As you know our theme is don’t do this alone… borrowed from our friend link here,

So whether you’re looking for game loot, or bad-ass warriors to help you on your journey,

We’ve got you covered. :-P

MACOS MANAGEMENT

- Preferences
- MDM
- Built In Security
- Imaging

Lets look at overall system management

What can we control on the machines?

a lot, of course, we'll be starting with preferences and property lists, which control a lot to the user experience

How are we going to get our controls on the machines?

MDM

What do you already have?

—built in security, all the stuff thats already there waiting for you

Where do we start?

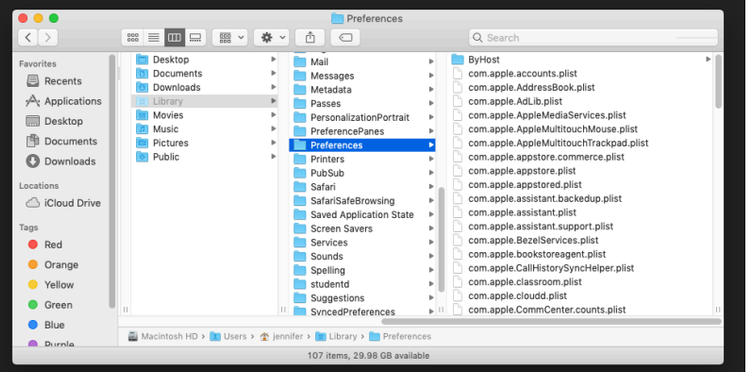
provisioning/imaging

PART ONE:
PREFERENCES

PLIST

*****.PLIST

- ~/Library/Preferences/
- /Library/Preferences
- /System/Library/Preferences/



P-list files, property list files

The property list format is just a specific use of XML - extensive markup language,
which stores several types of information (strings, numbers, Boolean values, dates, and data arrays), and is formatted with an identifier tag followed by a value for that identifier….which will make more sense in a moment when we look at them.

Property list files are often used to store a user's settings.

Where are they?

The ones we’re going to be investigating and modifying today are all User specific plists, they are located in the User folder.

If you're not familiar, the tilde (squiggle) in the path indicates that you are in the current user's folder, so /Macintosh HD/Users/$username

Your system level preferences are in Library, with some additional ones in /System / Library, which you can't modify, as they are protected by SIP, which we'll talk about around slide 57.

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

Here is an example plist, this is the com dot apple dot menu extra dot battery plist. It is controlling how the battery icon in the menubar appears.

It starts with the identifier tag, and the value for the identifier (mentioned on the last slide)

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

Lets zoom in a bit,

After the headers, we have a dictionary <dict> element, that contains our key value pair.

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

The key value pair is a general term, that includes a group of key identifiers and a set of associated values.

Here we have the key, SHOW PERCENT and the value is YES.

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

Notice that for each key or string input, you have an opening

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

and closing tag, your closing tags have the added slash before the name of the key

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

Next we have the closing tag for the dictionary

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

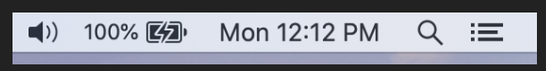
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```

and the closing tag for the whole plist. Without these tags, your file won’t be read properly.

PLIST

COM.APPLE.MENUEXTRA.BATTERY.PLIST

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ShowPercent</key>
    <string>YES</string>
</dict>
</plist>
```



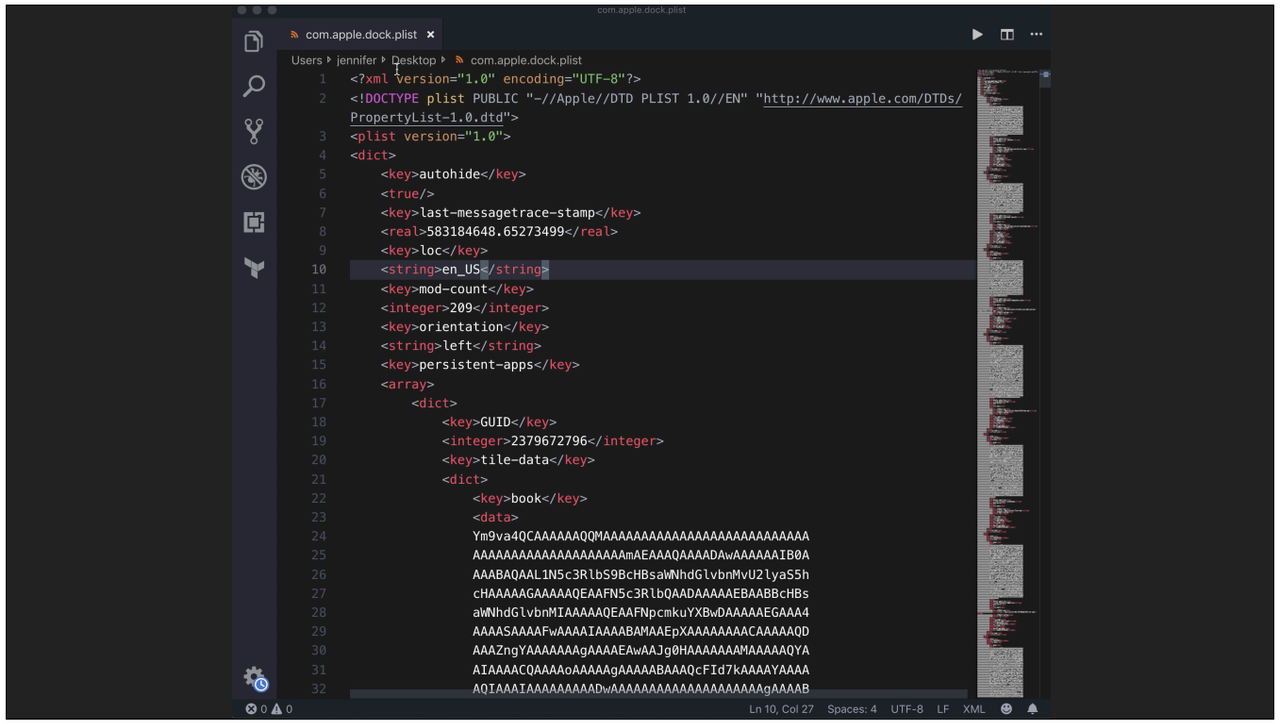
And the result of this plist — my ability to see the percentage of battery remaining in the menu bar.

