



Leveraging DEPNotify and Jamf Pro for Device Deployment

Or: How I learned to stop imaging and use Device Enrollment
Or or: LDEPNAJPFDD

(Delete before
presenting)

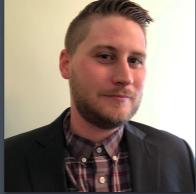
Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

C

9 |



John Mahlman

Network Systems Administrator
The University of the Arts, Philadelphia

- Over 10 years in Mac IT
- Write bad code
- Brew good beer
- Play Tabletop Games
- Love Philly sports teams

Find me: @jmahlman (slack, git, jamfnation)
Website: <https://yearofthegeek.net>

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

© JAMF Software, LLC

Just a bit about me. I'm a "Network Systems Administrator" at The University of the Arts in Philadelphia, I've been there since December 2015.

I have over 10 years of Mac IT experience, I write somewhat bad code as you will see today, I brew some good beer, and I play tabletop games.

I am also jmahlman on various platforms.

And my website is there, [yearfofthegeek.net](https://yearofthegeek.net). I don't post often but it's basically where I go to put my processes up for future reference and hopefully to help people out.

9

LDEPNAJPFDD

Presentation agenda:

- The recent past (Imaging)
- What happened? *It* happened...
- Options we considered
- Find the process
- What we built
- What's next?

(Delete before
presenting)

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

Here's a basic rundown of what I'm going to go over today. Past, present, and future?

UArts at a Glance

- Approximately 1,800 students
- 6 Academic buildings
- Over 200 “student facing” public Macs
- Offices, faculty/staff, Students (BYOD) – 97% Macs
- Computers range from 2009-2018 models
- On-Prem Jamf Pro since 2012
- Over 1,700 managed systems

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

© JAMF Software, LLC

As mentioned, I work for The University of the Arts in Philadelphia. Here are some stats at a glance.

C We have around 1800 students _C_ spread across 6 buildings. We’re a majority Mac campus, only a handful of windows machines...so

C We have over 200 “student facing” machines in a little over 20 labs, 40 smart-classrooms, and a few other rooms such as studios and production suites.

C We also manage office systems, faculty and staff laptops which are university provided, and finally student BYOD systems.

C We have a lot of variation in systems...2009–2018 models are in use with operating systems from 10.9 to 10.13. Thankfully most, if not all of those 2009–2011 systems and 10.9–10.10 systems are being upgraded or removed. This year.

C We’ve had on-prem jamf pro since 2012 _C_ with over 1700 systems at any time in there.

9

Where did we start?

Let's go back in time a few years..er..months...weeks?

*(Delete before
presenting)*

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

C



Not too long ago..the imaging process looked like this...

C You sent out this command or something like it to a bunch of machines using ARD or jamf remote or a policy and your machine would boot to a netboot set _C_ that was on a server then your imaging application (jamf imaging) would take over.

And walk in the next day or whatever and your machines would be imaged.

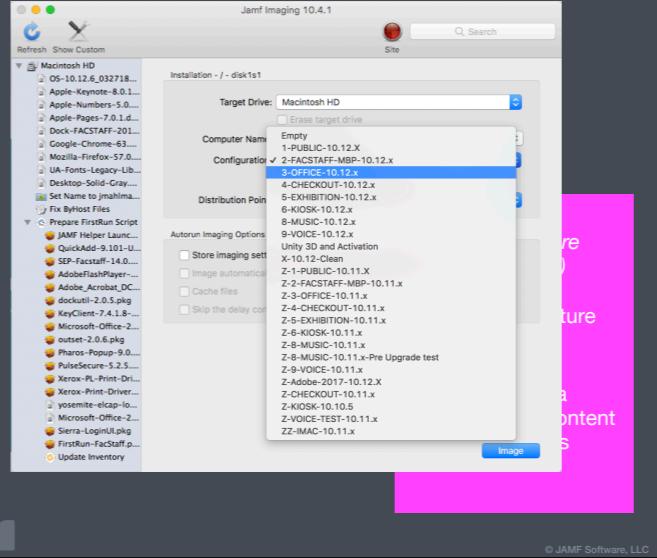
We did this a lot actually, during summer maintenance we would scope a bunch of labs with our netboot policy, have autorun set and we didn't have to babysit anything.

9 | Imaging... (not too long ago)

Send command or
Set policy....

Go home...

Have beverage!



Imaging would load the configuration based on the machine record or pre-stage, or our helpdesk manually chose one and it just worked.

And things were good. Set it, forget it, drink!

9

Imaging... (not too long ago)

Send command or
Set policy....

Go home...

Have beverage!



(Delete before
presenting)

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

And then *it* happened...

You all know what I'm talking about...



(Leave
area
free
of critical
content
on all slides)

Leave area
free of critical content
on all slides

© JAMF Software, LLC

C You know what I'm talking about...

9 | *It happened*

“Apple doesn't recommend or support monolithic system imaging as an installation method, because the system image might not include model-specific information such as firmware updates.”

*Apple, <https://support.apple.com/en-us/HT208020>
(Obtained 8/7/18)*

(Delete before presenting)

Picture-in-picture area.

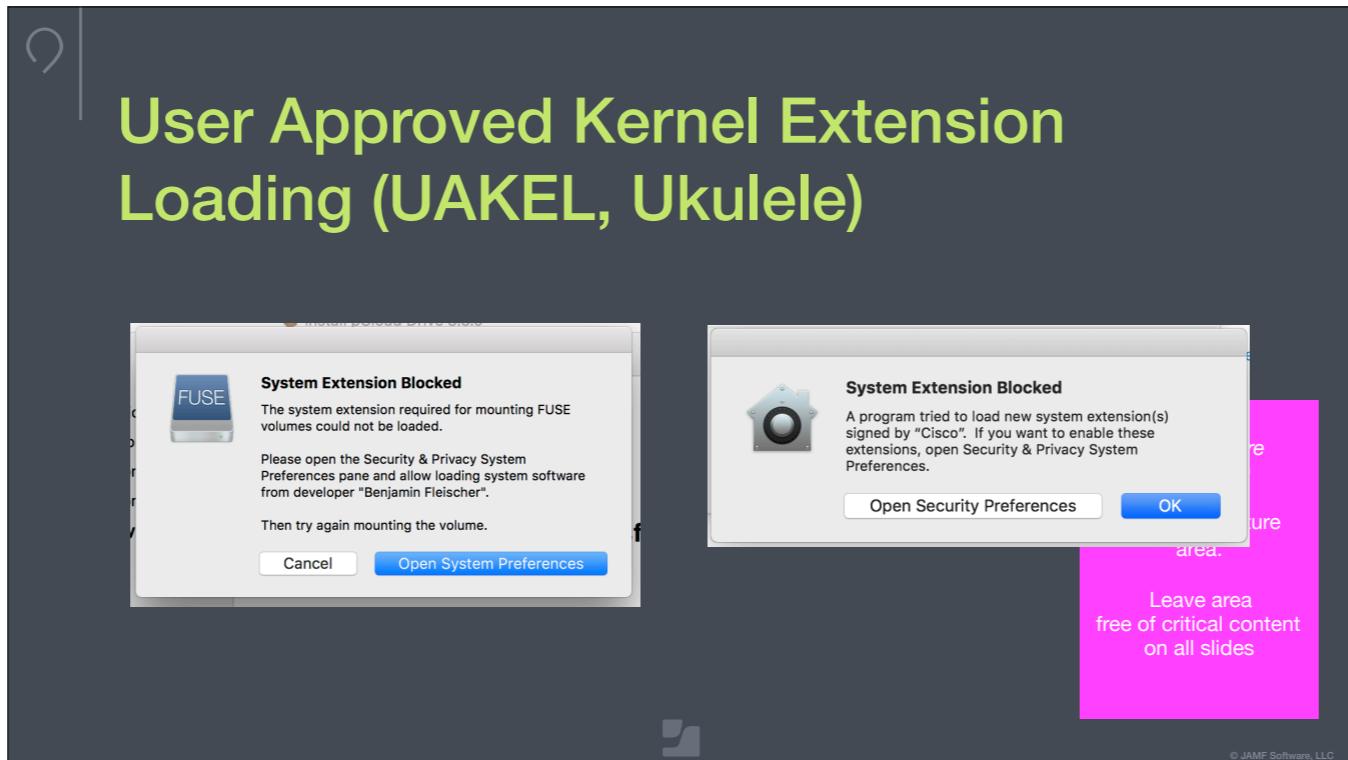
Leave area free of critical content on all slides

© JAMF Software, LLC

Apple gave us this update.

But I mean, just because it's not recommended doesn't mean it wouldn't work..or..

User Approved Kernel Extension Loading (UAKEL, Ukulele)



Ah, yeah...10.13.3 gave us UAKEL.

User Approved MDM (UAMDM)



(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

And then UAMDM. Which was nice because it helped with UAKEL but we still couldn't automate this really..

