banner",
         "description": "Overlay Title text over Banner",
         "type": "boolean"
        },
        "titlecolour": {
         "title": "Title Colour",
         "description": "Colour for Title Text, colour name or HEX",
         "anyOf": [
          {
           "type": "null",
           "title": "Black"
          },
          {
           "type": "string",
           "title": "Title Colour"
          }
         ]
        }
       }
      }
     ]
    },
    "fullscreen": {
     "type": "boolean",
     "title": "Fullscreen",
     "default": "false",
     "description": "Run the dialog in fullscreen to prevent user working during
                     build"
    },
    "genericIcon": {
     "type": "string",
     "title": "Generic List Icon",
```

```
      "description": "Path or URL to use for install items without an icon
                      specified",
      "default": "/System/Library/PrivateFrameworks/MobileIcons.framework/
                  Versions/A/Resources/
                  DefaultIcon-60@2x~iphone.png"
     }
    }
   },
  "installList": {
   "title": "List of Items to be Installed",
   "description": "Installed items to show in the Dialog",
   "property_order": 20,
   "type": "array",
   "items": {
    "type": "object",
    "properties": {
     "label": {
      "title": "Name",
      "description": "Item name as displayed on list",
      "type": "string",
      "default": "Application"
     },
     "path": {
      "title": "Completion Path",
      "description": "Path to validate item has completed",
      "type": "string",
      "default": "/Applications/Application.app"
     },
     "jamfPolicy": {
      "title": "Item is installed by Jamf Policy",
      "description": "Custom Trigger for Policy",
      "default": "installApp",
      "anyOf": [
       {
        "type": "null",
        "title": "Item installed via Jamf Catalog or Mac App Store"
       },
       {
        "title": "Item Installed with Custom Policy Trigger",
        "type": "string"
       }
      ]
     },
     "iconPath": {
      "title": "Icon Path (Optional)",
      "description": "File Path or URL for icon to display in list",
      "default": "/Library/Management/Icons/App.png",
      "anyOf": [
       {
        "type": "null",
        "title": "Use Default Icon"
       },
       {
        "title": "Path to Icon",
        "type": "string"
       }
      ]
     }
    }
   }
  },
  "misc": {
   "title": "Misc Options",
   "description": "Additional Configuration",
   "property_order": 30,
   "type": "object",
   "properties": {
    "exitAction": {
     "type": "string",
     "title": "Exit Action",
     "description": "Action to Perform after Build Process completes",
     "default": "Quit",
     "enum": [
      "Quit",
      "Restart",
      "Shut Down",
      "Log Out"
     ]
    },
    "timeout": {
     "type": "integer",
     "title": "TimeOut",
     "description": "Maximum Time in Seconds before timing out",
     "default": 1800
    },
    "waitToClose": {
     "type": "integer",
     "title": "Wait before automatically closing",
     "description": "Time in Seconds that build dialog will stay open after build
                     completes before exit action",
     "default": 60
    },
    "installDialogURL": {
     "title": "SwiftDialog Pkg URL",
     "description": "URL to download and install SwiftDialog if missing",
```

```
      "default": "https://github.com/bartreardon/swiftDialog/releases/download/
                                  v2.1.0/dialog-2.1.0-4148.pkg",
      "type": "string"
     }
    }
   },
   "cleanup": {
    "title": "Cleanup",
    "description": "Optional Clean-up settings",
    "property_order": 50,
    "type": "object",
    "properties": {
     "cleanUpPolicy": {
      "title": "Run Policy at Cleanup",
      "description": "Perform any outstanding Jamf Policies due at checkin
                                  during the cleanup phase",
      "type": "boolean",
      "default": true
     },
     "cleanUpRecon": {
      "title": "Update Inventory at Cleanup",
      "description": "Perform a final Inventory update during the cleanup phase",
      "type": "boolean",
      "default": true
     }
    }
   },
   "buidFlags": {
    "title": "Build Flags",
    "description": "Optional Build Record preference file",
    "property_order": 60,
    "type": "object",
    "properties": {
     "buildFlag": {
      "title": "Write Build Preference",
      "description": "Write a Preference file to track if build completed on client
                                  Mac",
      "anyOf": [
       {
        "type": "null",
        "title": "No Build File"
       },
       {
        "type": "object",
        "title": "Create Build File",
        "properties": {
         "buildFlagPath": {
          "title": "Preference Path",
          "description": "Path for preference file to track if build completed on
                                  client Mac",
          "type": "string",
          "default": "/Library/Preferences/com.rbm.swiftdialogbuild.plist"
         },
         "buildName": {
          "title": "Build Name",
          "description": "Name record in preference file to track which Build
                                  profile was used",
          "type": "string"
         }
        }
       }
      ]
     }
    }
   }
  }
 }
}
```

Appendix 4: Swift Dialog Scoping Extension Attribute

```
#!/bin/bash

# RBM Installed Items EA

prefFile="/Library/Managed Preferences/com.rbm.swiftdialogbuild.plist"

### Get a specific key from preferences
readPref(){
    /usr/libexec/PlistBuddy -c "Print $1" "$prefFile" 2>/dev/null
}

if [ -e "$prefFile" ]; then

    index=0
    while [ -n "$itemTitle" ] || (( $index < 1 )); do
     itemTitle="$(readPref "installList:$index:label")"
     if [ -n "$itemTitle" ]; then
          if [ -n "$itemList" ]; then
               itemList="$itemList
$itemTitle"
          else
               itemList="$itemTitle"
          fi
     fi
     index=$(( $index + 1 ))
    done

    echo “<result>$itemList</result>"

else

    echo “<result></result>"

fi
```

Appendix 5: Swift Dialog Build Status Extension Attribute

```
#!/bin/bash

# RBM Build Status EA

prefFile="/Library/Preferences/com.rbm.swiftdialogbuild.plist"

### Get a specific key from preferences
readPref(){
    /usr/libexec/PlistBuddy -c "Print $1" "$prefFile" 2>/dev/null
}

if [ -e "$prefFile" ]; then

    result=$(readPref "buildStatus")
    echo "<result>$result</result>"

else

    echo "<result></result>"

fi
```

Appendix 6: Swift Dialog Build Type Extension Attribute

```
#!/bin/bash

# RBM Build Type EA

prefFile="/Library/Preferences/com.rbm.swiftdialogbuild.plist"

### Get a specific key from preferences
readPref(){
    /usr/libexec/PlistBuddy -c "Print $1" "$prefFile" 2>/dev/null
}

if [ -e "$prefFile" ]; then

    result=$(readPref "buildType")
    echo "<result>$result</result>"

else

    echo "<result></result>"

fi
```