```
<?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>autohide</key>
    <true/>
    <key>last-messagetrace-stamp</key>
    <real>583184648.65273499</real>
    <key>loc</key>
    <string>en_US</string>
    <key>mod-count</key>
    <integer>209</integer>
    <key>orientation</key>
    <string>left</string>
    <key>persistent-apps</key>
    <array>
        <dict>
            <key>GUID</key>
            <integer>2379672796</integer>
            <key>tile-data</key>
            <dict>
                <key>book</key>
                <data>
                Ym9va4QCAAAAAAQQMAAAAAAAAAAAAAAAAAAAAAAAAAAA
                AAAAAAAAAAAAAAAAAAAAmAEAAAQAAAADAwAAAAAIB0A
                AAABAQAAL1N5c3RlbS9BcHBsaWNhdGlvbnMvU2lyaS5h
                cHAAAAAGAAAAAQEAAFN5c3RlbQAADAAAAAEBAABBcHBs
                aWNhdGlvbnMIAAAAAQEAAFNpcmkuYXBwDAAAAAEGAAA4
                AAAASAAAAFwAAAAIAAAABAMAAEpXAAAAAAAACAAAAAQD
                AAAZngYAAAAAAAgAAAAEAwAAJg0HAAAAAAAMAAAAAQYA
```

This is the dock plist. Its rather long, as it has an array for each persistent app included in the dock, but lets focus on just the top part

We are going to look at two keys, autohide and orientation.

How are we going to make these changes?



(video showing opening file in text editor, and manually changing the values)

We can pop open a text editor and make some changes.

Maybe I want to turn off autohide so I can see the dock,

And maybe I'll move the dock from the left to the bottom…

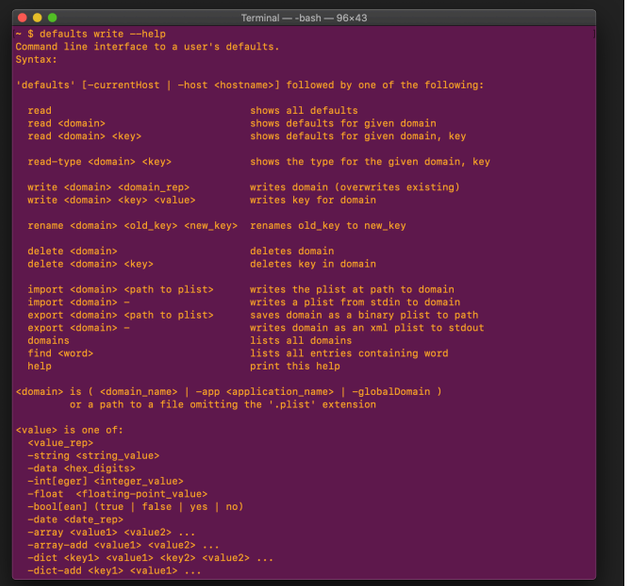
But this isn't scaleable, you're not going to every one of your machines and manually modifying text files…

Enter defaults

PLIST

DEFAULTS

- read domain
- read domain key
- write domain key ‘value’

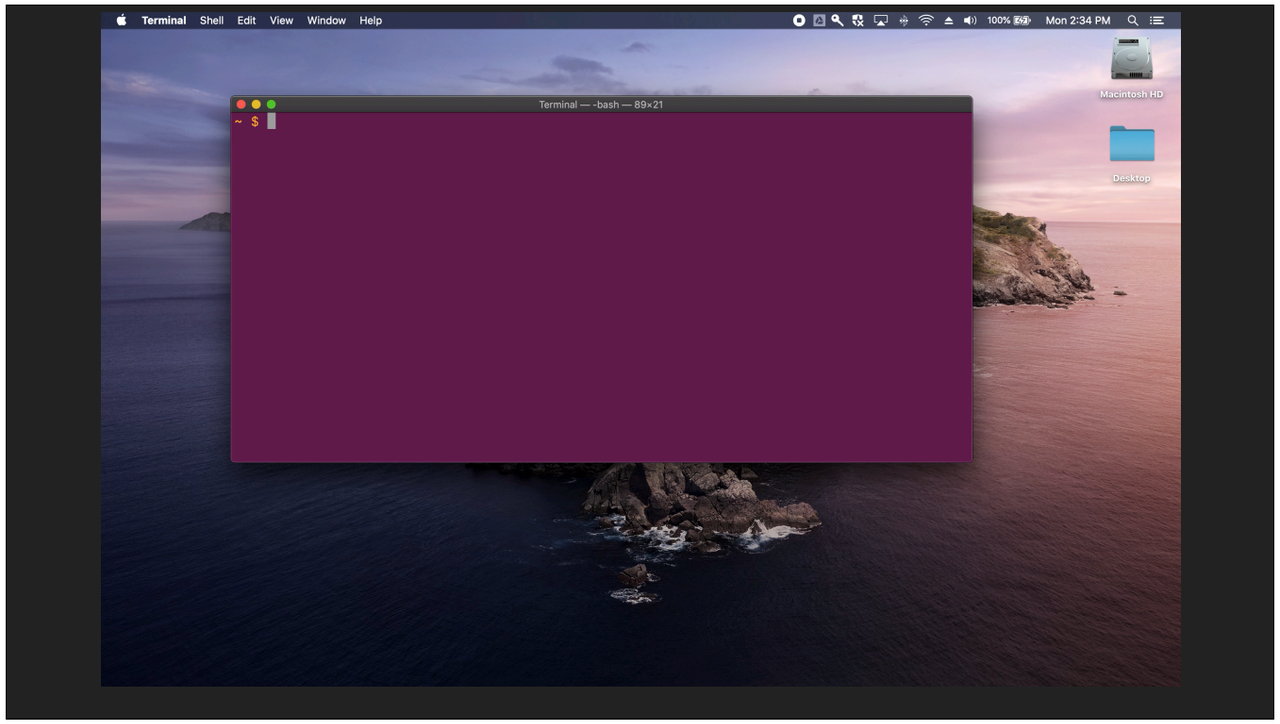


Defaults allows users to read, write, and delete Mac OS X user defaults from a command-line shell. Or you can incorporate it into a script.

This screenshot is from the help page on defaults in terminal, ‘defaults write —help’

There are two commands we’re going to work with right now, read and write.

You can read the whole domain,
The domain is the com.apple.dock part of the filename. Its underlined here to indicate that it is the part of the command you would change, you can read a specific key in the domain, and you can write a value to a key in your domain.

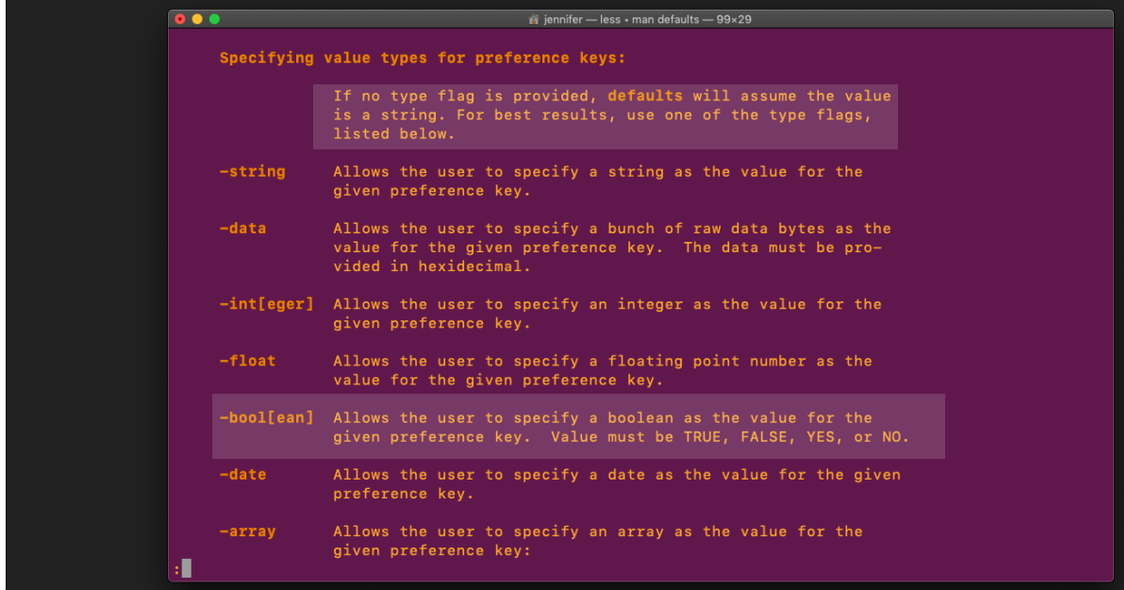


(video demo)
to start, we'll do defaults read com.apple.dock, this will read the whole file, or domain. Which is a lot of information, more than we need for this demo.

Remember on the earlier slide, I mentioned we wanted to note 'autohide' and 'orientation'?
So this time, we will specify what we want to see by adding one of those keys to the defaults read command.

Now we just get the value we want, which is 1, or TRUE.

PLIST



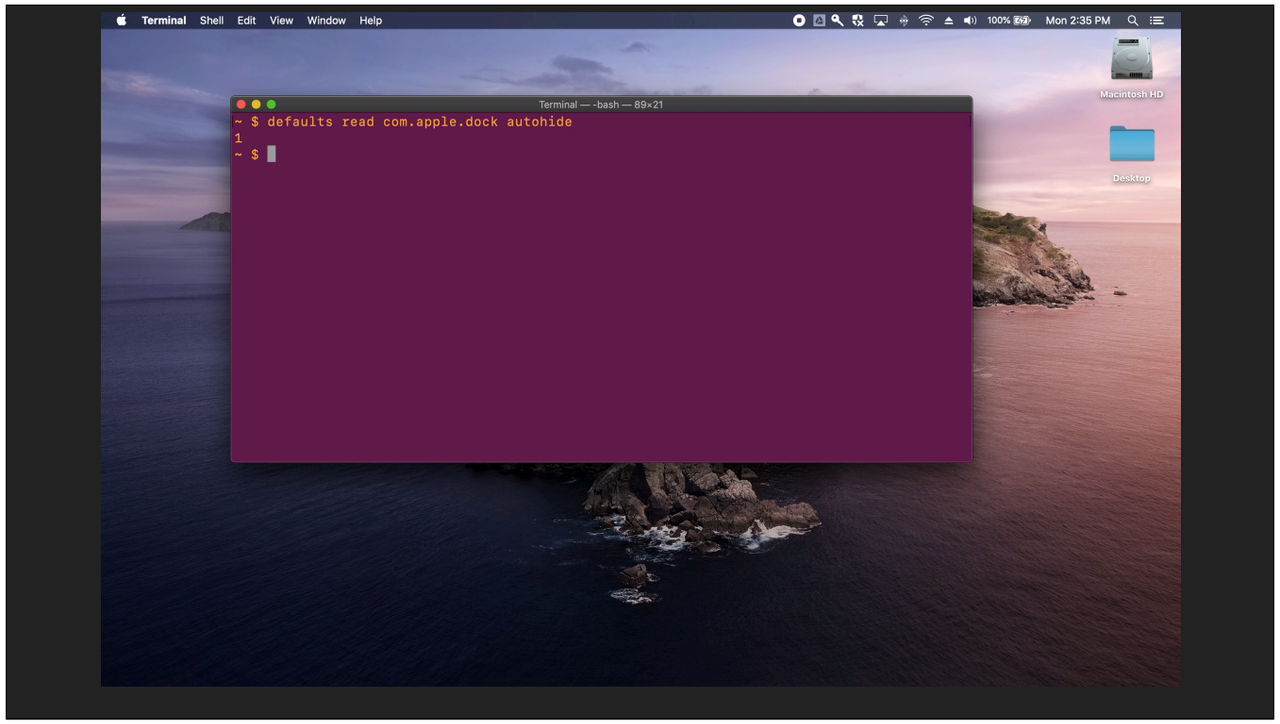
Lets pause for a moment, and look at our result. The read command returned a 1, which is a boolean value, and is equal to TRUE.

If we check out the man page for defaults, you can see that it allows you to specify a value type for a preference key.

If no flag is provided, it will assume the value is a string,

Since the read command gave a boolean value, when we go to change it, we need to make sure we return a boolean value.

The man page is telling us to add dash bool to indicate we are using a boolean value



Back to the video, to change our autohide value

we type in the defaults write command, spelling correctly of course, adding the flag for a boolean value, and type in false or zero.

If we run the read command again, you can see that the value is now zero, as false and zero are the same thing here.
But you can’t see the dock.
Since the dock was already running when we made this change, we need to relaunch it in order to have the change take effect.

And voila, the dock is no longer hidden.

Lets look at the orientation next, you can see that it is positioned at the bottom. so I’ll change it to be on the left side.

I can arrow up in my terminal history to run my previous command a second time, and verify my change has taken place.
Add another killall Dock, to relaunch the dock and show the change on the screen.

YOUR TURN

- Change save location of screenshots
  - com.apple.screencapture
  - <key>location</key>
- Show hidden files in Finder
  - com.apple.Finder
  - <key>AppleShowAllFiles</key>
  - Relaunch Finder

Your turn.

Pop open your User Library Preferences folder, and see if there is a plist you'd like to change. I recommend going for something visual, and non destructive, like the dock example.

If you are not sure what value to put in, start by changing it in the GUI. So, for the dock, look at the plist, then open System Preferences, make your change, and then reopen the plist and find the changes.
This will help you determine what value type is, and what keys you may want to use.

Here are two suggestions—resist the temptation to google.

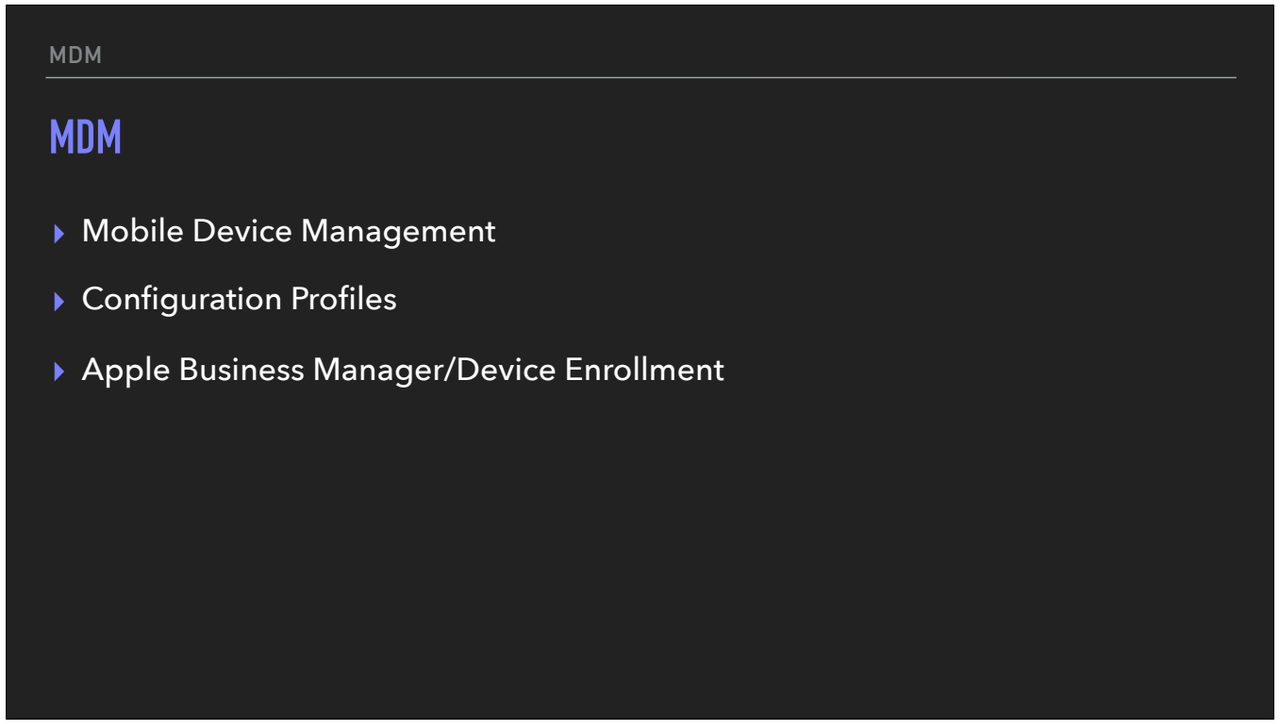
Here are the domains you'll want. I'll give you a few minutes to locate the key that will help you

SUMMARY

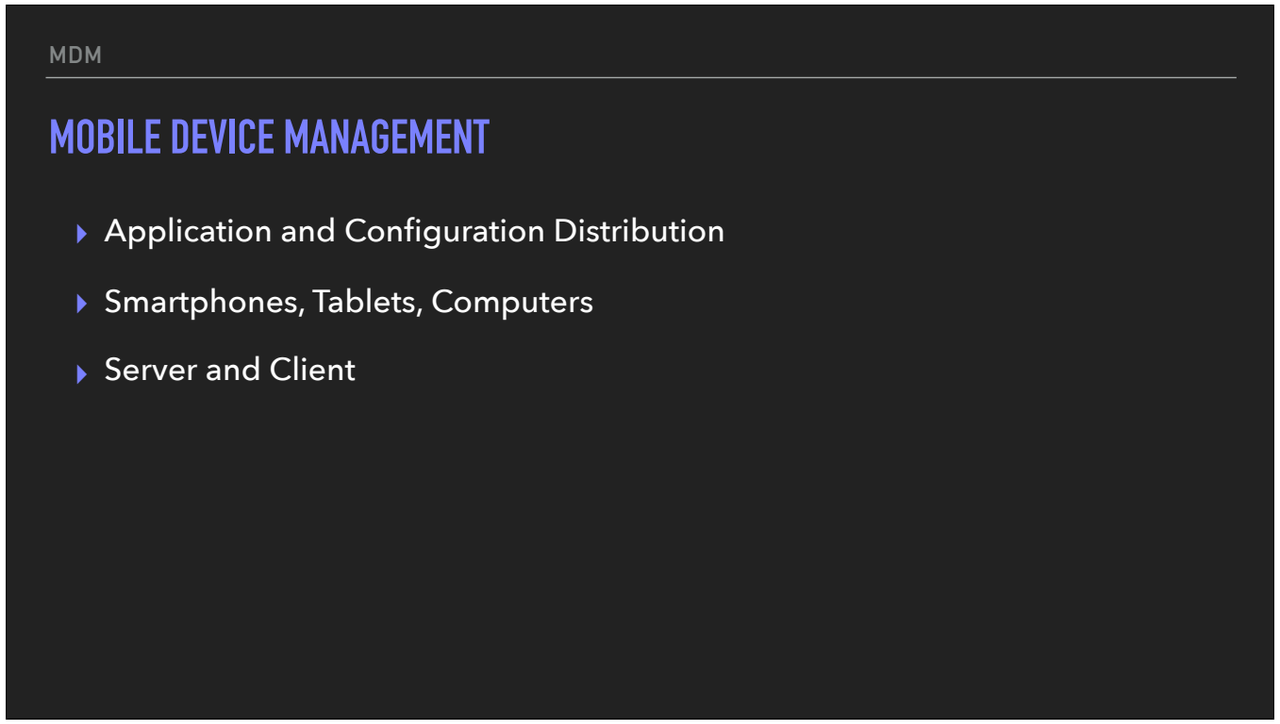
WHAT DID WE LEARN?

- What language is a plist written in?
  - XML
- Where are plists generally stored?
  - ~/Library/Preferences
  - /Library/Preferences
  - /System/Library/Preferences

PART TWO:
MDM



MDM stands for Mobile device management, which can cover a broad range of things. We’re going to look at the basics of MDM, Configuration Profiles and Apple Business Manager/ Device Enrollment



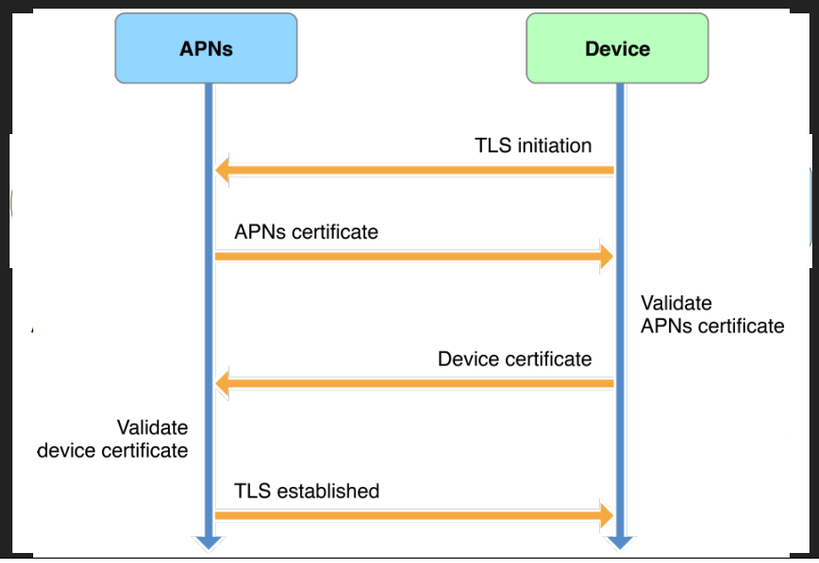
Th purpose of MDM is to simplify and enhance it management of end user devices.

MDM can include, distribution of applications and configurations
Can be used on various types of phones, tablets and computers

Typically MDM solutions include a sever component, to send the management commands to the mobile devices and a client component to run on the managed device and implement management commands.

MDM

APPLE PUSH NOTIFICATION SERVICE (APNS)



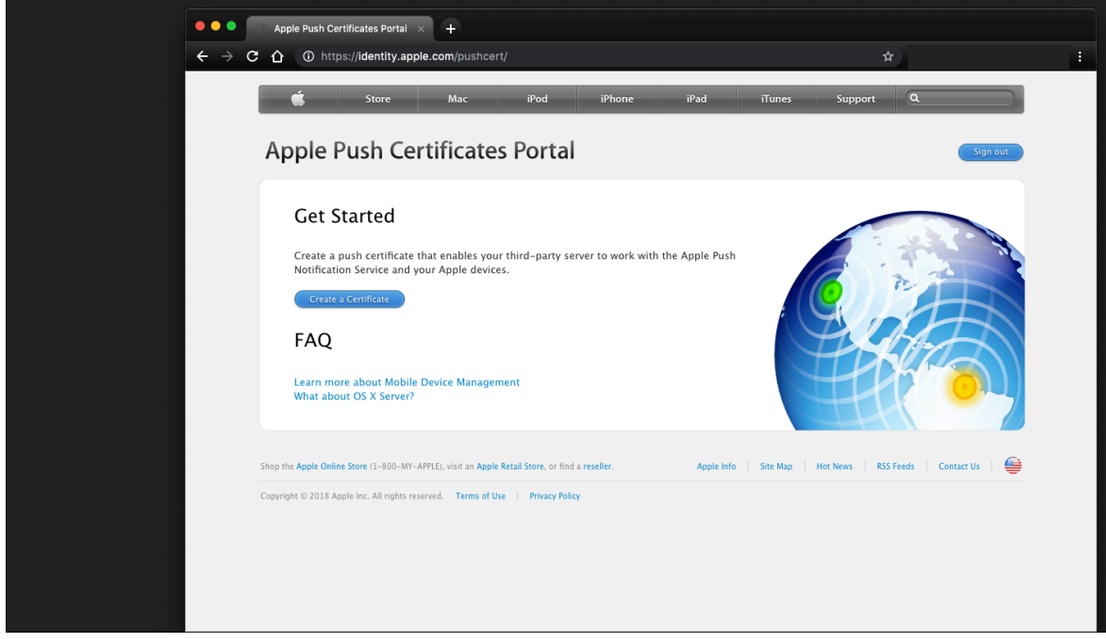
To enable the mdm traffic and communication between devices we have Apple Push notification service.

When a command is entered, the sever sends a message to the APNS saying it has an MDM profile or command awaiting delivery to an enrolled device.

Mac computers and iOS devices maintain a persistent connection to APNs when connected to a network so they will receive new notifications quickly.

When you setup your mdm service, Trust is established between the mdm provider and APNs, using certificates and a TLS connection

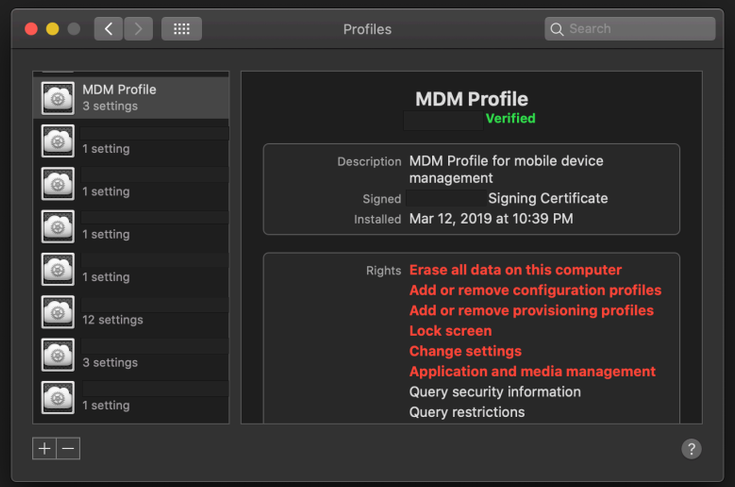
and between APNs and the device, again with the TLS initiation and validation of certificates on both sides.



To get started with Push notifications, you can go to identity.apple.com/pushcert

CONFIGURATION PROFILES

- .mobileconfig
  - plist
- Over-the-air distribution
- Encryption
- Automate configurations
- Mass Deployment



Configuration profiles are plist formatted files that allow you to distribute confirmation information.

There are multiple methods of delivery, the most likely in our cases is from a Mobile Device Management Server.

Both iOS and macOS support using encryption to protect the contents of profiles, they can also be signed to guarantee data integrity.

Many configuration profiles are going to modify your plist files similarly to what we did with the defaults command. But unlike defaults, configuration profiles can be automated and mass deployed.

PROFILE KEYS

```
<key>PayloadDescription</key>
<string>Connect to our Wifi Network</string>
<key>PayloadDisplayName</key>
<string>My Fake Company Wifi</string>
<key>PayloadIdentifier</key>
<string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-c378-721898ac9c4e.alacarte</string>
<key>PayloadOrganization</key>
<string>My Fake Company</string>
<key>PayloadRemovalDisallowed</key>
<false/>
<key>PayloadScope</key>
<string>System</string>
<key>PayloadType</key>
<string>Configuration</string>
<key>PayloadUUID</key>
<string>6339b9a0-79bd-0137-c378-721898ac9c4e</string>
<key>PayloadVersion</key>
<integer>1</integer>
```

Lets take a look at an example profile, I’m just going to hit the highlights here, because its not likely you’ll be writing these by hand. There are tools to create profiles, this part is so you have a general understanding of the content.

There are a series of keys that are common to all payloads, depending on the source of your profile, they may appear in a different order, but that doesn’t matter, as long as they are present.

PROFILE KEYS

```
<key>PayloadDescription</key>
<string>Connect to our Wifi Network
<key>PayloadDisplayName</key>
<string>My Fake Company Wifi</strin
<key>PayloadIdentifier</key>
<string>com.apple.mdm.f01898ac9c4e.
<key>PayloadOrganization</key>
<string>My Fake Company</string>
<key>PayloadRemovalDisallowed</key>
<false/>
<key>PayloadScope</key>
<string>System</string>
<key>PayloadType</key>
<string>Configuration</string>
<key>PayloadUUID</key>
<string>6339b9a0-79bd-0137-c378-721
<key>PayloadVersion</key>
<integer>1</integer>
```

Here we have our payload description, and I’ve added a screenshot from the Profiles pane in system preferences, to show how it appears to the user.

PROFILE KEYS