But there were some workarounds with upgrades and grandfathering.

9 | Apple T2 chip/Secure Boot

“Secure Boot offers three settings to make sure that your Mac always starts up from a legitimate, trusted Mac operating system...Full Security is the default Secure Boot setting...”

*Apple, <https://support.apple.com/en-us/HT208330>
(Obtained 8/7/18)*

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

© JAMF Software, LLC

Ah, right. The T2 chip and Secure boot.

These essentially killed any automated imaging..so the question that's been asked for a while now _C_

Is imaging dead?

Let's google!

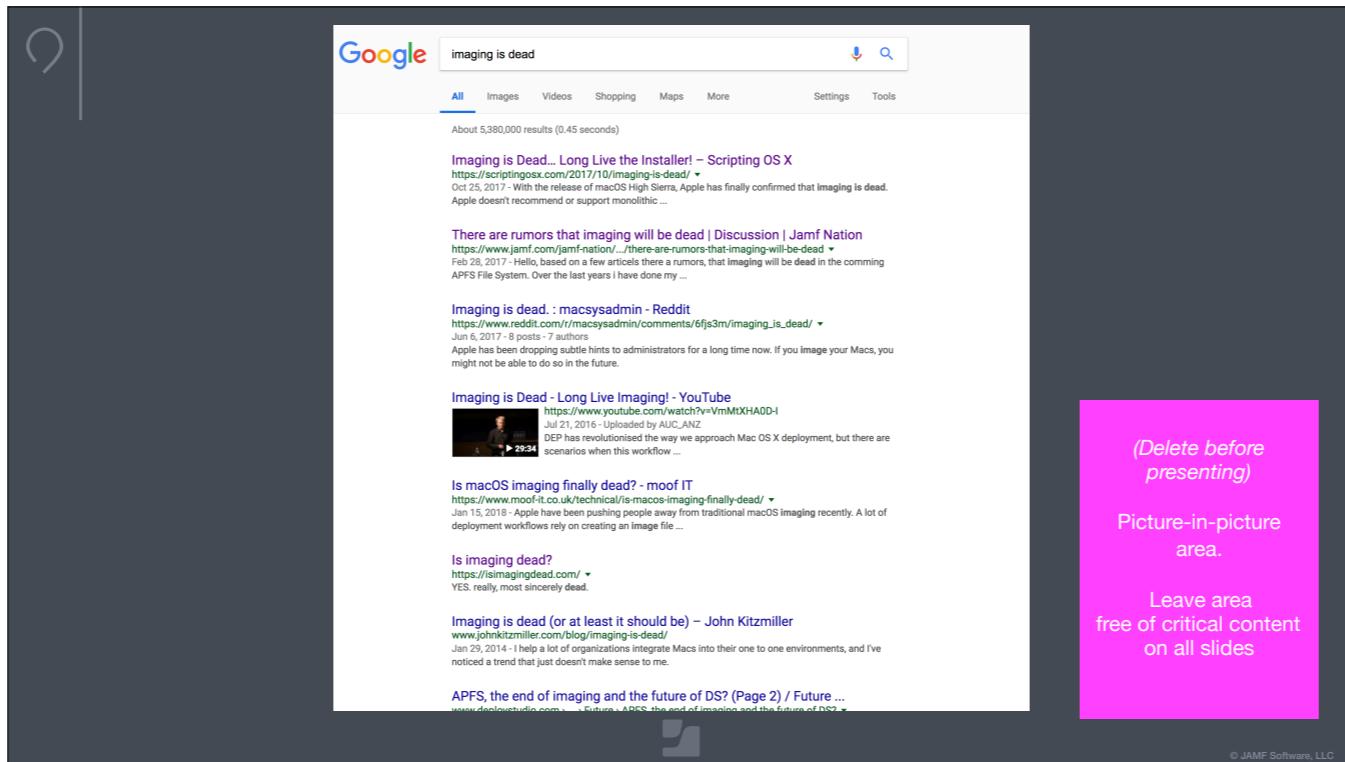
*(Delete before
presenting)*

Picture-in-picture
area.

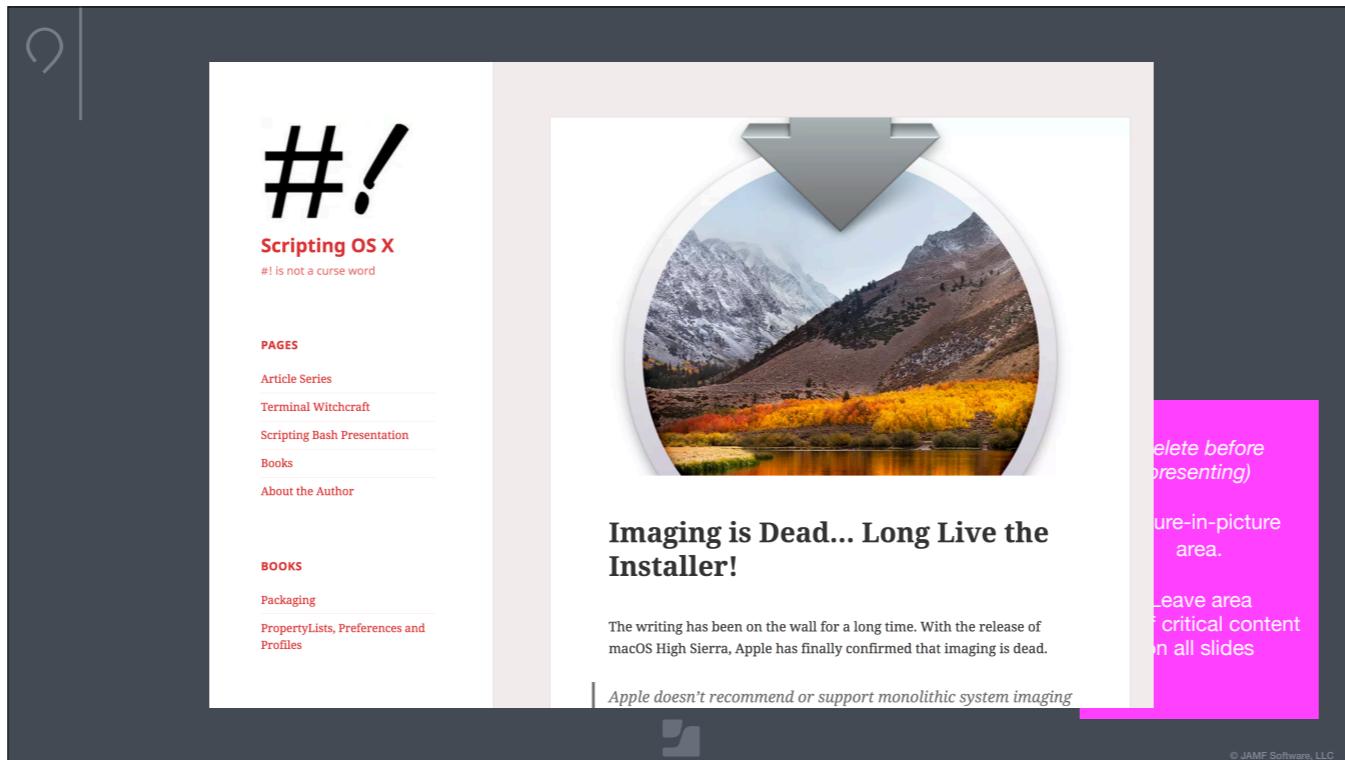
Leave area
free of critical content
on all slides

© JAMF Software, LLC

Let's check google. _C_.



Wow, that's a lot of stuff..let's click some of them



Or buddy Armin...Oh, yeah.. Okay, how about Jamf Nation _C_

Deploying High Sierra 10.13 with Casper Imaging

Posted: 11/2/17 at 11:28 PM by reddrop

Imaging, Casper Imaging, AutoDMG

Hi All,

I thought i would share with the community my method for deploying High Sierra with Casper Imaging. The method basically consists for two scripts.

I have tested this with a 10.13.1 Netboot Image created with AutoDMG and AutoCasperNBI

restoreMacOS.sh To deploy the correct image which runs as a "before" script

imagingPostInstall. To run after reboot to run the Sierra Upgrade if needed.

The first script will decide which image to deploy to the Mac based on the currently installed Operating System or File system.

-If it is APFS it will deploy the 10.13.1 APFS image to the APFS container.
-If it is HFS+ it will check the currently installed OS. If the OS is already 10.13 it will image using the 10.13 HFS+ image.
-If the filesystem is HFS+ and the current OS is not 10.13 it will deploy 10.12.6 and then upgrade to 10.13 on next reboot using startOSInstall.

Script 1 RestoreMacOS.sh This replaces your base image in your Casper Admin Config

```
#!/bin/bash
#####
# RestoreMacOs.sh - Ashley Stonham <reddrop>
# Restores either High Sierra or Sierra based on a best guess
#
# Variables:
# DP      - Set this to the path that your DP mounts as
# SIERRA   - Set this to the filename of your Sierra image
# HSIERRAAPFS  - Set this to the filename of your HighSierra APFS Image
# HSIERRAHFS  - Set this to the filename of your HighSierra HFS Image
```

(Delete before presenting)

picture-in-picture area.

Leave area of critical content on all slides

Deploying High Sierra with casper imaging?!

jamf NATION Discussions Feature Requests Knowledge Base Support More jmahlman

High Sierra Imaging: How-To

Posted: 2/2/18 at 3:10 PM by chrisdaggett

CCT

First: This is not Apple approved. I know that, you know that, do it at your own risk. I do, this works fine.

This is not a discussion about why or opinions :). Some of us have this need, this is a solution!

Thought I would share as I have seen a number of folks looking for these answers. Some of you are just looking for a certain piece of this, and that's great. Some of you are looking for a way to make an imager and that is here too. If you want to take the whole thing and make a fully automated USB Imager than that is also here :). If you don't like imaging, don't want to use this, or don't like it. That is okay! We each have our own thoughts and way of doing things, this isn't for you :).

If you are interested, dabble in this, or have other thoughts/ways to make it better PLEASE post and share below!

APFS / High Sierra Imaging: (Steps 1 and 3 are assuming your machines have never been on 10.13 and are missing the new firmware. If they are already upgraded you can skip Step 1 and 3)

STEP 1: Make a Firmware update Package for/from the device you want to deploy:

From the type of device you want to deploy to, have full High Sierra install App in application folder and run: (doesn't have to be from a 10.13 machine)

```
#!/bin/sh
# Based on investigations and work by Pepijn Bruienne
# Expects a single /Applications/Install macOS High Sierra*.app on disk

IDENTIFIER="com.foo.FirmwareUpdateStandalone"
VERSION=1.0

# find the Install macOS High Sierra.app and mount the embedded InstallESD disk image
```

© JAMF Software, LLC

How to image high sierra??

Yeah..posts about people basically learning how to deploy again. _C_



So yeah, it's dead , Jim_C_

Mostly. Because let's be honest, it still works (for most machines), but that may not last for long.. _C_

9 |

So, what are we going to do?

-Me, 2017

*(Delete before
presenting)*

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

Option 1: Stay on 10.12

- + Most of our software works fine on 10.12
- + Our current workflow works fine
- Security Updates will eventually stop
- New Machines will come with 10.13
- Some Apple software already updated to 10.13 only

Option 2: In-Place Upgrade

- + Quick process
- + No more imaging at all on public systems
- Computers will have leftover bits from software
- A lot more manual work than desired

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

© JAMF Software, LLC

Well, we looked at a bunch of options. Forgive the wall of text...

Our first option was to just stay on 10.12. <READ LIST> This would be fine for us in the short term but not in the long term at all. _C_

Our next option was to do an in-place upgrade. Apple gives us the tools, right? Let's use them...so the pros and cons again. <READ LIST> This one was looking good for us but we still didn't like those cons at all. _C_

9

Option 3: In-place Upgrade then image in future

- + Firmware is installed at upgrade
- + Workflows are already good
- + Same issues as Option 2 (leftovers, more work)
- UAMDM will not automatically work
- UAKEI will not work until we manually allow MDM
(AV software, sound drivers, etc.)

(Delete before presenting)
Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

Option 3 was sort of a hybrid solution and I'll be honest with you, this is the process we're using as a very quick stop-gap for our labs. Because again, imaging STILL works on 10.13 with OUR hardware..but we would run into the same issues again plus the UAMDM and UAKEI issues, unless you image to 10.13.3 then upgrade..but for a long term solution, that won't work.

9 |

Really, what are we going to do?