```
<key>PayloadDescription</key>
<string>Connect to our Wifi Network</string>
<key>PayloadDisplayName</key>
<string>My Fake Company Wifi</string>
<key>PayloadIdentifier</key>
<string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-c378-721898ac9c4e.alacarte</string>
<key>PayloadOrganization</key>
<string>My Fake Company</string>
<key>PayloadRemovalDisallowed</key>
<false/>
<key>PayloadScope</key>
<string>System</string>
<key>PayloadType</key>
<string>Configuration</string>
<key>PayloadUUID</key>
<string>6339b9a0-79bd-0137-c378-721898ac9c4e</string>
<key>PayloadVersion</key>
<integer>1</integer>
```

Next we have our display name

CONFIGURATION PROFILES

PROFILE KEYS

```
<key>PayloadDescription</key>
<string>Connect to our Wifi Network<
<key>PayloadDisplayName</key>
<string>My Fake Company Wifi</strin
<key>PayloadIdentifier</key>
<string>com.apple.mdm.f01898ac9c4e.(
<key>PayloadOrganization</key>
<string>My Fake Company</string>
<key>PayloadRemovalDisallowed</key>
<false/>
<key>PayloadScope</key>
<string>System</string>
<key>PayloadType</key>
<string>Configuration</string>
<key>PayloadUUID</key>
<string>6339b9a0-79bd-0137-c378-7218
<key>PayloadVersion</key>
<integer>1</integer>
```

And its corresponding location

CONFIGURATION PROFILES

PROFILE KEYS

```
<key>PayloadDescription</key>
<string>Connect to our Wifi Network<
<key>PayloadDisplayName</key>
<string>My Fake Company Wifi</strin
<key>PayloadIdentifier</key>
<string>com.apple.mdm.f01898ac9c4e.(
<key>PayloadOrganization</key>
<string>My Fake Company</string>
<key>PayloadRemovalDisallowed</key>
<false/>
<key>PayloadScope</key>
<string>System</string>
<key>PayloadType</key>
<string>Configuration</string>
<key>PayloadUUID</key>
<string>6339b9a0-79bd-0137-c378-7218
<key>PayloadVersion</key>
<integer>1</integer>
```

And finally, the payload removal option. You can see disallowed is false, therefore the minus option in system preferences is greyed out

```
<?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>PayloadContent</key>
    <array>
        <dict>
            <key>AuthenticationMethod</key>
            <string></string>
            <key>AutoJoin</key>
            <true/>
            <key>CaptiveBypass</key>
            <false/>
            <key>EncryptionType</key>
            <string>None</string>
            <key>HIDDEN_NETWORK</key>
            <false/>
            <key>Interface</key>
            <string>BuiltInWireless</string>
            <key>IsHotspot</key>
            <false/>
            <key>PayloadDisplayName</key>
            <string>WiFi (My Fake Company Wifi)</string>
            <key>PayloadEnabled</key>
            <true/>
            <key>PayloadIdentifier</key>
            <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-c378-721898ac9c4e.alacarte.interfaces.a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
            <key>PayloadType</key>
            <string>com.apple.wifi.managed</string>
            <key>PayloadUUID</key>
            <string>a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
            <key>PayloadVersion</key>
            <integer>1</integer>
            <key>ProxyType</key>
            <string>None</string>
            <key>SSID_STR</key>
            <string>My Fake Company Wifi</string>
            <key>SetupModes</key>
            <array/>
        </dict>
    </array>
    <key>PayloadDescription</key>
    <string>Connect to our Wifi</string>
    <key>PayloadDisplayName</key>
    <string>My Fake Company Wifi</string>
    <key>PayloadIdentifier</key>
    <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-c378-721898ac9c4e.alacarte</string>
    <key>PayloadOrganization</key>
    <string>My Fake Company</string>
    <key>PayloadRemovalDisallowed</key>
    <false/>
```

This is an example profile, its actually a rather short one, but you can see I've not completely squeezed it on there. But regardless of length, they are easy to read, with only one key or value per line.
This sample profile tells the computer to connect to My Fake Company's wifi network, helpfully titled "My Fake Company Wifi"

There are a lot of payload specific keys. Seriously. In apples Configuration profiles reference pdf, pages 8 through 118 are the listings/explanation of the keys.

I have a link in the resources section to a nice GitHub that is a little easier to read and navigate than the PDF, if you want to take a look at all of the available options.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dt
<plist version="1.0">
<dict>
    <key>PayloadContent</key>
    <array>
        <dict>
            <key>AuthenticationMethod</key>
            <string></string>
            <key>AutoJoin</key>
            <true/>
            <key>CaptiveBypass</key>
            <false/>
            <key>EncryptionType</key>
            <string>None</string>
            <key>HIDDEN_NETWORK</key>
            <false/>
            <key>Interface</key>
            <string>BuiltInWireless</string>
            <key>IsHotspot</key>
            <false/>
            <key>PayloadDisplayName</key>
            <string>WiFi (My Fake Company Wifi)</string>
            <key>PayloadEnabled</key>
            <true/>
            <key>PayloadIdentifier</key>
            <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-
c378-721898ac9c4e.alacarte.interfaces.a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
```

if we zoom in on the sample configuration profile, we can see its components.

to start, we have the xml file designations

Followed by our DICT, and our individual keys. This example, unlike the last, has payload content included.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dt
<plist version="1.0">
<dict>
    <key>PayloadContent</key>
    <array>
        <dict>
            <key>AuthenticationMethod</key>
            <string></string>
            <key>AutoJoin</key>
            <true/>
            <key>CaptiveBypass</key>
            <false/>
            <key>EncryptionType</key>
            <string>None</string>
            <key>HIDDEN_NETWORK</key>
            <false/>
            <key>Interface</key>
            <string>BuiltInWireless</string>
            <key>IsHotspot</key>
            <false/>
            <key>PayloadDisplayName</key>
            <string>WiFi (My Fake Company Wifi)</string>
            <key>PayloadEnabled</key>
            <true/>
            <key>PayloadIdentifier</key>
            <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-
c378-721898ac9c4e.alacarte.interfaces.a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
```

Here we can see the wifi specifics.

And again, just the highlights, we have the encryption type, none

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dt
<plist version="1.0">
<dict>
    <key>PayloadContent</key>
    <array>
        <dict>
            <key>AuthenticationMethod</key>
            <string></string>
            <key>AutoJoin</key>
            <true/>
            <key>CaptiveBypass</key>
            <false/>
            <key>EncryptionType</key>
            <string>None</string>
            <key>HIDDEN_NETWORK</key>
            <false/>
            <key>Interface</key>
            <string>BuiltInWireless</string>
            <key>IsHotspot</key>
            <false/>
            <key>PayloadDisplayName</key>
            <string>WiFi (My Fake Company Wifi)</string>
            <key>PayloadEnabled</key>
            <true/>
            <key>PayloadIdentifier</key>
            <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-
c378-721898ac9c4e.alacarte.interfaces.a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
```

Hidden network, false. Clearly network security is a top priority for Fake Company

```
            <false/>
            <key>Interface</key>
            <string>BuiltInWireless</string>
            <key>IsHotspot</key>
            <false/>
            <key>PayloadDisplayName</key>
            <string>WiFi (My Fake Company Wifi)</string>
            <key>PayloadEnabled</key>
            <true/>
            <key>PayloadIdentifier</key>
            <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-
c378-721898ac9c4e.alacarte.interfaces.a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
            <key>PayloadType</key>
            <string>com.apple.wifi.managed</string>
            <key>PayloadUUID</key>
            <string>a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
            <key>PayloadVersion</key>
            <integer>1</integer>
            <key>ProxyType</key>
            <string>None</string>
            <key>SSID_STR</key>
            <string>My Fake Company Wifi</string>
            <key>SetupModes</key>
            <array/>
        </dict>
    </array>
    <key>PayloadDescription</key>
    <string>Connect to our Wifi</string>
    <key>PayloadDisplayName</key>
```

Moving down a bit, we can see the SSID

```
            <false/>
            <key>Interface</key>
            <string>BuiltInWireless</string>
            <key>IsHotspot</key>
            <false/>
            <key>PayloadDisplayName</key>
            <string>WiFi (My Fake Company Wifi)</string>
            <key>PayloadEnabled</key>
            <true/>
            <key>PayloadIdentifier</key>
            <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-
c378-721898ac9c4e.alacarte.interfaces.a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
            <key>PayloadType</key>
            <string>com.apple.wifi.managed</string>
            <key>PayloadUUID</key>
            <string>a7e8e3a0-79bd-0137-c379-721898ac9c4e</string>
            <key>PayloadVersion</key>
            <integer>1</integer>
            <key>ProxyType</key>
            <string>None</string>
            <key>SSID_STR</key>
            <string>My Fake Company Wifi</string>
            <key>SetupModes</key>
            <array/>
        </dict>
    </array>
    <key>PayloadDescription</key>
    <string>Connect to our Wifi</string>
    <key>PayloadDisplayName</key>
```

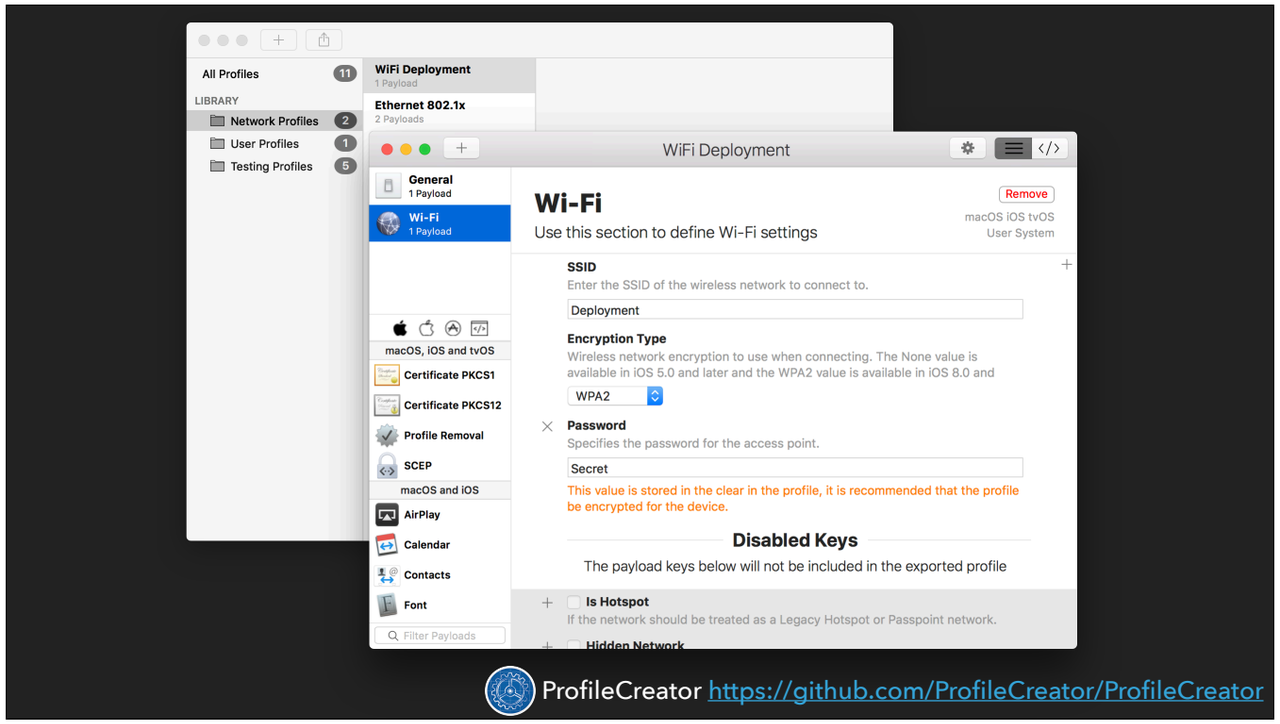
Moving down a bit, we can see the SSID

```
            <key>SSID_STR</key>
            <string>My Fake Company Wifi</string>
            <key>SetupModes</key>
            <array/>
        </dict>
    </array>
    <key>PayloadDescription</key>
    <string>Connect to our Wifi</string>
    <key>PayloadDisplayName</key>
    <string>My Fake Company Wifi</string>
    <key>PayloadIdentifier</key>
    <string>com.apple.mdm.f01898ac9c4e.6339b9a0-79bd-0137-c378-721898ac9c4e.alacarte</string>
    <key>PayloadOrganization</key>
    <string>My Fake Company</string>
    <key>PayloadRemovalDisallowed</key>
    <false/>
    <key>PayloadScope</key>
    <string>System</string>
    <key>PayloadType</key>
    <string>Configuration</string>
    <key>PayloadUUID</key>
    <string>6339b9a0-79bd-0137-c378-721898ac9c4e</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
</dict>
</plist>
```

Followed by the end of the payload content array which leads to the

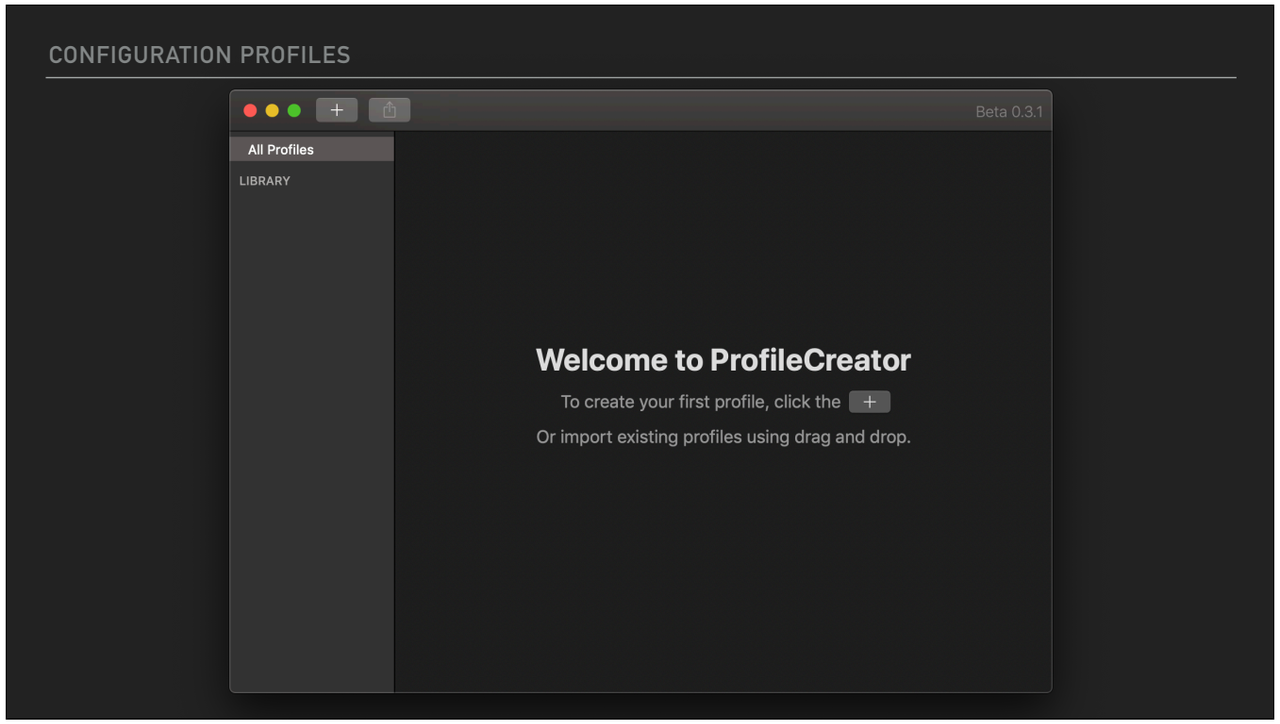
Our keys included for all payloads

again, yours may have this information in a different order, your payload content may be below your required keys, or it may not.



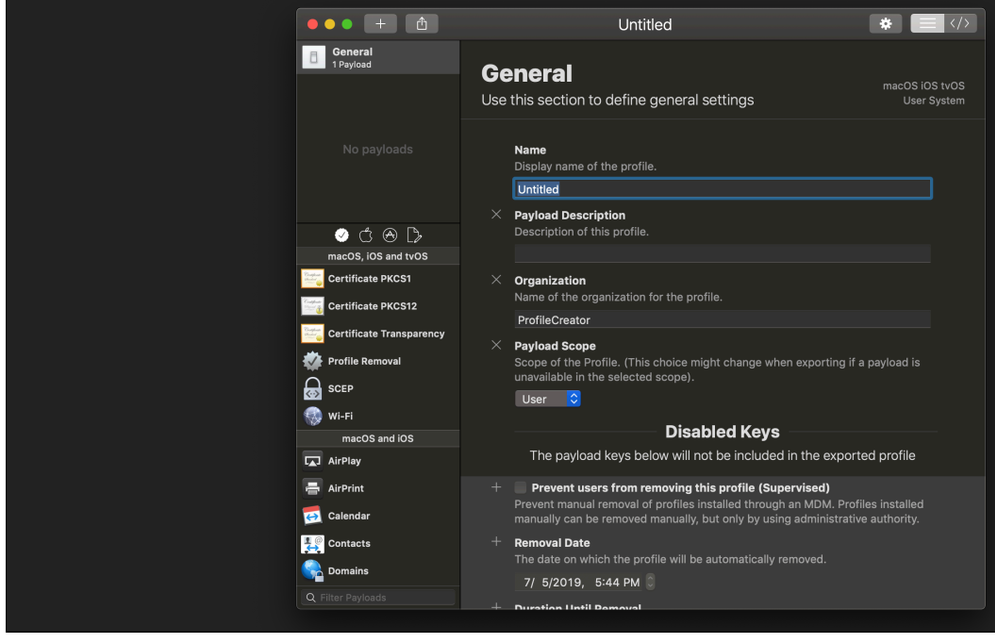
How are we really going to create configuration profiles?