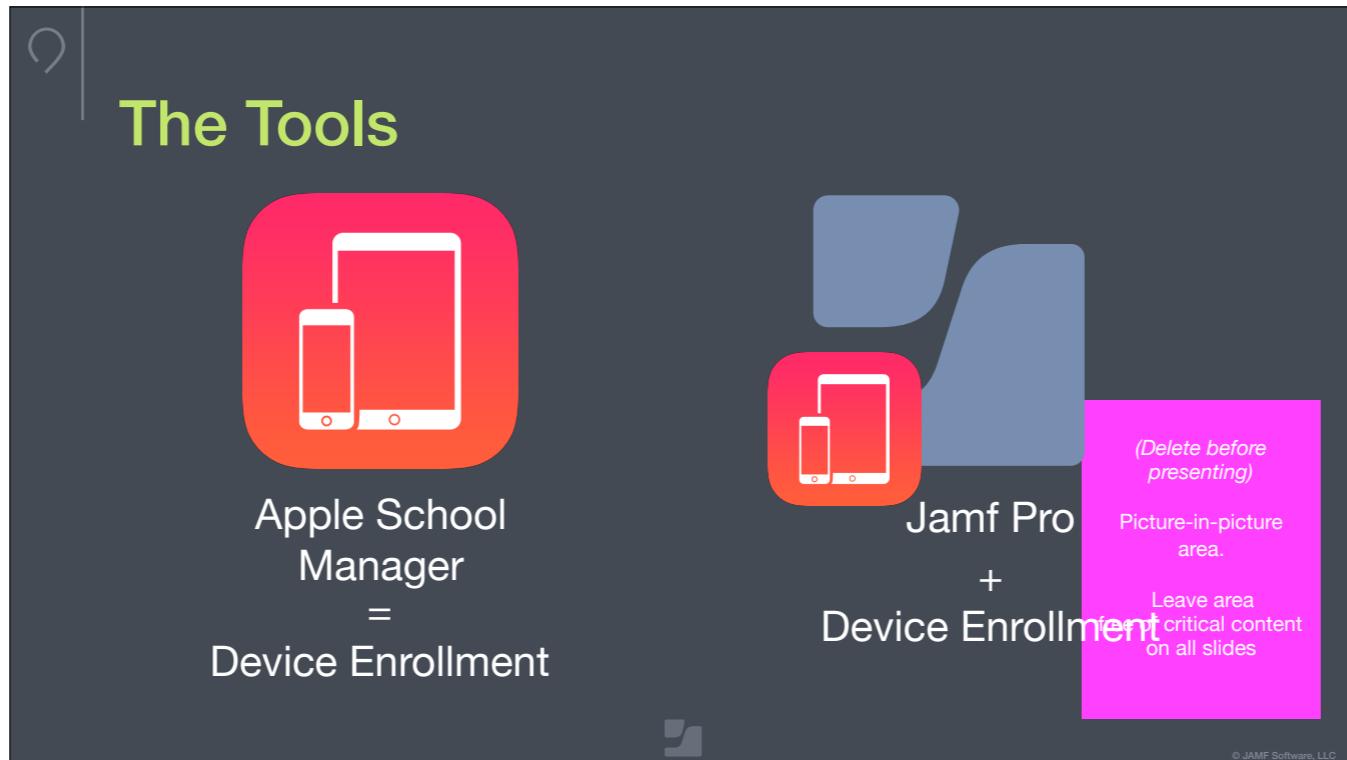
-Also me, 2018

*(Delete before
presenting)*

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

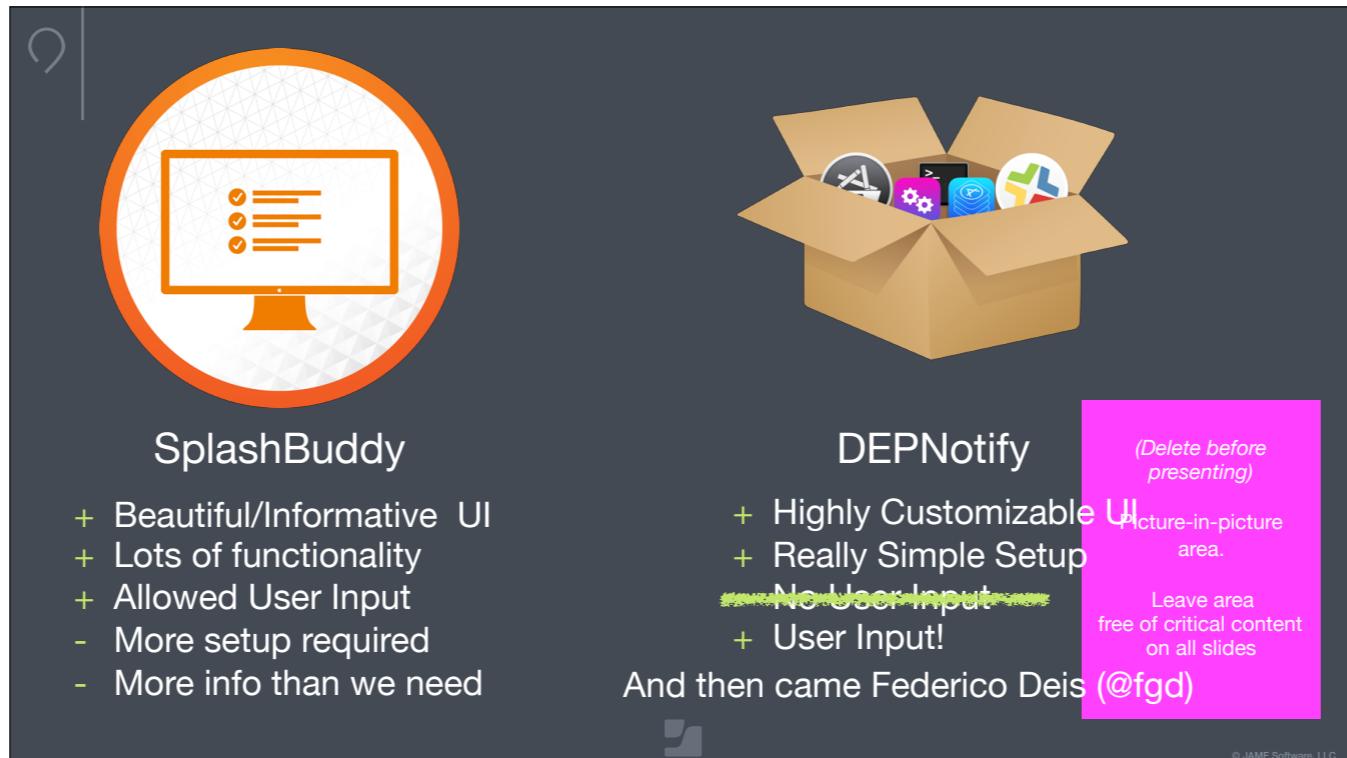


Let's look at the tools we have.

We have Apple School manager _C_ which is just Device Enrollment for schools.

And we have jamf pro _C_ which works with Device Enrollment

Okay, great, now how do we put these together and make it simple and a nice user experience? Well..there are apps for that! And we took a look at two of them. _C_



The first one was SplashBuddy and the other was the new kid DEPNotify.

Splashbuddy was very enticing _C_ Looking at the pros, it was a beautiful and informative UI and offered lots of nice functionality and most importantly, user input.

However, _C_ it had a little more setup on the server side than we really wanted to deal with and was more information than we needed for our process.

C And now DEPnotify. <READ PRO LIST> Unfortunately _C_ no user input... Now, that last one was pretty important to us, we wanted user input so we can name machines and add asset information..so we were really going to go for splash buddy... _C_

And then came Frederico Deis (FGD on slack). He posted in the depnotify channel that he was looking for people to test his forked build that offered user input! So I downloaded, tested it and figured out how to use it and it worked really well! _C_ So taking that “con” away..I had my decision.

DEPNotify

- Reads input echoed into log file
- Input sets up UI and controls flow
- All UI aspects are controllable



```
echo "Command: MainTitle: New Mac Setup" >> $DNLOG
echo "Command: Image: /var/tmp/your-logo.png" >> $DNLOG
echo "Command: WindowStyle: NotMovable" >> $DNLOG
echo "Command: ContinueButtonRegister: Begin" >> $DNLOG
echo "Status: Please click the button below..." >> $DNLOG
```

(Delete before presenting)
Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

A quick and dirty rundown of how DEPNotify works _C_
it reads input echoed into a log file, that input controls the UI and flow of the app and basically all UI aspects are controllable.
Here is a snippet of what the code looks like...

You can see it just echoes COMMAND and then a command type and then a value. Pretty nifty!

The Process...

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

So here was the process we were going with..

Get new machines and prep them with the base OS with either DEP for a new machine or internet recovery for a reused one. Boot machines to setup and install the software that was needed and when the machine was going to be assigned we would rename it and enter asset information then give it to the user.

The Process...

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

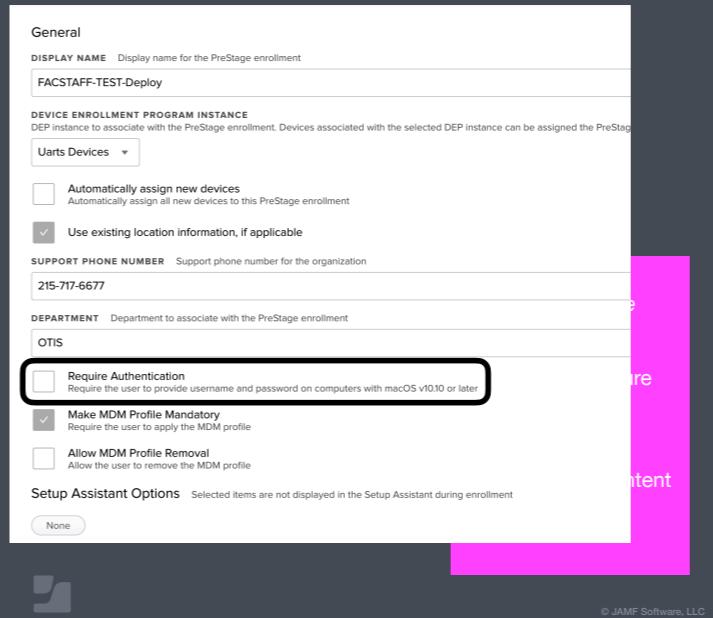
© JAMF Software, LLC

Let's take a look at the preparation.

Preparation...

New Machines

- Assign to MDM
- Setup Prestage
- Assign Devices to Prestage



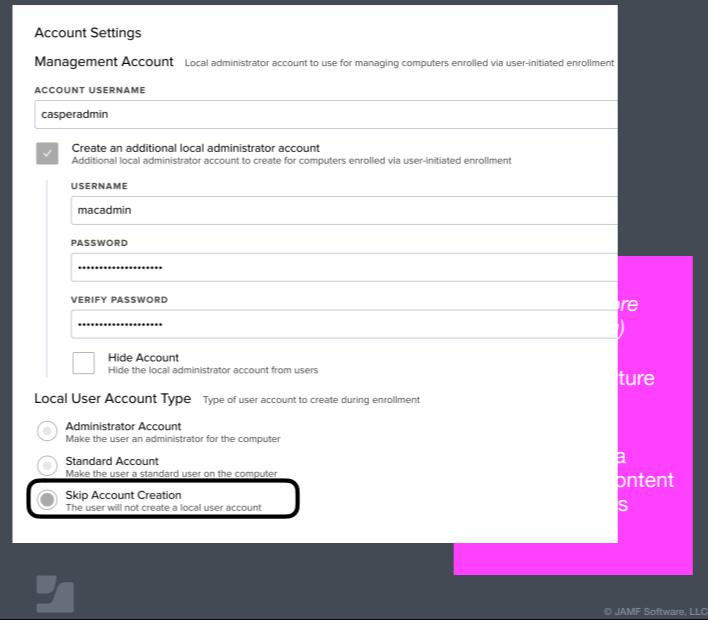
New machines would get assigned in our server via Apple School Manager. Make sure prestage is configured then assign the machine to the correct pre-stage setup.

Note C the checkbox for “require authentication” is unchecked. This is because we want our techs to setup the machine and if you require authentication, that machine gets assigned to that user and we didn’t want that.

Preparation...

New Machines

- Assign to MDM
- Setup Prestage
- Assign Devices to Prestage



The next setting was to create our local admin account, macadmin. That is an account that we have on all machines.

Note C_ that we skip account creation. We do this again so the machine isn't completely locked to a user until we're ready.

Preparation...

Existing Machines (APFS)

- Package Installer
- Script with
‘eraseinstall’ and
‘nointeraction’ flags
- Make Policy

```
#!/bin/bash  
/Applications/Install\ macOS\ High\ Sierra.app/Contents/Resources/  
startosinstall --applicationpath "/Applications/Install\ macOS\ High\ Sierra.app"  
--rebootdelay 30 --nointeraction $4
```



(Delete before

The screenshot shows a Jamf Pro policy configuration screen. At the top, it says "Install macOS High Sierra (In-Place) Ongoing". Below that, there are two steps listed:

1. Install macOS-High-Sierra-10.13.4-Installer.pkg
2. Run Script macOS High Sierra Install

A pink box on the right side of the screen contains the text "Leave area free of critical content on all slides".

Now, for machines already in the jamf server we can utilize the start os install binary from the High Sierra installer with a policy. First you push out the installer app then run a script that will take care of the install process with start os install.

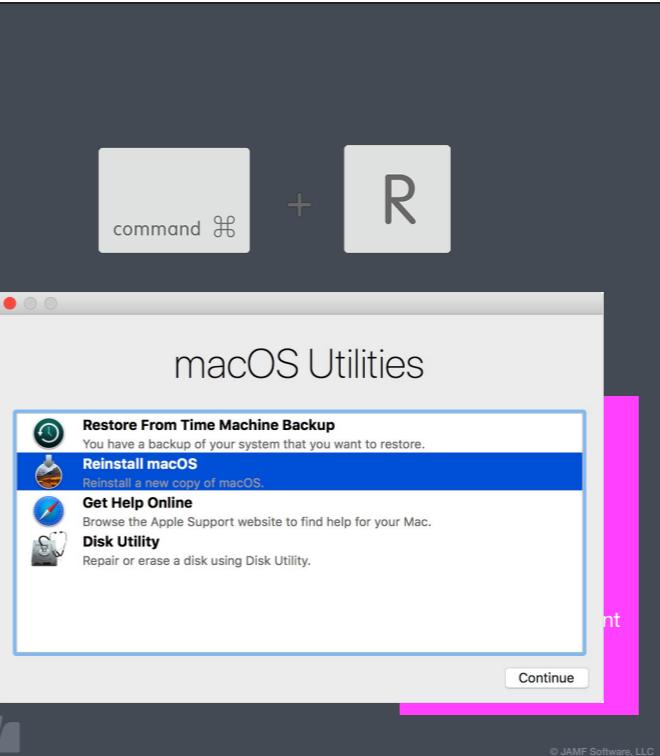
C Note the \$4 flag, we can use that for additional flags such as erase install or convert to apfs.

This is nice because if you have machines that are APFS compatible, you can make a self service policy for your techs to wipe the drive and re-install a clean OS (we do this) The example above is just using a custom trigger.

Preparation...

Existing Machines (HFS)

- Internet Recovery!



For machines that are either not in the jamf server or you cannot use the “erase install” command you should just boot to internet recovery with good ol’ command R.

C which of course brings up the utilities and you can go from there.

The Process...

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

So let's go back to our steps...we just prepped our machines and next we should deploy.

The Process...

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

So deployment is simply booting the machine to get the mobile config and install the software we need based on the machine type... But what if you don't know who is getting the machine or where it's going? How are you going to figure out what software gets installed before assignment?

We don't assign the machine until the next step..

C

The Process...

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deployment

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)

Assignment

- Rename machine and assign to user
- Enter Asset Tag
- Give to user
- Enjoy a drink

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

Well, maybe we can do both at the same time... _C_

The Process...

Preparation

- New machines get added to DEP then assigned to jamf
- Old machines get wiped via internet recovery or policy

Deploy and Assign

- Boot machines to Setup Assistant
- Install Mobile Config
- Install software based on cohort (machine type)
- Rename machine and assign to user
- Enter Asset Tag

Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

Which is what we did.

So, after we prepare a machine, that machine can either go into storage until it's ready to be deployed (which happens often for us) or it can be deployed and assigned right away.

This will also lower the amount of steps needed to deploy the machine, then assign it, then rename the machine and add the user account, etc. We can take care of everything at one time.

The Process...

Enrollment Trigger

- Install DEPNotify
- App Package
- Logo
- Provisioning Script

Run Script to do things!

- Install Software
- Assign computer to user in Jamf Pro
- Create local account
- Rename computer
- Install updates

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

So here is our enrollment complete process. Again, sorry for the wall of text.

We'll Install DEPNotify with extra payload items then run a script. That script will do the things!

Sounds great, right? It was except..

But...we ran into issues...

- Ran behind the login window
 - Added a “wait for dock” loop
- Ran before user was completely logged in
 - Added timer
 - Still was not running every time...
- Launch Daemon!

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

We ran into issues.

C First we found it was running behind the login window. Things would work, but nothing would show up which was obviously bad.

C So we added a “wait for dock loop”. This worked about 60% of the time because the next issue was

C It started running before the user was completely logged in. _C_ So we added a timer.

C This too didn’t run every time, we would have a bunch of cases where it would again run but it be in the background or what if we had a tech close the laptop at the login window..then you would have the script waiting to run in the background.

So we decided to solve that with _C_ a launch daemon!

The Process...

Enrollment Trigger

- Install DEPNotify
- App Package
- Logo
- Launch Daemon
- Deployment Script

Launch Daemon runs Script

- Install Software
- Assign computer to user in jamf
- Create local account
- Rename computer
- Install updates

(Delete before presenting)

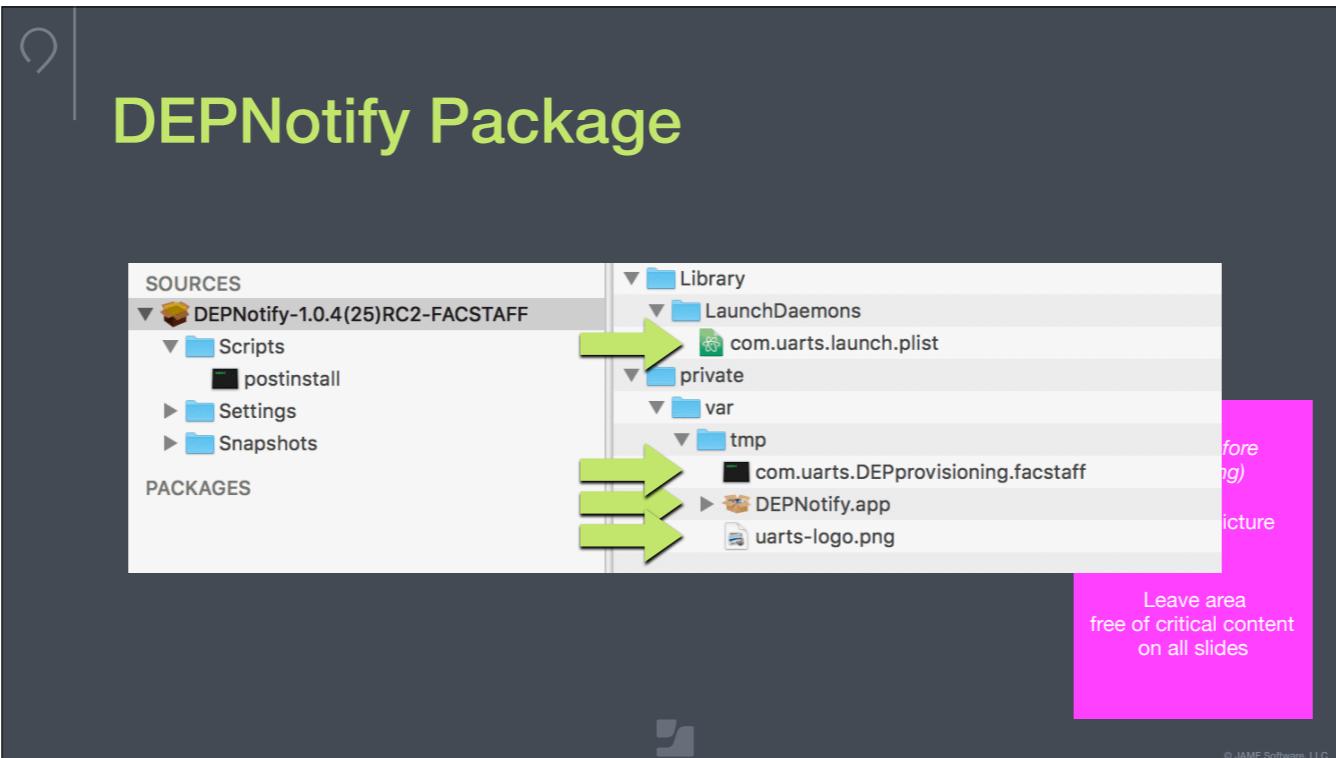
Picture-in-picture area.

Leave area free of critical content on all slides

So we just updated our enrollment step with these _C_

We're still installing DEP with a payload but we're going to add the deployment script and launch daemon.

DEPNotify Package



So here is our package.