The mdm solutions I've used, have built in templates and setup assistants in their admin consoles, but I want to be vendor agnostic today so I am going to show you an open source tool called Profile Creator



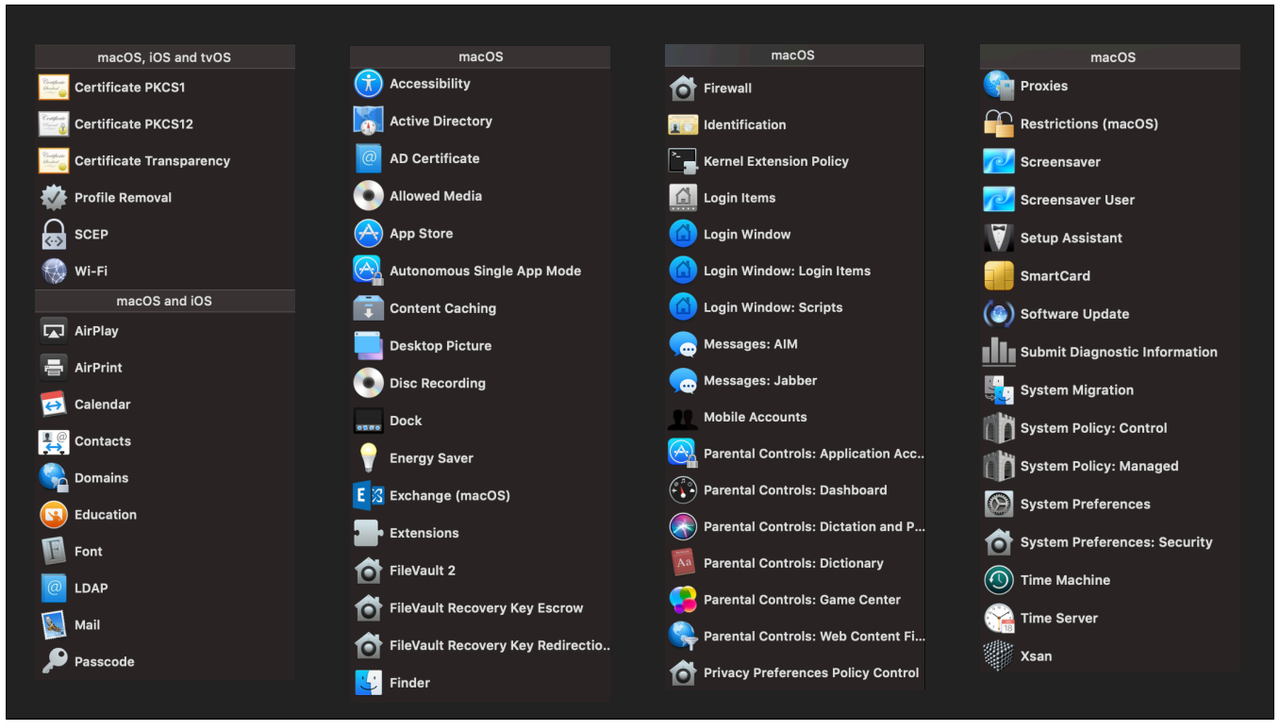
This is a very straightforward and easy to use tool.

CONFIGURATION PROFILES

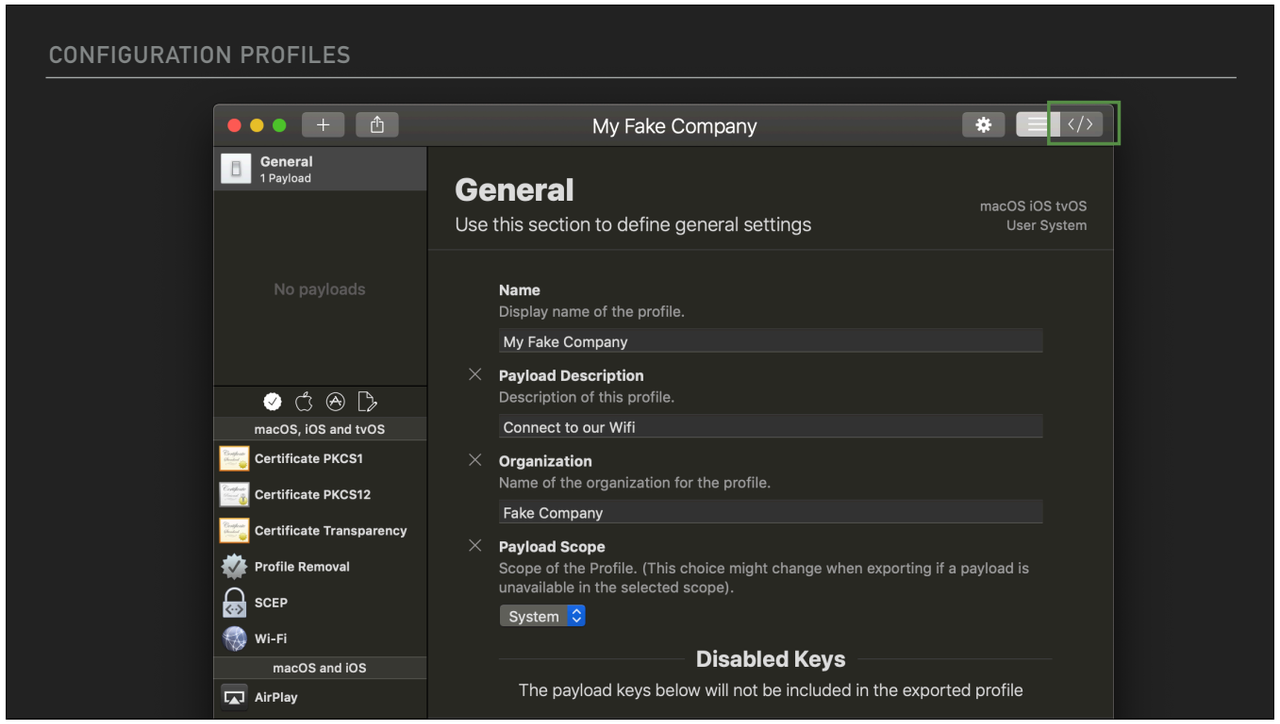


To start, you are given the opportunity to file out the general settings, these are the payload keys I spoke of before, that are required for all profiles. And the app gives you the option of including some additional keys.

If we focus on the left side of the window, you can see a list of all of the payloads you can add to your profiles.

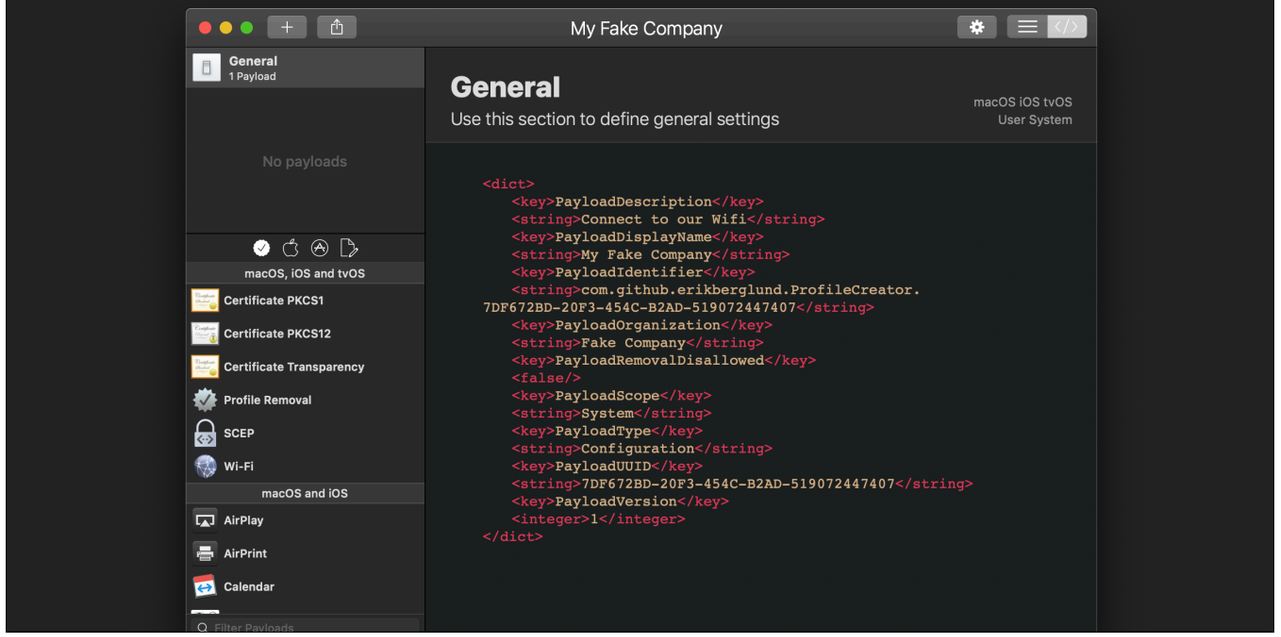


First the app lists payloads that are used on macOS, iOS and tvOS. Then just macOS and iOS, then I skipped iOS,
And then macOS, And you can see that there are quite a few options, as I mentioned earlier.

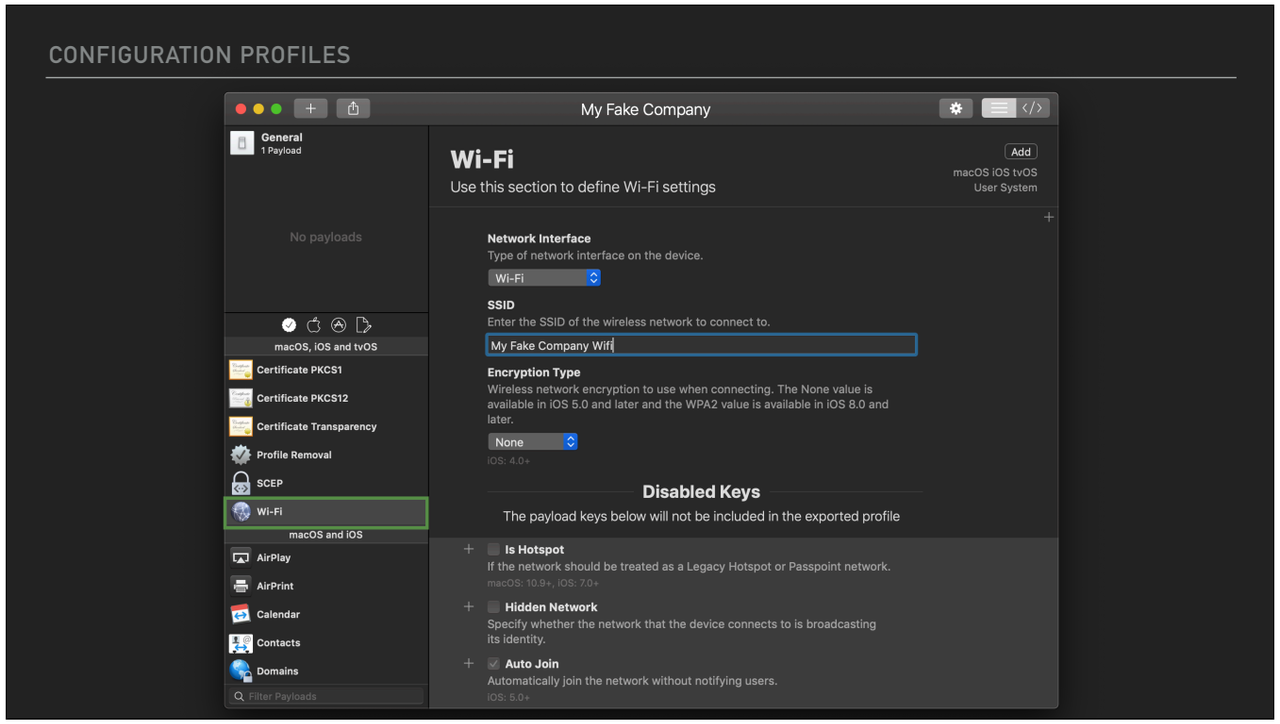


I’ve gone in and entered matching data from my earlier config profile.

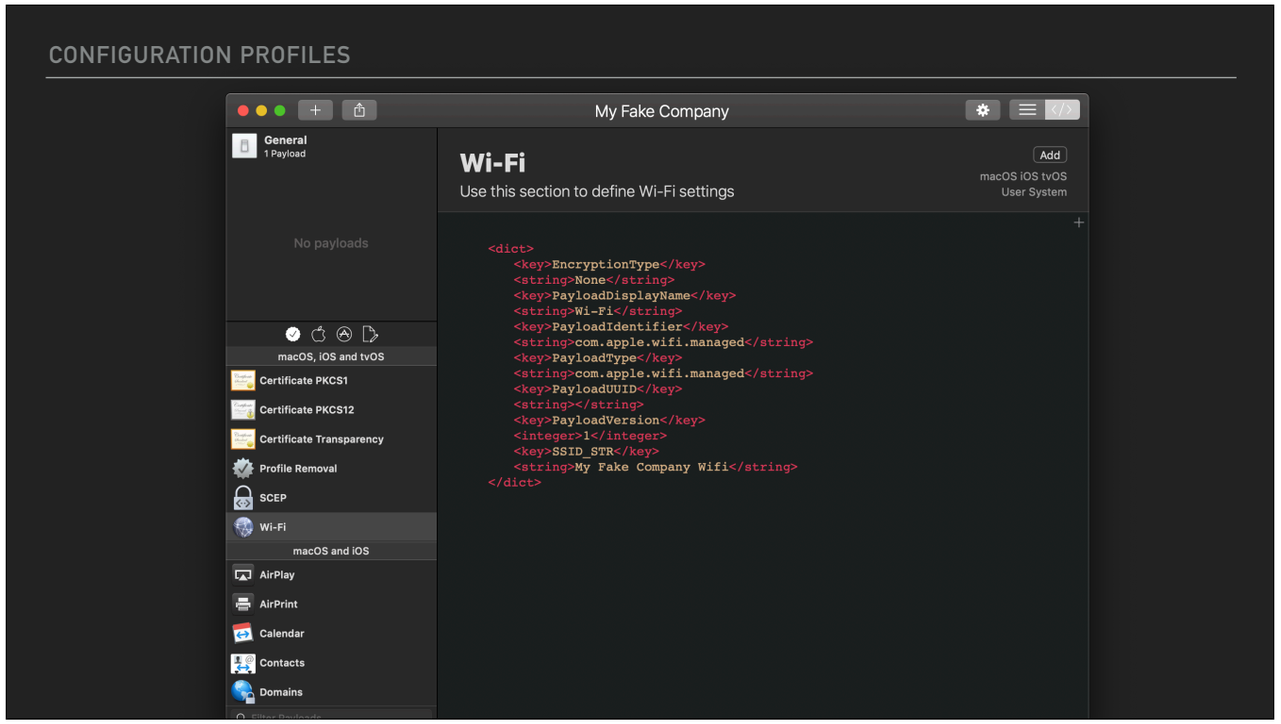
And if you click this helpful button in the right corner



The view will switch so you can see the xml you are creating

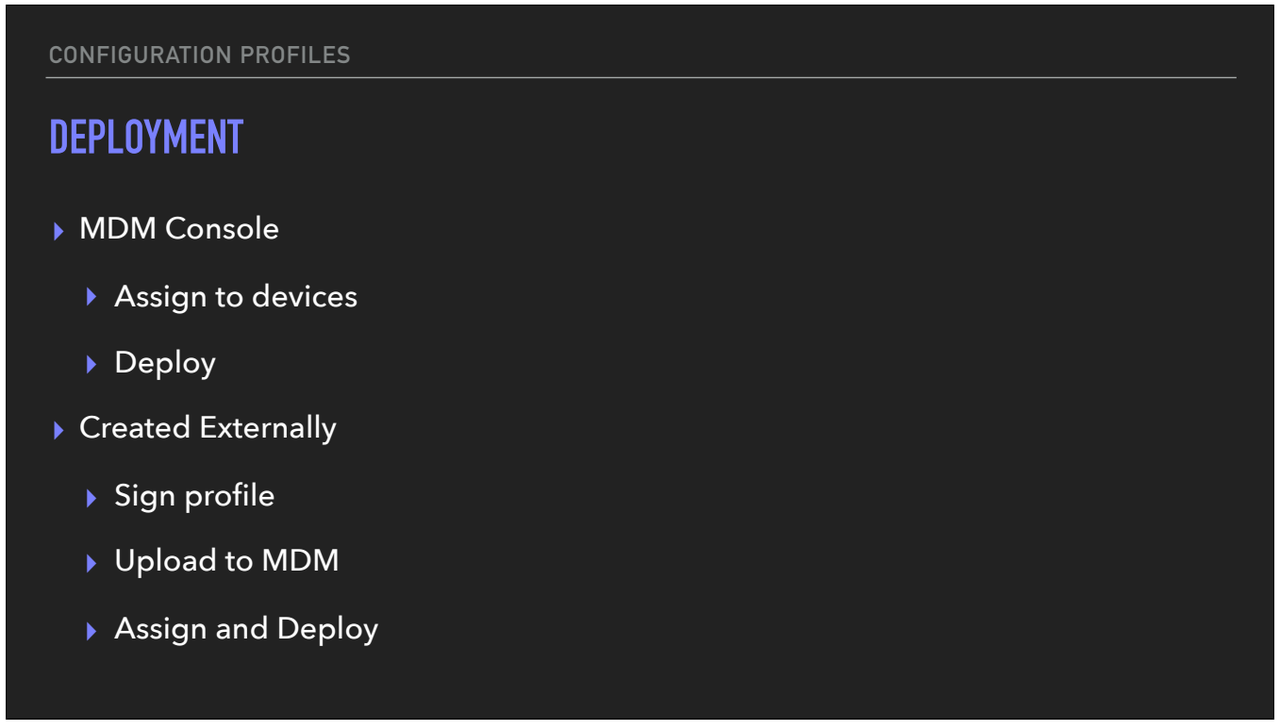


Next, I've selected the Wifi payload from the left side and begun to enter my details. The app has a long list of key options you can add to your profile, just scroll through the disabled keys section at the bottom, and click the plus icon to add them



And if we quickly pop back over to the xml view, it helpfully only shows the section you are working with, so you aren’t overwhelmed,

And it has the same keys we looked at earlier



Once you have your configuration profile, you, I assume, would like to deploy it. If you've made the profile within your MDM providers system, likely all you need to do is assign it to devices and deploy.