C Here is the app in /var/tmp

C Our logo

C And our provisioning script

That script gets called by _C_ our launch daemon.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3  <plist version="1.0">
4  <dict>
5    <key>GroupName</key>
6    <string>wheel</string>
7    <key>InitGroups</key>
8    <false/>
9    <key>Label</key>
10   <string>com.uarts.launch</string>
11   <key>Program</key>
12   <string>/var/tmp/com.uarts.DEPprovisioning.facstaff</string>
13   <key>RunAtLoad</key>
14   <true/>
15   <key>StartInterval</key>
16   <integer>10</integer>
17   <key>UserName</key>
18   <string>root</string>
19   <key>StandardErrorPath</key>
20   <string>/var/tmp/deponotify.launch.err</string>
21   <key>StandardOutPath</key>
22   <string>/var/tmp/deponotify.launch.out</string>
23 </dict>
24 </plist>
```

(Delete before presenting)

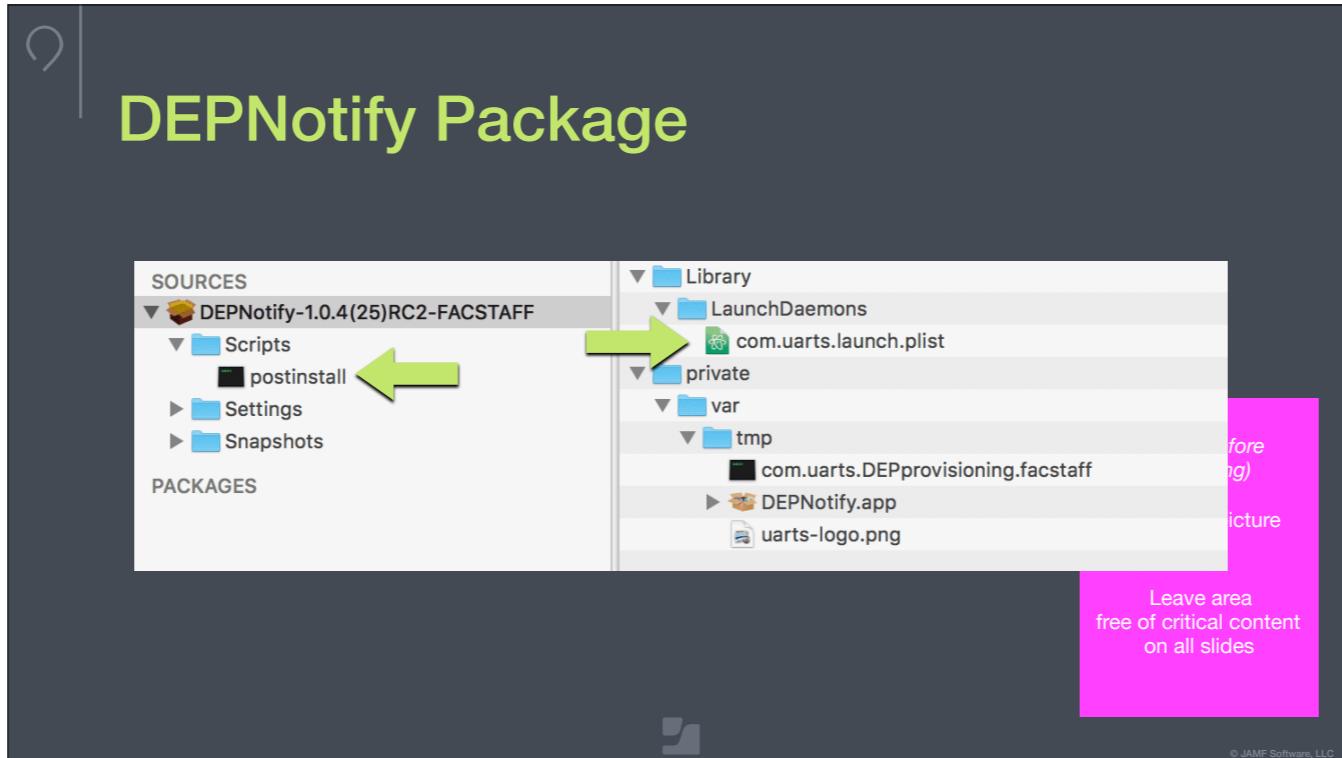
Picture-in-picture area.

Leave area of critical content on all slides

This is our launch daemon here. Pretty straight forward, it loads _C_ the provisioning script 10 seconds after login is complete or when it's loaded.

It will also run if the enrollment trigger occurs while the user is logged in, we have this issue quite often.

DEPNotify Package



Now that we have that there, we have to load it and that is what
C our post install script is for.