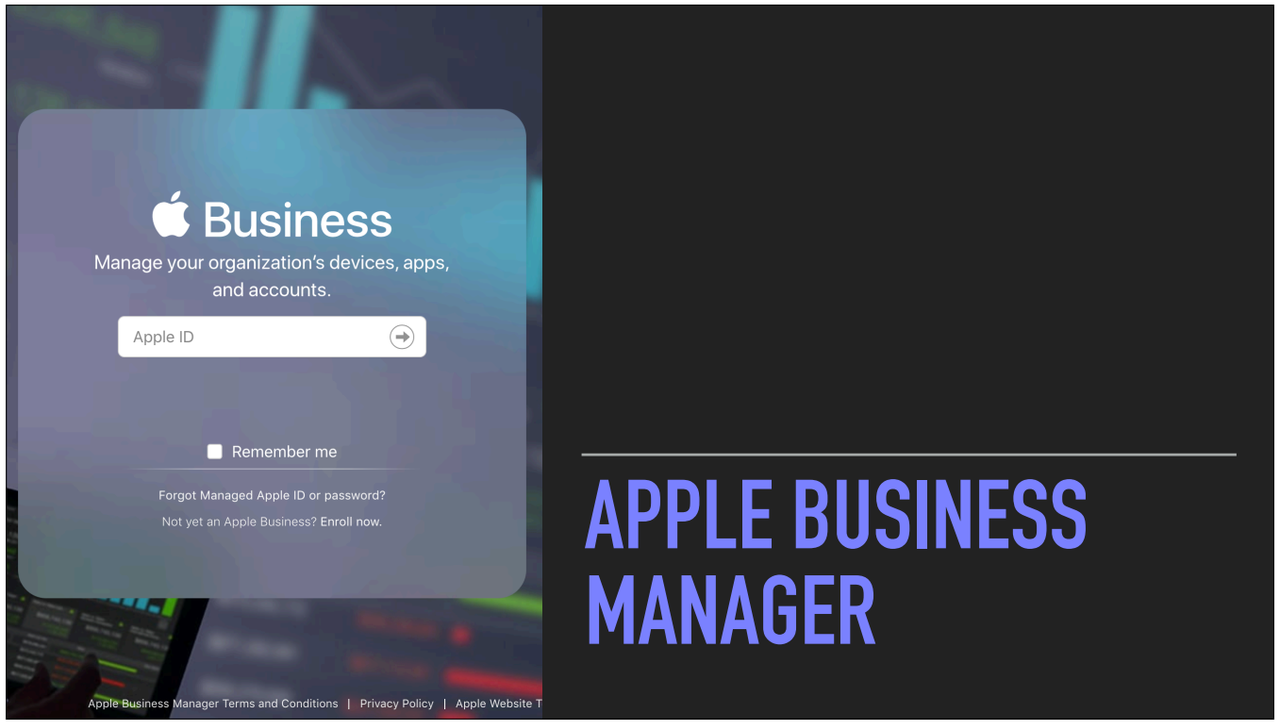
If you've made it in profile creator or another tool, I'd suggest signing it, then uploading to your MDM, assign to devices and deploy

DEPLOYMENT

- Signing your profile
  - https://www.macblog.org/post/signing-configuration-profiles/
  - https://www.amsys.co.uk/sign-configuration-profile/

Signing your profile protects the contents, so its highly recommended. In the resources section, I've got two blog posts that both give a good walk through of the process, one from macblog.org and one from Amsys.

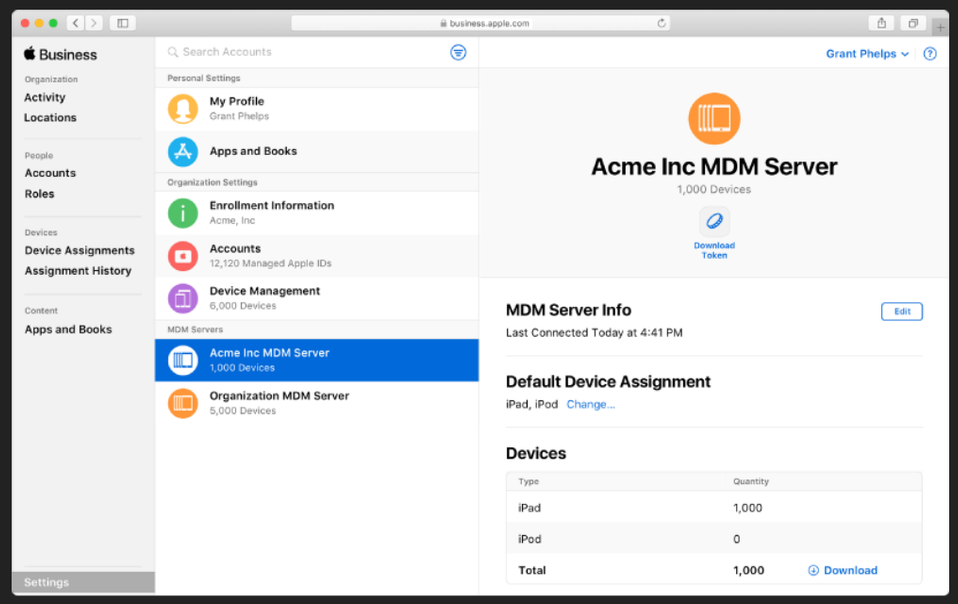
note from experience: Make sure you have a certificate with a signing identity.



I talked at the beginning of the MDM section about Apple Push Notification service traffic and establishing trust between it and your mdm provider. now that we've linked your MDM to Apple's notification system, we can move on to the next step of linking our MDM to Apple's device enrollment and volume purchase programs to your MDM

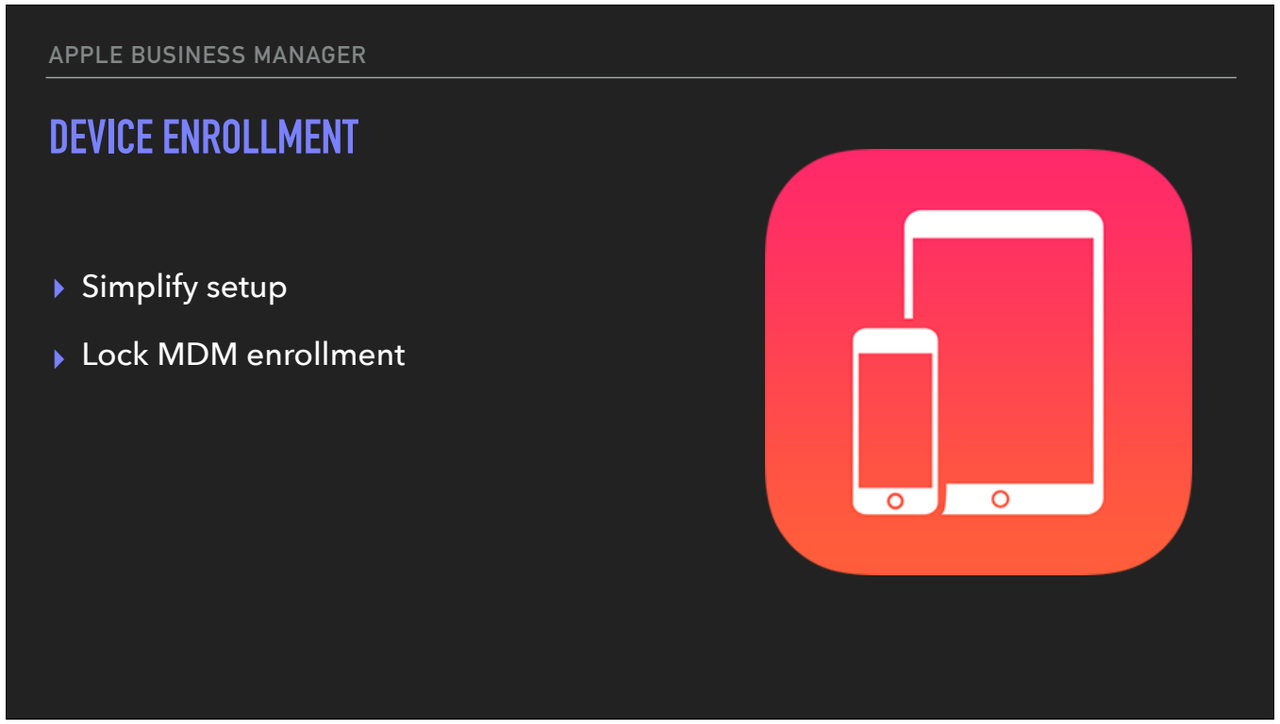
You can control your fleet with configuration profiles, via your mdm solution and stop there. Or you can continue to Apple Business Manager, which is designed to simplify the initial enrollment process. And purchased app devlivery.

ADMINISTRATION CONSOLE



Here you can connect your mdm servers, setup your office locations, assign devices, purchase apps, and view your history.

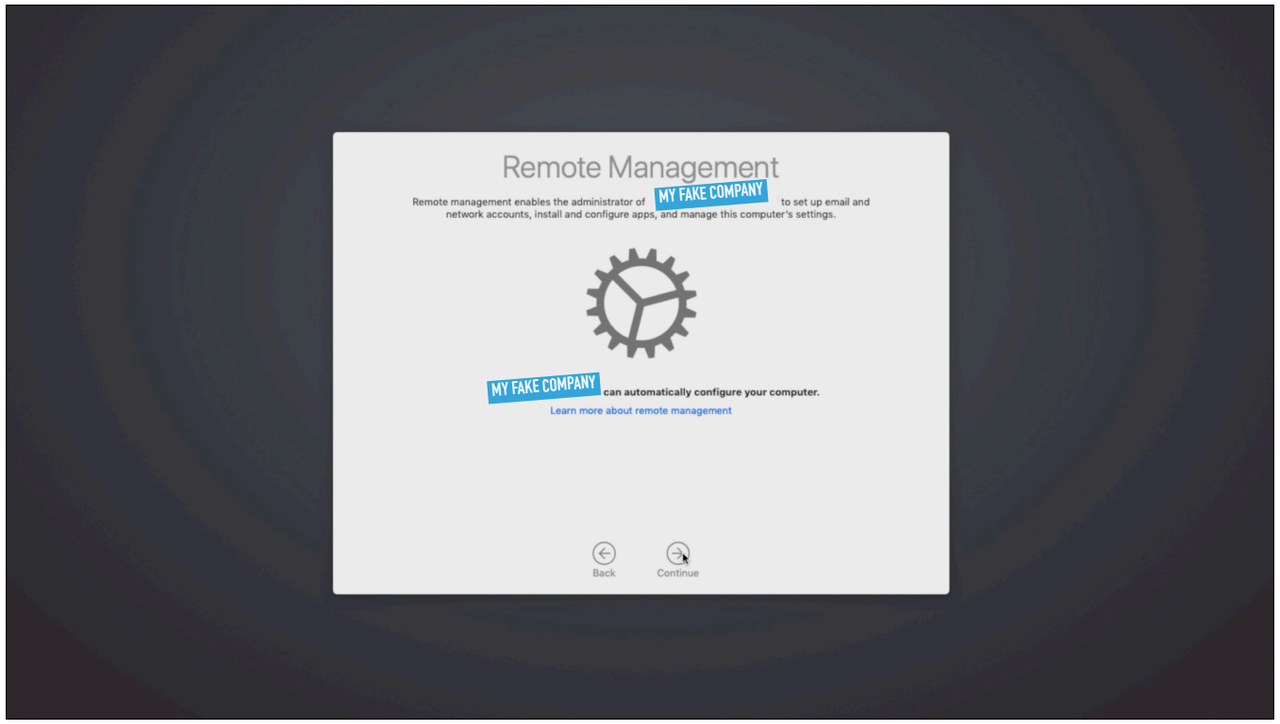
Business manager is what allows you to connect your MDM to Apple. If you set your devices to auto assign, this likely won't be a tool that you use daily. You come in to purchase apps, create new users, maybe troubleshoot, but once the connection is properly setup, you'll do the real 'work' from your MDM server.



Device Enrollment aka DEP is one aspect of ABM

Device Enrollment lets you automate Mobile Device Management (MDM) enrollment and simplify initial device setup. You can supervise devices during activation without touching them, and lock MDM enrollment for ongoing management.

For example, Device enrollment allows me to do this.



Enroll a machine to my MDM during the initial setup, within the setup assistant.

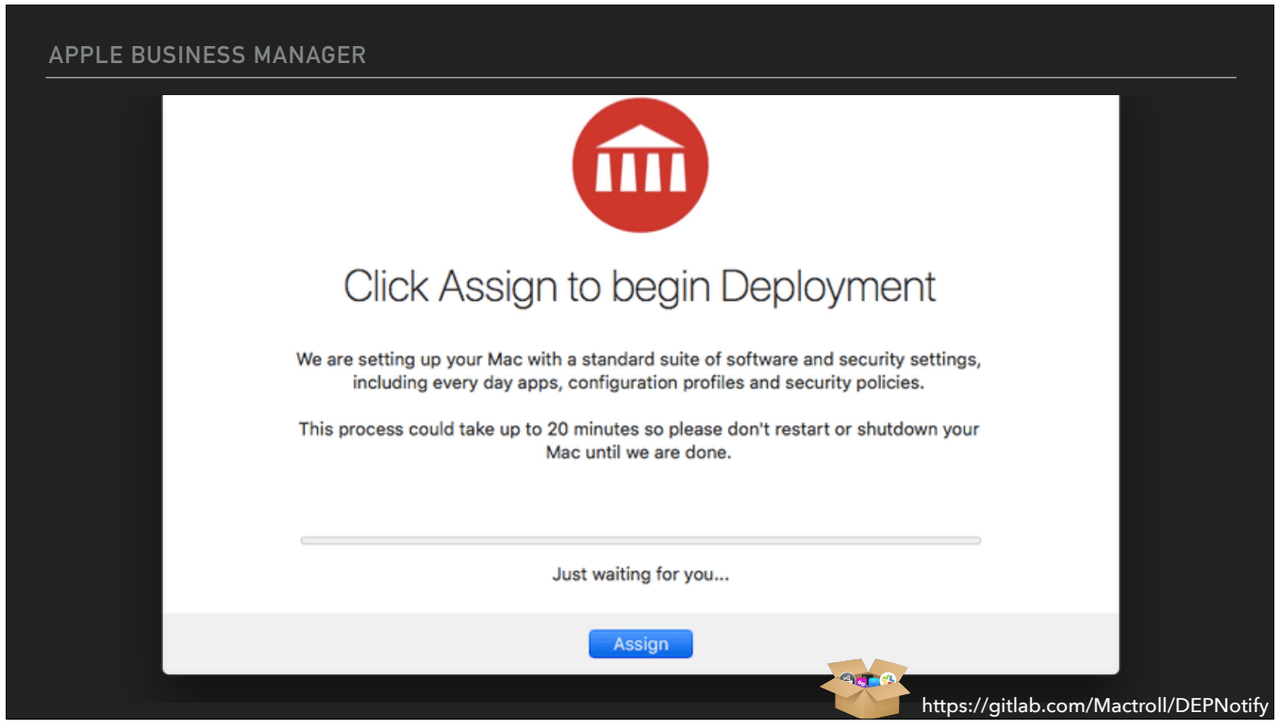
As soon as the machine is turned on, it connects to the internet, talks to apple, and apple directs it to your server and your configurations.

And, if the machine is ever wiped, this will happen all over again from setup.

After that, you can do whatever you want (within reason).

There are open source tools to help, this setup screen was made by splash buddy. splash buddy takes over the screen and shows the user your html based input while their applications install in the background.

At the bottom is the GitHub link, which you can also find in our resources section



Also available DEPNotify, vendor agnostic, as it reads from a text file, also includes MDM specific flags for filewave, jamf and munki

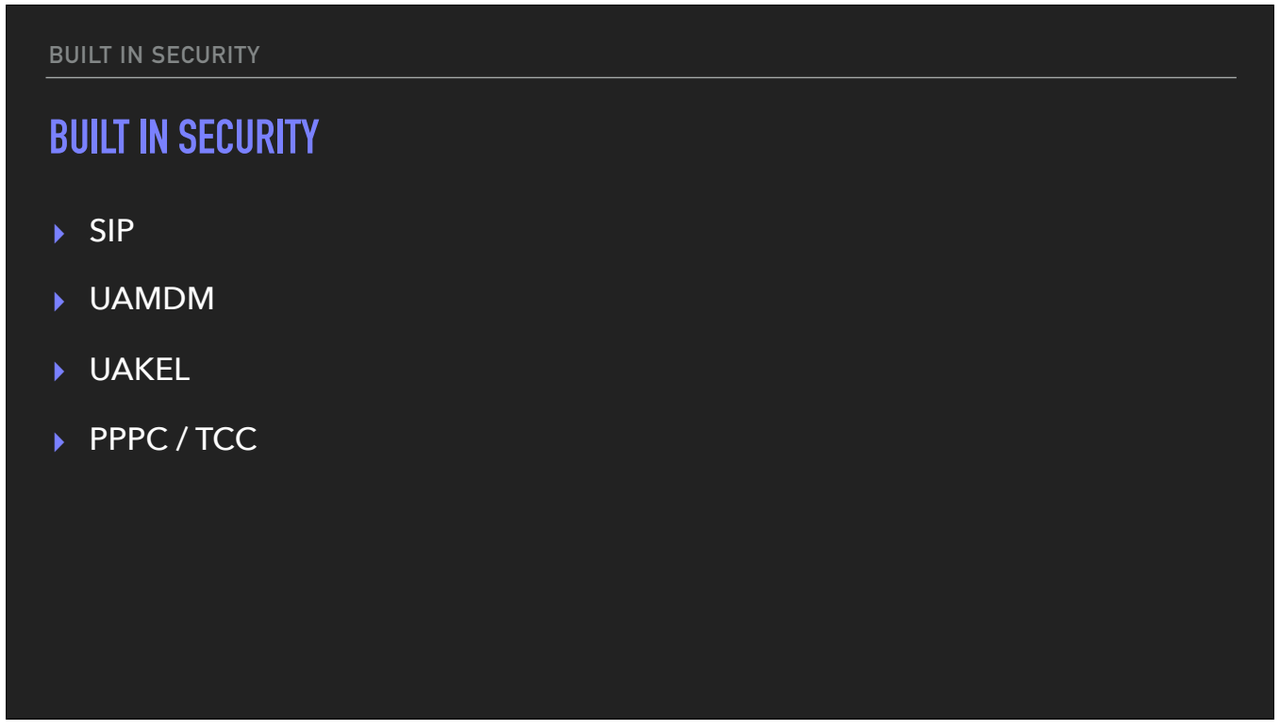
SUMMARY

WHAT DID WE LEARN?

- What file format are configuration profiles?
  - .mobileconfig
- What services allows your devices to get notifications and commands from your MDM?
  - APNs

PART THREE:
BUILT IN SECURITY

Now that you've able to configured deploy your machines, lets talk about some of the built in security. Before you start thinking about adding, well, anything, you need to know what you're starting with.



These are four important, built in, security functions of MacOS.

SIP
UAMDM
UAKEL
And
PPPC

And I promise I’ll define each of these acronyms as we go though.

BUILT IN SECURITY

SYSTEM INTEGRITY PROTECTION

- Protected Locations
  - /System
  - /usr
  - /bin
  - /sbin
  - Apps that are pre-installed with OS X

Introduced in El Capitan, SIP is designed to help prevent potentially malicious software from modifying protected files on your Mac. It restricts the root user account and limits the actions it can perform on the protected parts of the OS.

Before SIP, the root user had ZERO restrictions, so, if software obtained root-level access when you entered your Admin info to install software, that software could modify or overwrite any system file or app, potentially screwing up your machine pretty good.

Talk about the protect3ed directories

BUILT IN SECURITY

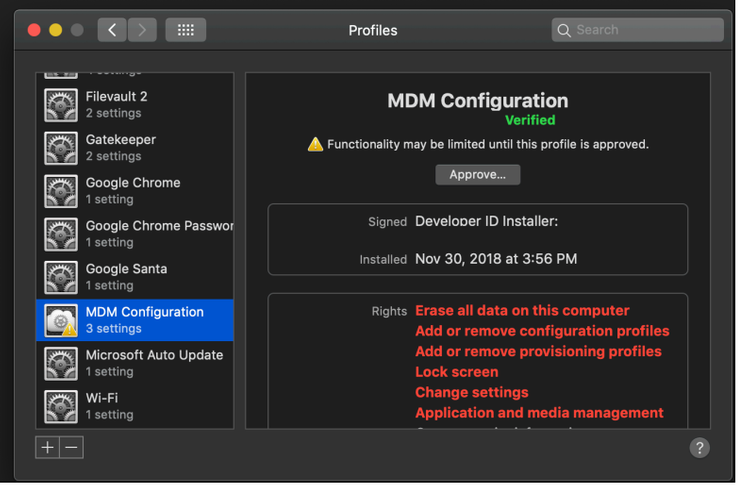
SYSTEM INTEGRITY PROTECTION

- Writeable Locations
  - /Applications
  - /Library
  - /usr/local

SIP isn’t there to inhibit developers,
/Applications, /Library and /usr/local/ are the paths that thirty part apps and installers can continue to write to. In my experience, apple advised against writing to the SIP protected locations, but applications were still doing it, so now its not just advised against, its simply prohibited.

App Store Apps all comply to SIP standards already. According to apple . “Other third-party software, if it conflicts with System Integrity Protection, might be set aside when you upgrade to OS X El Capitan or later.” And really, this was introduced four versions ago, five if you count the beta, so there’s little excuse to not comply by now.

UAMDM

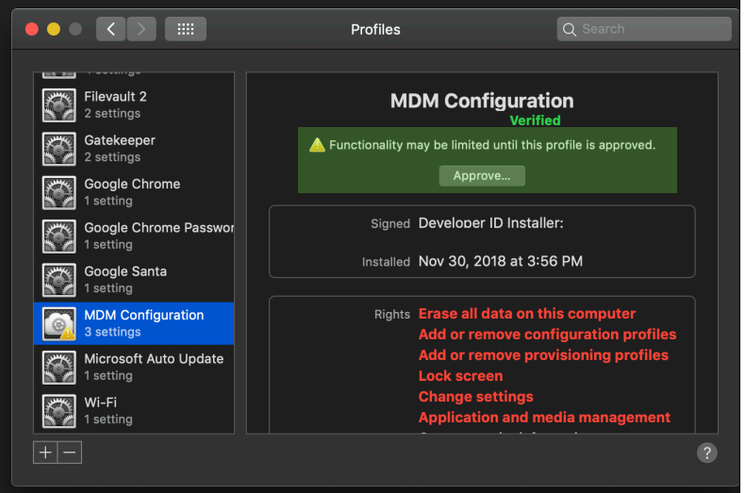


macOS High Sierra 10.13.2 introduced the concept of "User Approved" MDM enrollment. This enrollment type is required only if you want to **manage certain security-sensitive settings** on a Mac whose MDM enrollment is **not done through DEP.**

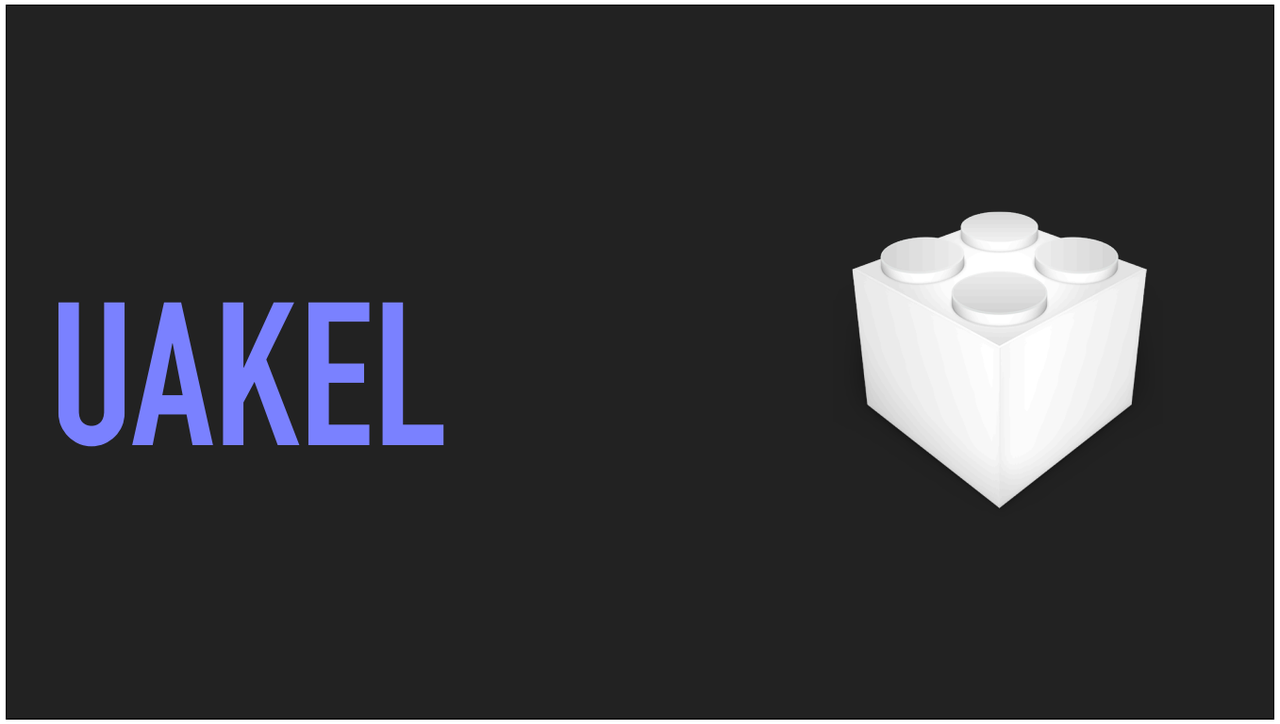
BUILT IN SECURITY

How to enroll a Mac in User Approved MDM:

- DEP
- ‘Grandfathered’ before 10.13.4
- Manually install
  - Approve in System Preferences



How to enroll a Mac in User Approved MDM:
If a Mac is enrolled in DEP, its enrollment is equivalent to User Approved when it enrolls in MDM.
If a Mac was enrolled in non-User Approved MDM before updating to macOS 10.13.4, its enrollment is converted to User Approved when installing macOS 10.13.4.
You can also download or email yourself an enrollment profile.
Double-click the profile, then follow the prompts in System Preferences to enroll in MDM.

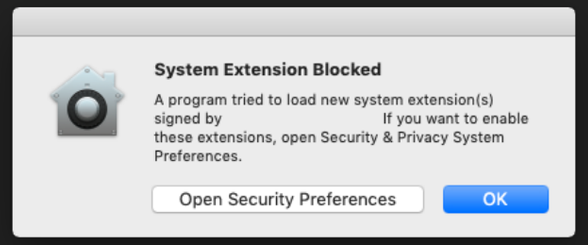


macOS High Sierra 10.13 introduces a new feature that requires user approval before loading newly-installed third-party kernel extensions (KEXTs).

BUILT IN SECURITY

USER APPROVED KEXT LOADING

- Initial load requests are denied
- Future load attempts will cause it to appear again BUT not prompt the user.

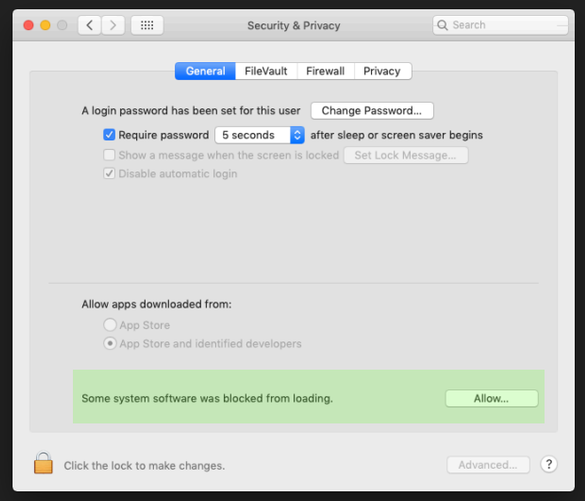


This feature enforces that only kernel extensions approved by the user will be loaded on a system. When a request is made to load a KEXT that the user has not yet approved, the load request is denied and macOS presents the alert.

Until the user approves the KEXT, it will not trigger another user alert.

USER APPROVED KEXT LOADING

- The user is prompted to approve the KEXT in System Preferences > Security & Privacy.
- UI is present for 30 minutes after the alert.



This approval UI is only present in the Security & Privacy preferences pane for 30 minutes after the alert. Until the user approves the KEXT, future load attempts will cause the approval UI to reappear but again, will not trigger another user alert.

In this example, I actually had multiple items attempt to load, if you only have one, you'll see the name of the item.

BUILT IN SECURITY

UAKEL & UAMDM

- Kernel Extension Policy
  - Specify which kernel extensions should load without user consent.
  - Optionally prevent users from approving additional kernel extensions.

What do we mean by security settings? One is kext loading.

Starting with macOS 10.13.4, User Approved Kernel Extension Loading is enabled on all devices, including those enrolled in MDM. A Kernel Extension Policy payload can be deployed by your MDM to:
Specify which kernel extensions should load without user consent.
Optionally prevent users from approving additional kernel extensions.

BUILT IN SECURITY

KERNEL EXTENSION POLICY

Key	Type	Value
AllowUserOverride	Boolean	If set to true, users can approve additional kernel extensions not explicitly allowed by configuration profiles.
AllowedTeamIdentifiers	Array of Strings	An array of team identifiers that define which validly signed kernel extensions will be allowed to load.
AllowedKernelExtensions	Dictionary	A dictionary representing a set of kernel extensions that will always be allowed to load on the machine. The dictionary maps team identifiers (keys) to arrays of bundle identifiers. For unsigned legacy kernel extensions, use an empty key for the team identifier.

Here we have the Keys and values that you can use to build your payload.

This profile must be delivered via a user approved MDM server.
In addition to the settings common to all payloads, this payload defines the following keys:

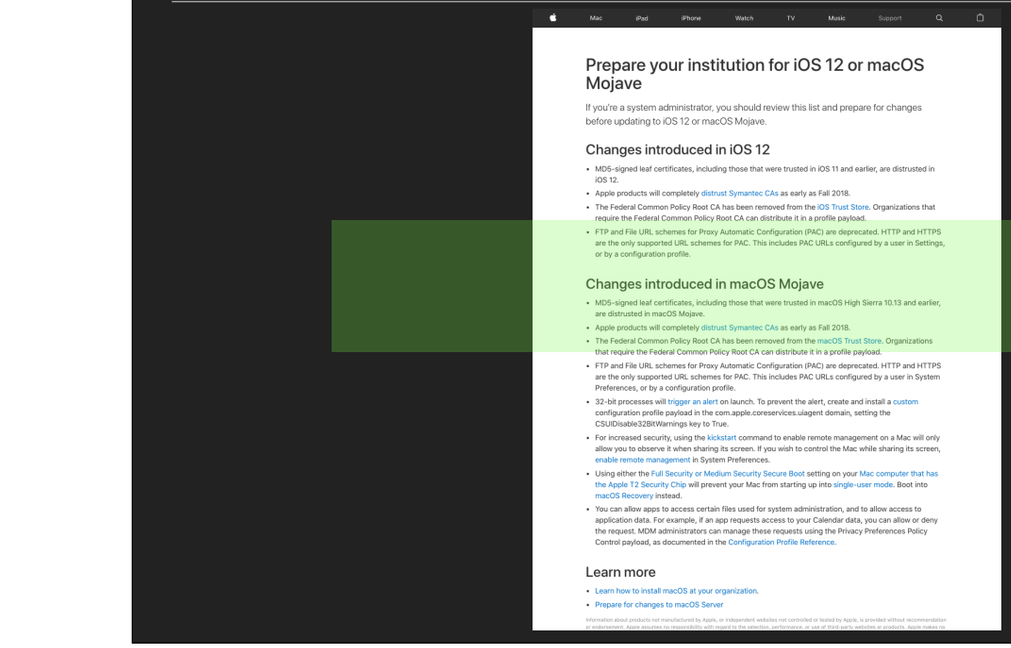
Allow User Override
Allowed Team Identifiers

Allowed Kernel Extensions

PRIVACY
PREFERENCES
POLICY
CONTROL

(Payload)

BUILT IN SECURITY



Mac iPad iPhone Watch TV Music Support

Prepare your institution for iOS 12 or macOS Mojave

If you're a system administrator, you should review this list and prepare for changes before updating to iOS 12 or macOS Mojave.