```
1 #!/bin/sh
2 ## postinstall
3
4 echo "disable auto updates ASAP" >> /var/log/jamf.log
5 /usr/sbin/softwareupdate --schedule off
6
7 # Disable diagnostic data
8 SUBMIT_DIAGNOSTIC_DATA_TO_APPLE=FALSE
9 SUBMIT_DIAGNOSTIC_DATA_TO_APP_DEVELOPERS=FALSE
10
11 ## Make the main script executable
12 echo "setting main script permissions" >> /var/log/jamf.log
13 chmod a+x /var/tmp/com.uarts.DEPprovisioning.facstaff.sh
14
15 ## Set permissions and ownership for launch daemon
16 echo "set LaunchDaemon permissions" >> /var/log/jamf.log
17 chmod 644 /Library/LaunchDaemons/com.uarts.launch.plist
18 chown root:wheel /Library/LaunchDaemons/com.uarts.launch.plist
19
20 ## Load launch daemon into the Launchd system
21 echo "load LaunchDaemon" >> /var/log/jamf.log
22 launchctl load /Library/LaunchDaemons/com.uarts.launch.plist
23
24 exit 0 ## Success
25 exit 1 ## Failure
26
```

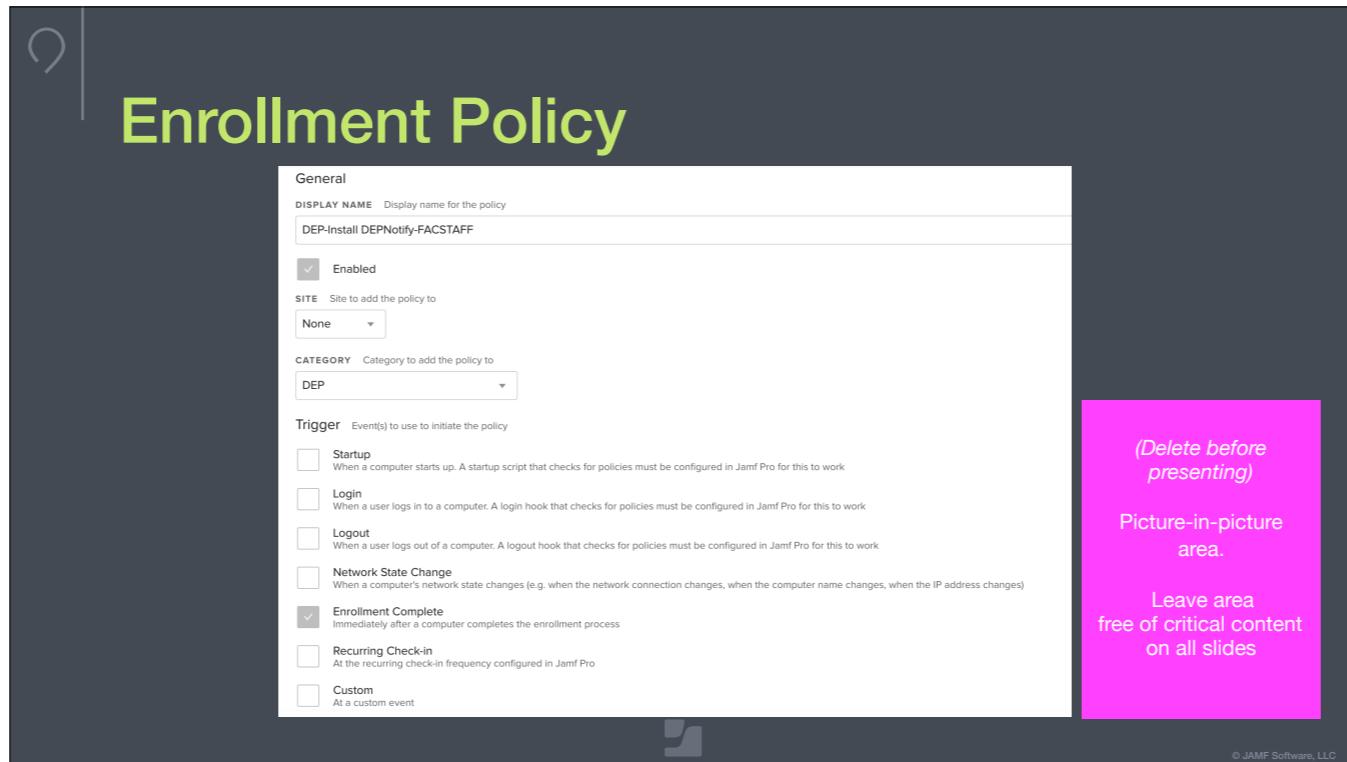
(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

Again, pretty straight forward. Most of this is just extra stuff to disable any auto updates, the most important line is here _C_ just loading our launch daemon.

Now our package is done..



Create our enrollment policy. Trigger is enrollment complete and we're just installing our custom PKG.
C

Enrollment Policy