Changes introduced in iOS 12

- MD5-signed leaf certificates, including those that were trusted in iOS 11 and earlier, are distrusted in iOS 12.
- Apple products will completely distrust Symantec CAs as early as Fall 2018.
- The Federal Common Policy Root CA has been removed from the iOS Trust Store. Organizations that require the Federal Common Policy Root CA can distribute it in a profile payload.
- FTP and File URL schemes for Proxy Automatic Configuration (PAC) are deprecated. HTTP and HTTPS are the only supported URL schemes for PAC. This includes PAC URLs configured by a user in Settings, or by a configuration profile.

Changes introduced in macOS Mojave

- MD5-signed leaf certificates, including those that were trusted in macOS High Sierra 10.13 and earlier, are distrusted in macOS Mojave.
- Apple products will completely distrust Symantec CAs as early as Fall 2018.
- The Federal Common Policy Root CA has been removed from the macOS Trust Store. Organizations that require the Federal Common Policy Root CA can distribute it in a profile payload.
- FTP and File URL schemes for Proxy Automatic Configuration (PAC) are deprecated. HTTP and HTTPS are the only supported URL schemes for PAC. This includes PAC URLs configured by a user in System Preferences, or by a configuration profile.
- 32-bit processes will trigger an alert on launch. To prevent the alert, create and install a custom configuration profile payload in the com.apple.coreservices.uiagent domain, setting the CSUIDisable32BitWarnings key to True.
- For increased security, using the kickstart command to enable remote management on a Mac will only allow you to observe it when sharing its screen. If you wish to control the Mac while sharing its screen, enable remote management in System Preferences.
- Using either the Full Security or Medium Security Secure Boot setting on your Mac computer that has the Apple T2 Security Chip will prevent your Mac from starting up into single-user mode. Boot into macOS Recovery instead.
- You can allow apps to access certain files used for system administration, and to allow access to application data. For example, if an app requests access to your Calendar data, you can allow or deny the request. MDM administrators can manage these requests using the Privacy Preferences Policy Control payload, as documented in the Configuration Profile Reference.

Learn more

- Learn how to install macOS at your organization.
- Prepare for changes to macOS Server

Information about products not manufactured by Apple, or independent websites not controlled or tested by Apple, is provided without recommendation or endorsement. Apple assumes no responsibility with regard to the selection, performance, or use of third-party websites or products. Apple makes no

PPPC was introduced in macOS Mojave.

The Privacy Preferences payload was a new payload type for configuration profiles.

These settings allow app to access certain files used for system administration and allow access to application data

The settings are displayed in the ”Privacy” tab of the ”Security & Privacy” pane in System Preferences.

Note that This profile must be delivered via a user approved MDM.

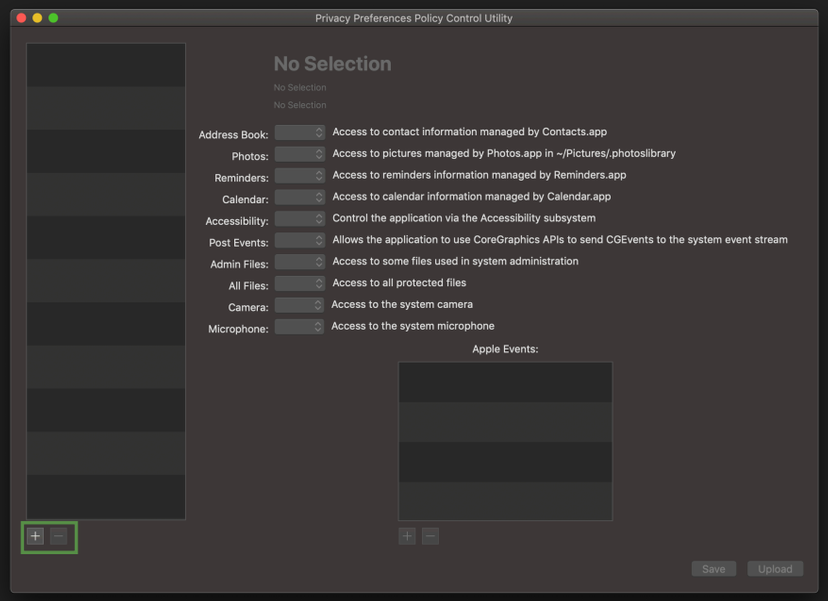
OPTIONAL KEYS

Key	Type	Description
AddressBook	array	Contact information managed by Contacts.app.
Calendar	array	Calendar information managed by Calendar.app.
Reminders	array	Reminders information managed by Reminders.app.
Photos	array	Pictures managed by Photos.app in ~/Pictures/.photoslibrary.
Camera	array	A system camera. Access to the camera cannot be given in a profile; it can only be denied.
Microphone	array	A system microphone. Access to the microphone cannot be given in a profile; it can only be denied.
Accessibility	array	Control the application via the Accessibility subsystem.
PostEvent	array	Allows the application to use CoreGraphics APIs to send CGEvents to the system event stream.
SystemPolicyAllFiles	array	Allows the application access to all protected files.
SystemPolicySysAdminFiles	array	Allows the application access to some files used in system administration.
AppleEvents	array	Allows the application to send a restricted AppleEvent to another process.

The Privacy Preferences payload is a specific PayloadType used a configuration profile.
It controls the settings that are displayed in the ”Privacy” tab of the ”Security & Privacy” pane in System Preferences.
This profile must be delivered via a user approved MDM server in a device profile.

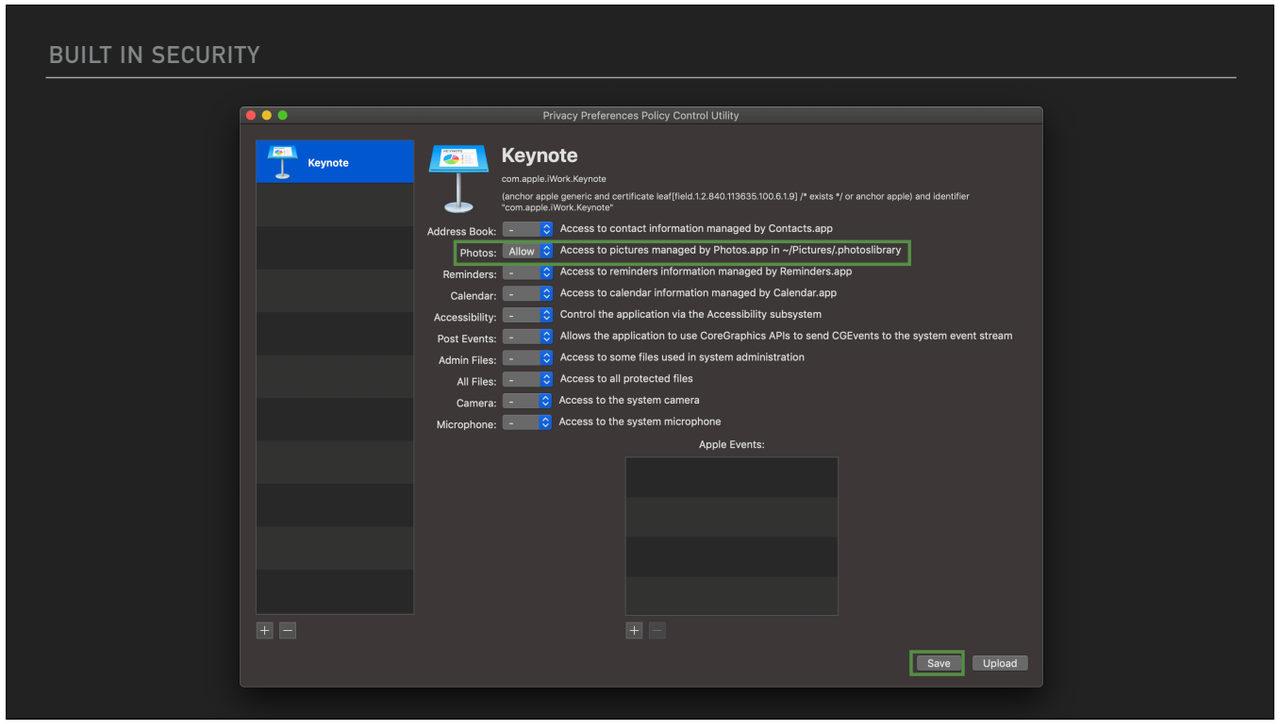
Here are some of the optional keys you can use.

BUILT IN SECURITY



There are a couple of PPC profile building utilities listed in the references, lets take a look at this open source one from jamf. You don’t need to be a jamf customer to use it, it is available on GitHub

Lets say you want to let keynote have access to your photos. You can click on the add button, to add the application of your choice

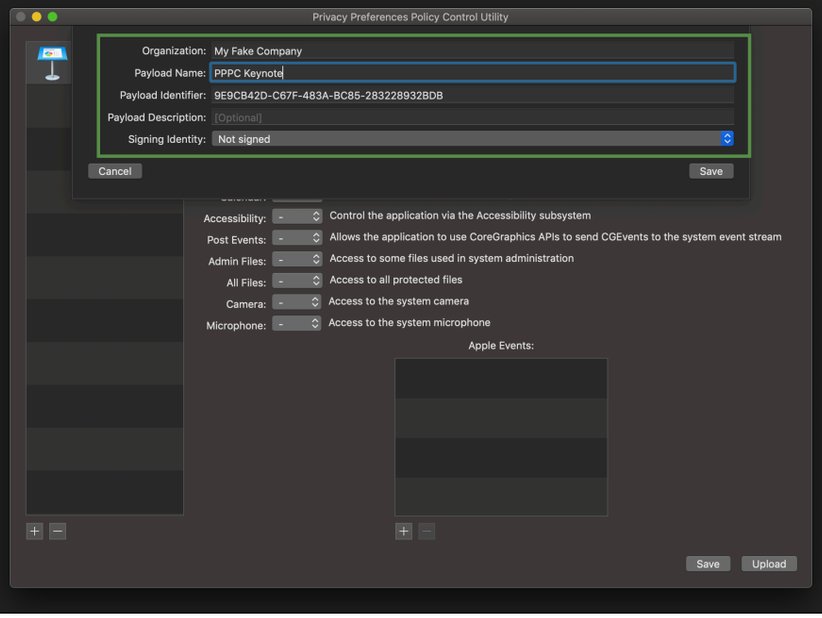


And toggle the photos option to allow.

Click save to move on to the next step.

If you click UPLOAD instead, it will attempt to upload directly to your jamf pro sever, because it is, after all, a jamf product.

BUILT IN SECURITY



After you click save, you add your details for the configuration profile, and you’re given the option to sign the profile. And then it will save the profile to your machine.

```
<?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>PayloadContent</key>
    <array>
        <dict>
            <key>PayloadDescription</key>
            <string>PPPC Keynote</string>
            <key>PayloadDisplayName</key>
            <string>PPPC Keynote</string>
            <key>PayloadIdentifier</key>
            <string>9E9CB42D-C67F-483A-BC85-283228932BDB</string>
            <key>PayloadOrganization</key>
            <string>My Fake Company</string>
            <key>PayloadType</key>
            <string>com.apple.TCC.configuration-profile-policy</string>
            <key>PayloadUUID</key>
            <string>312F3BA7-16DA-4153-A8B5-5D5FA2146084</string>
            <key>PayloadVersion</key>
            <integer>1</integer>
            <key>Services</key>
            <dict>
                <key>Photos</key>
                <array>
                    <dict>
                        <key>Allowed</key>
                        <true/>
                        <key>CodeRequirement</key>
                        <string>(anchor apple generic and certificate leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or anchor apple)
and identifier "com.apple.iWork.Keynote"</string>
                        <key>Comment</key>
                        <string></string>
                        <key>Identifier</key>
                        <string>com.apple.iWork.Keynote</string>
                        <key>IdentifierType</key>
                        <string>bundleID</string>
```

If you want to open that file and examine it, you'll find the xml, just as it was with our other configuration profiles.

SUMMARY

WHAT DID WE LEARN?

- How can you automate the authorization of a kext to load on all of your machines?
  - Kernel Extension Policy
  - Requires UAMDM
- How does PPPC relate to configuration profiles?
  - PPPC is a configuration profile payload

1. Whitelist profile, needs UAMDM
2. PPPC is a config profile payload, it is a config profile

PART FOUR:
“IMAGING”

ok, lets talk about imaging, the elephant in the room during many a meeting with Apple. Is it gone? The simple answer is,

yes. (with a small disclaimer at the bottom).

"IMAGING"

- What is it?
  - Traditional Imaging / Gold Master
- Why is it gone?

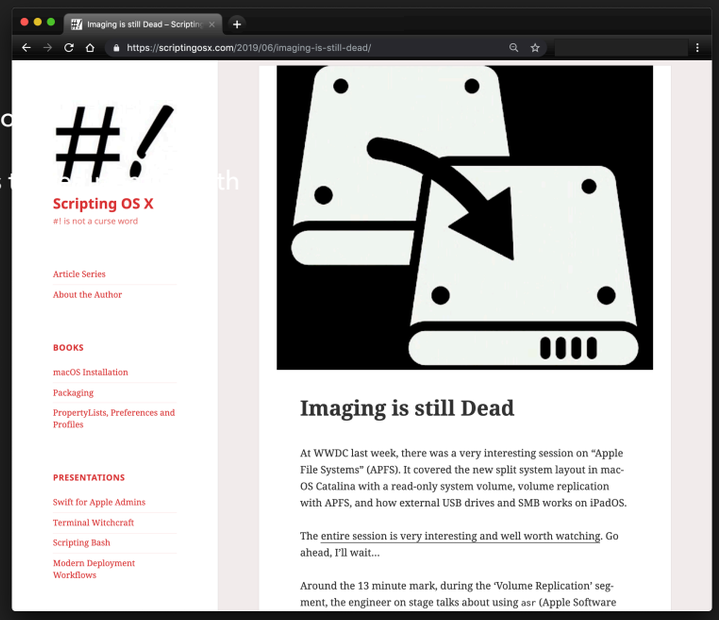
Because there is ambiguity, lets define what we are talking about.
Traditional imaging, using a gold master, block copying a single source of truth, is done.

No more gold master, no ultimate golden anything (that includes gauntlets).

Why is it gone? Examine the pieces

“IMAGING”

- Firmware is impo
- Firmware needs t
  the system



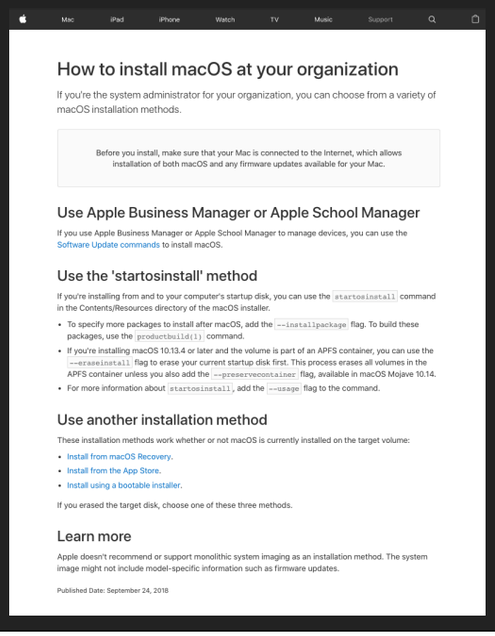
Scripting osx has a good write up about a WWDC session that got people talking about ‘imaging’.
Since it was explained better than I could have written, I’m going to read a short section for you:

Modern Macs don’t just require a few files on disk to make a bootable system. Inside your Mac are several subsystems that require their own systems (i.e. firmware) to run. Most prominent are the T1 or T2 system controllers which are actually independent custom ARM-based processors running a system called ‘iBridge’ which is an iOS derivate.

If you just exchange the ‘normal’ system files on the hard drive over TDM, without also updating the various firmwares in the system, you may get your Mac into state where it cannot boot.

This was most obvious with the macOS High Sierra upgrade. After re-imaging a 10.12 Sierra Mac to High Sierra running on APFS, would lead to a Mac that could not read the new system volume. The firmware update that came with High Sierra is needed, so the firmware can mount, read and start the APFS system volume.

‘IMAGING’

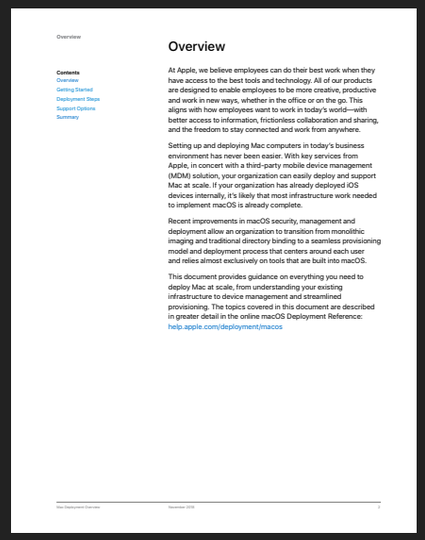


How to install macOS at your organization.

Apple has given clear instruction in their documentation that the gold master imaging method should be left in the past.

Apple doesn’t recommend or support monolithic system imaging as an installation method. The system image might not include model-specific information such as firmware updates.

‘IMAGING’


Overview

Contents
Overview
Getting Started
Deployment Steps
Support Options
Summary

Overview

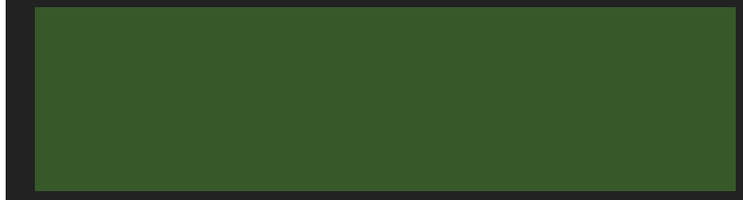
At Apple, we believe employees can do their best work when they have access to the best tools and technology. All of our products are designed to enable employees to be more creative, productive and work in new ways, whether in the office or on the go. This aligns with how employees want to work in today’s world—with better access to information, frictionless collaboration and sharing, and the freedom to stay connected and work from anywhere.

Setting up and deploying Mac computers in today’s business environment has never been easier. With key services from Apple, in concert with a third-party mobile device management (MDM) solution, your organization can easily deploy and support Mac at scale. If your organization has already deployed iOS devices internally, it’s likely that most infrastructure work needed to implement macOS is already complete.

Recent improvements in macOS security, management and deployment allow an organization to transition from monolithic imaging and traditional directory binding to a seamless provisioning model and deployment process that centers around each user and relies almost exclusively on tools that are built into macOS.

This document provides guidance on everything you need to deploy Mac at scale, from understanding your existing infrastructure to device management and streamlined provisioning. The topics covered in this document are described in greater detail in the online macOS Deployment Reference: help.apple.com/deployment/macos

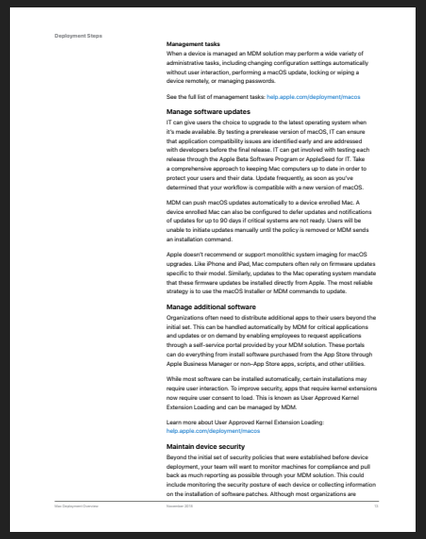
Mac Deployment Overview November 2018 2



And they mean it, they keep saying it. Here in Mac Deployment Overview, they restate saying

Recent improvements in macOS security, management and deployment allow an organization to transition from monolithic imaging and traditional directory binding to a seamless provisioning model and deployment process that centers around each user and relies almost exclusively on tools that are built into macOS.

‘IMAGING’


Deployment Steps

Management tasks
When a device is managed an MDM solution may perform a wide variety of administrative tasks, including changing configuration settings automatically without user interaction, performing a macOS update, locking or wiping a device remotely, or managing passwords.

See the full list of management tasks: help.apple.com/deployment/macos

Manage software updates
IT can give users the choice to upgrade to the latest operating system when it's made available. By testing a prerelease version of macOS, IT can ensure that application compatibility issues are identified early and are addressed with developers before the final release. IT can get involved with testing each release through the Apple Beta Software Program or AppleSeed for IT. Take a comprehensive approach to keeping Mac computers up to date in order to protect your users and their data. Update frequently, as soon as you've determined that your workflow is compatible with a new version of macOS.

MDM can push macOS updates automatically to a device enrolled Mac. A device enrolled Mac can also be configured to defer updates and notifications of updates for up to 90 days if critical systems are not ready. Users will be unable to initiate updates manually until the policy is removed or MDM sends an installation command.

Apple doesn't recommend or support monolithic system imaging for macOS upgrades. Like iPhone and iPad, Mac computers often rely on firmware updates specific to their model. Similarly, updates to the Mac operating system mandate that these firmware updates be installed directly from Apple. The most reliable strategy is to use the macOS Installer or MDM commands to update.

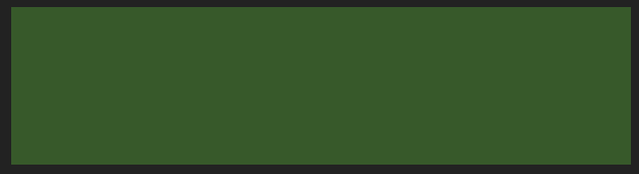
Manage additional software
Organizations often need to distribute additional apps to their users beyond the initial set. This can be handled automatically by MDM for critical applications and updates or on demand by enabling employees to request applications through a self-service portal provided by your MDM solution. These portals can do everything from install software purchased from the App Store through Apple Business Manager or non–App Store apps, scripts, and other utilities.

While most software can be installed automatically, certain installations may require user interaction. To improve security, apps that require kernel extensions now require user consent to load. This is known as User Approved Kernel Extension Loading and can be managed by MDM.

Learn more about User Approved Kernel Extension Loading:
help.apple.com/deployment/macos

Maintain device security
Beyond the initial set of security policies that were established before device deployment, your team will want to monitor machines for compliance and pull back as much reporting as possible through your MDM solution. This could include monitoring the security posture of each device or collecting information on the installation of software patches. Although most organizations are

Mac Deployment Overview November 2018 13



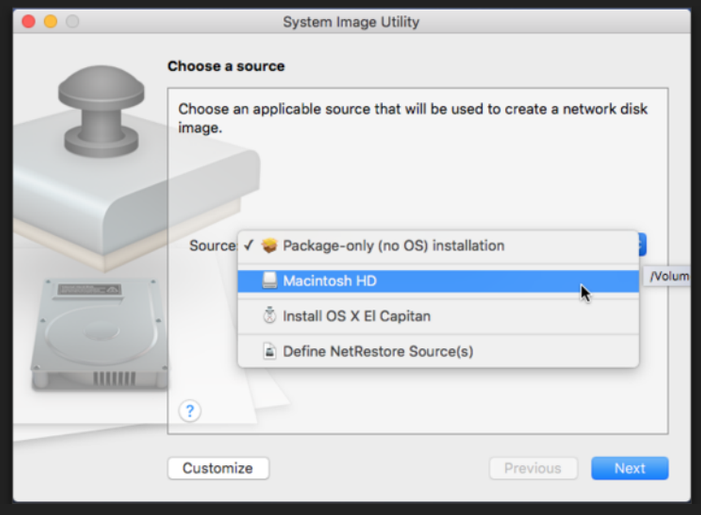
And, third times the charm.

Apple doesn’t recommend or support monolithic system imaging for macOS upgrades,

And it continues to explain again why…..

“like iPhone and iPad, Mac computers often rely on firmware updates specific to their model. Similarly, updates to the Mac operating system mandate that these firmware updates be installed directly from Apple. The most reliable strategy is to use the macOS Installer or MDM commands to update.”

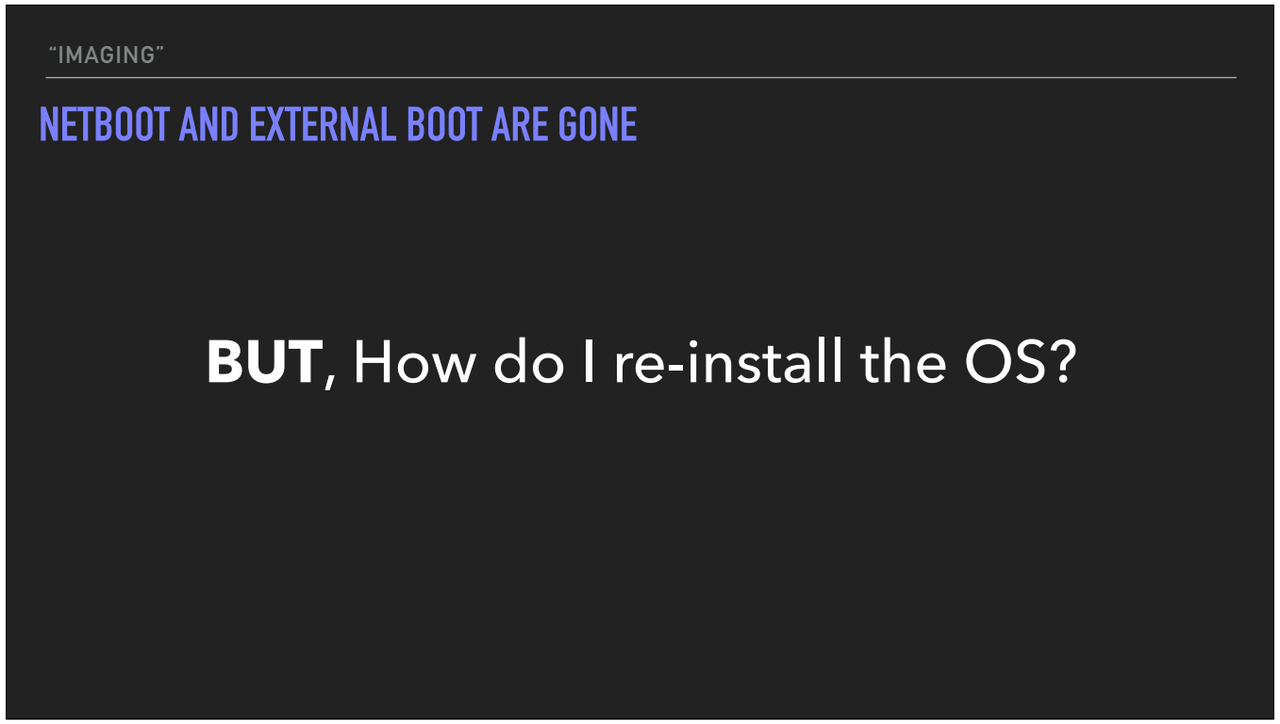
NETBOOT AND EXTERNAL BOOT ARE GONE



With the T2 chip, Apple has eliminated Netboot and external boot is disabled by default.

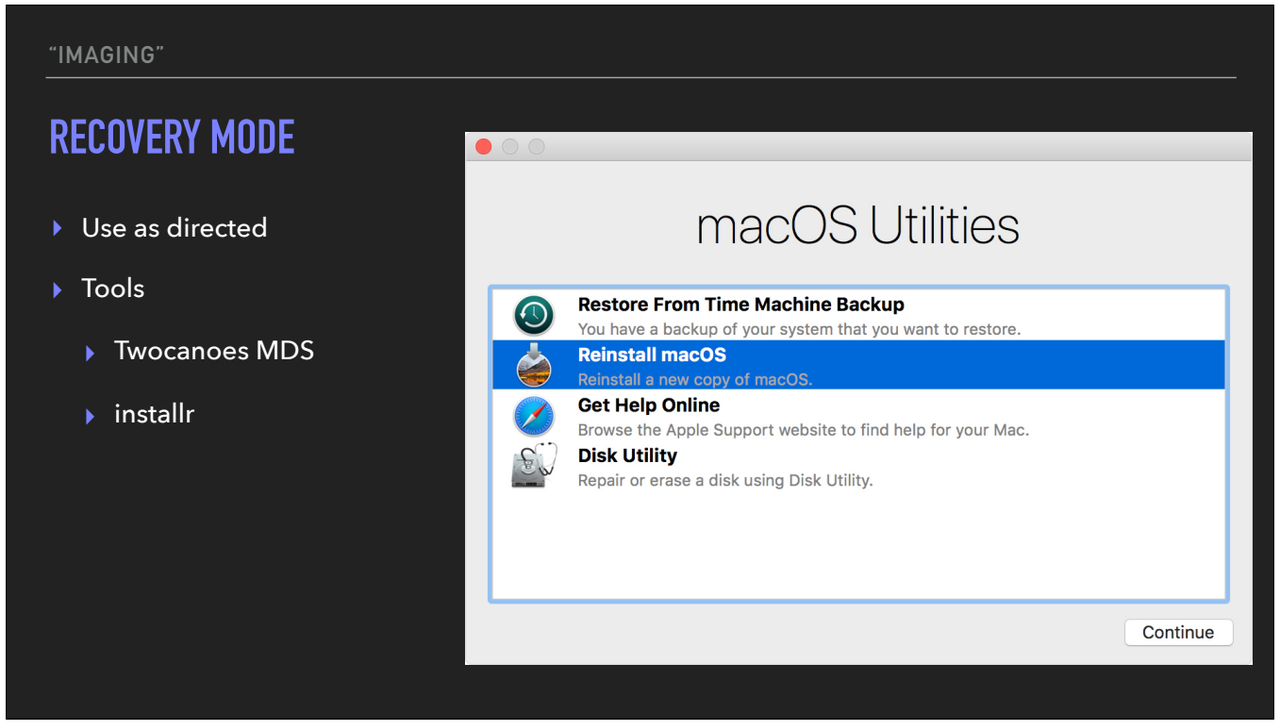
In fact, if you're less than say two years in, I hope you don't even recognize this screenshot.

You can still External boot, you have to turn off the secure boot function, and its a non-automated process, requiring a full setup of the machine. Which, in my option, eliminates any benefit you'd have from automation.



How do I reinstall the OS?

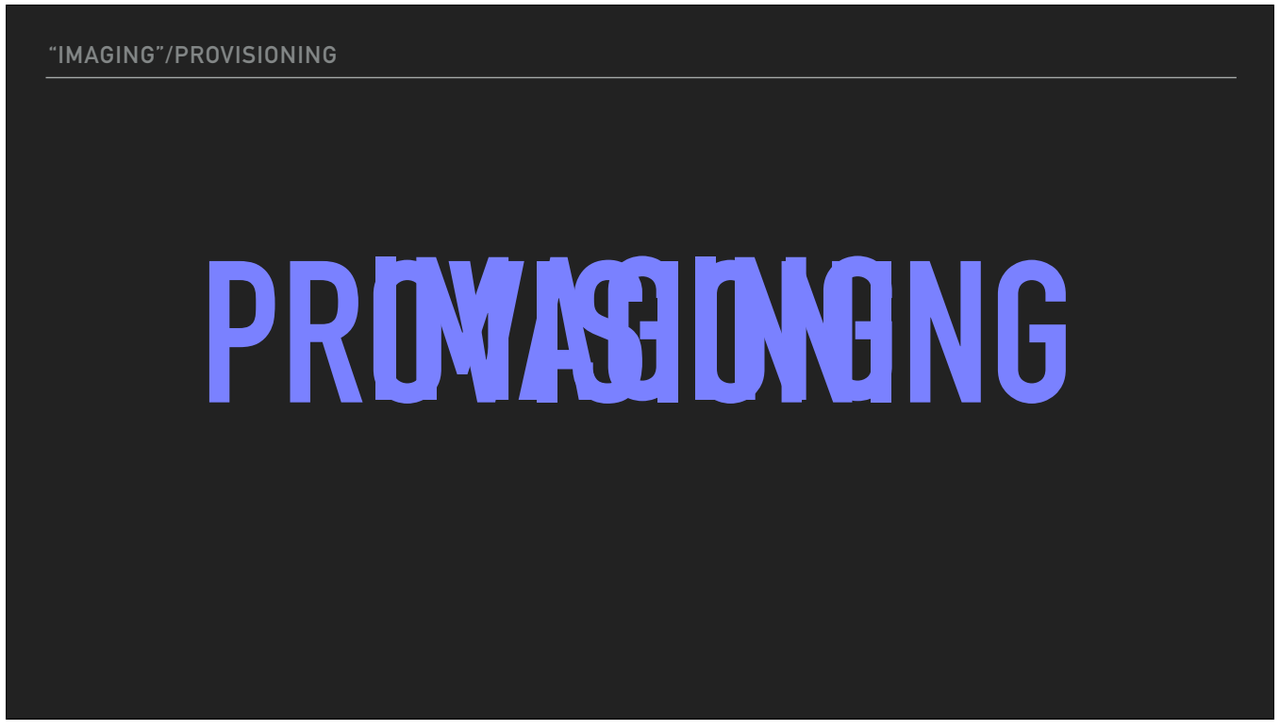
Use recovery mode.



Depending on who you ask, and their personal vocabulary, you might hear/read that "re-imaging" is still a thing. But for simplicity's sake, its not. Reinstalling the OS is not re-imaging, it is returning the machine to factory settings, which you can do with Recovery Mode.

Already there are free tools from the community you can use (this is not an exhaustive list) that focus on Recovery in the T2 world.

MDS from Twocanoes and installr, which can be found in the Munki repo on github.



I hope that I've established "imaging is dead.
So lets destroy it for good,

And replace it with provisioning

PROVISIONING

What do I mean by provisioning?

Provisioning is taking all the tools we’ve talked about today, your mdm, profiles, DEP, packaging, and more and you’re adding it to a known good base.

Take, carol, captain marvel, picking her new suit colors. Theres nothing wrong with her suit.

Its got lots of fancy tech, maybe its bullet proof,

but now she’s personalizing it for her purpose.

Obviously you’ll be doing more than changing the color of your ‘suit’ (OS), but you know, its an analogy.
Adding personalization to a known good base.

SUMMARY

WHAT DID WE LEARN?

- Is imaging dead?
  - YES.
- Why?
  - Firmware
  - Removal of Netboot
  - Removal of External boot (SecureBoot)

so, what did we learn?

imaging, is it dead?
Yes. sorry, no prize for that one, it should be obvious.

Why is it dead?

REFERENCES:

https://bit.ly/2JdU0BE

This google drive folder will contain references for all four sessions.

QUESTIONS ?