The screenshot shows a software interface for managing computer packages. On the left, there's a sidebar with a gear icon and the text "Enrollment Policy". The main area is titled "Packages". It displays a single package entry: "DEPNotify-1.0.4(25)RC2-FACSTAFF.pkg". Below the package name, there's a dropdown menu labeled "ACTION" with the option "Install" selected. Underneath the action menu, there's a checkbox labeled "Update Autorun data" with the sub-instruction "Add or remove the package from each computer's Autorun data". At the bottom of the package card, there's a small "JAMF" logo. To the right of the package card is a pink rectangular area containing the following text:

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

At the very bottom right of the slide, there's a small copyright notice: "© JAMF Software, LLC".

C

The Process...

Enrollment Trigger

- Install DEPNotify
- App Package
- Logo
- Launch Daemon
- Deployment Script

Launch Daemon runs Script

- Install Software
- Assign computer to user in jamf
- Create local account
- Rename computer
- Install updates

(Delete before presenting)

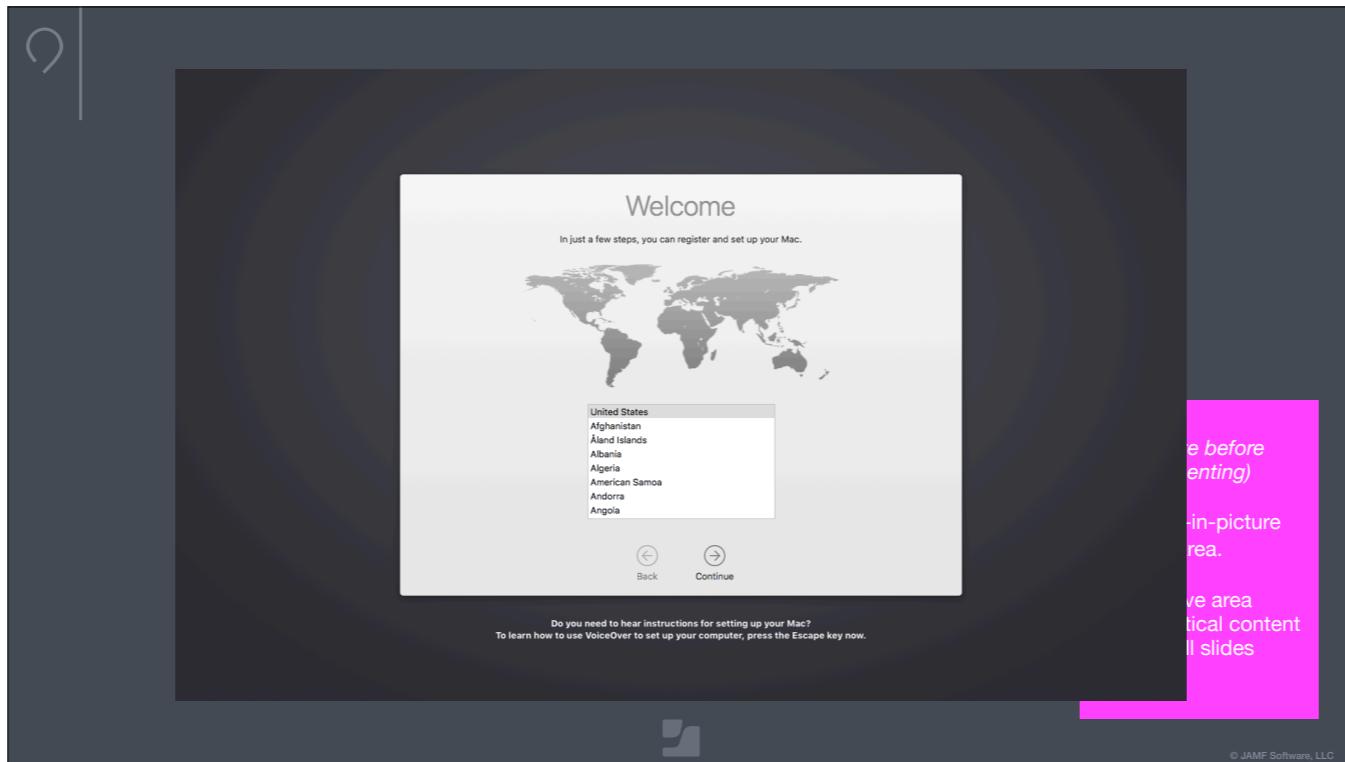
Picture-in-picture area.

Leave area free of critical content on all slides

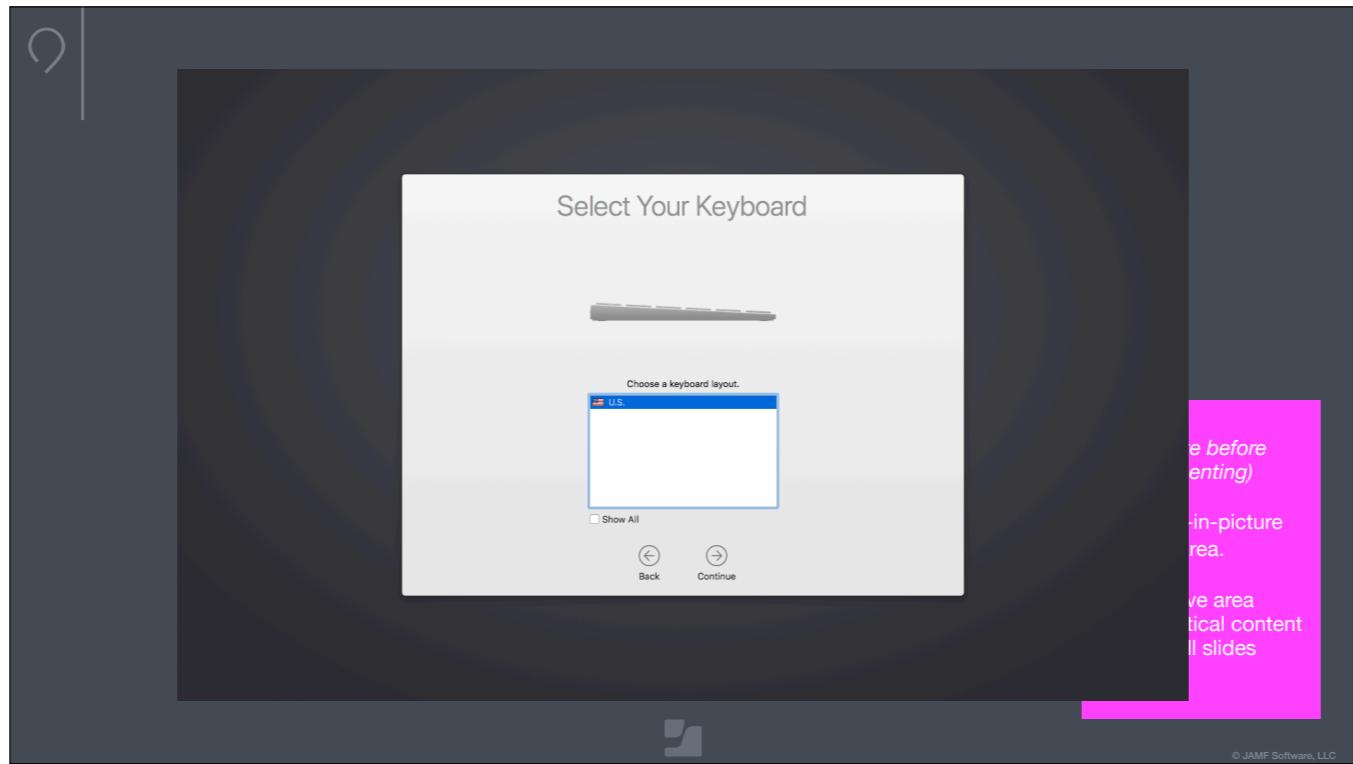
So once DEPNotify is installed the only thing left to do is _C_

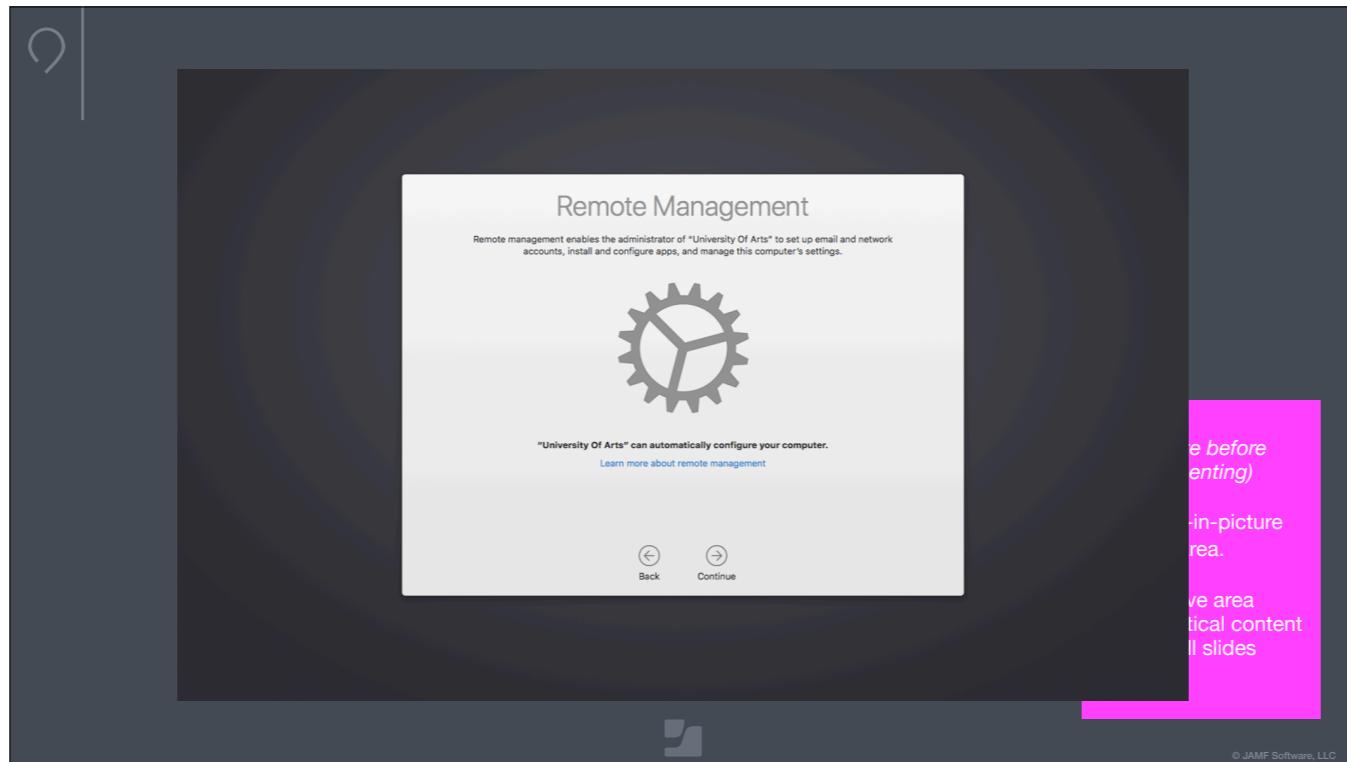
Have our launch daemon run our provisioning script.

Let's see that in action.

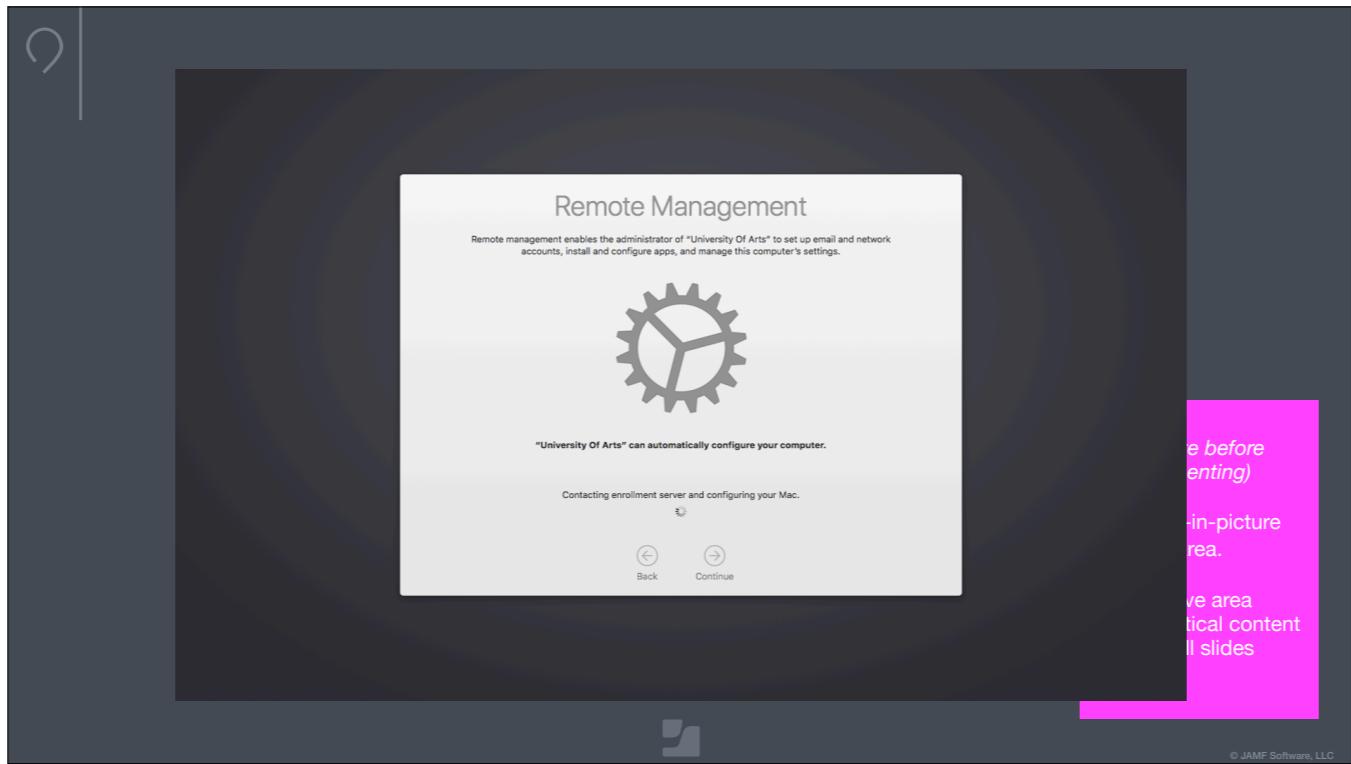


Okay, after we boot and we go through the setup assistant (unfortunately).

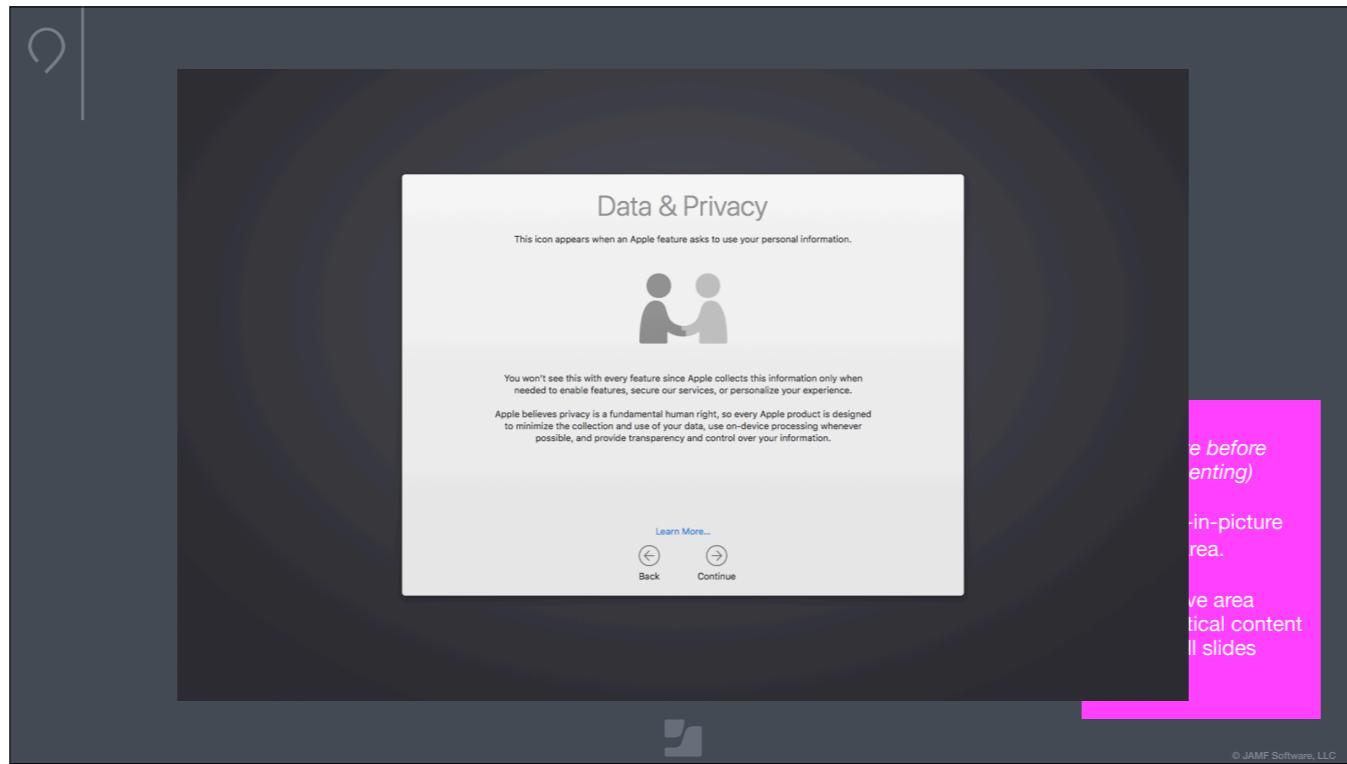




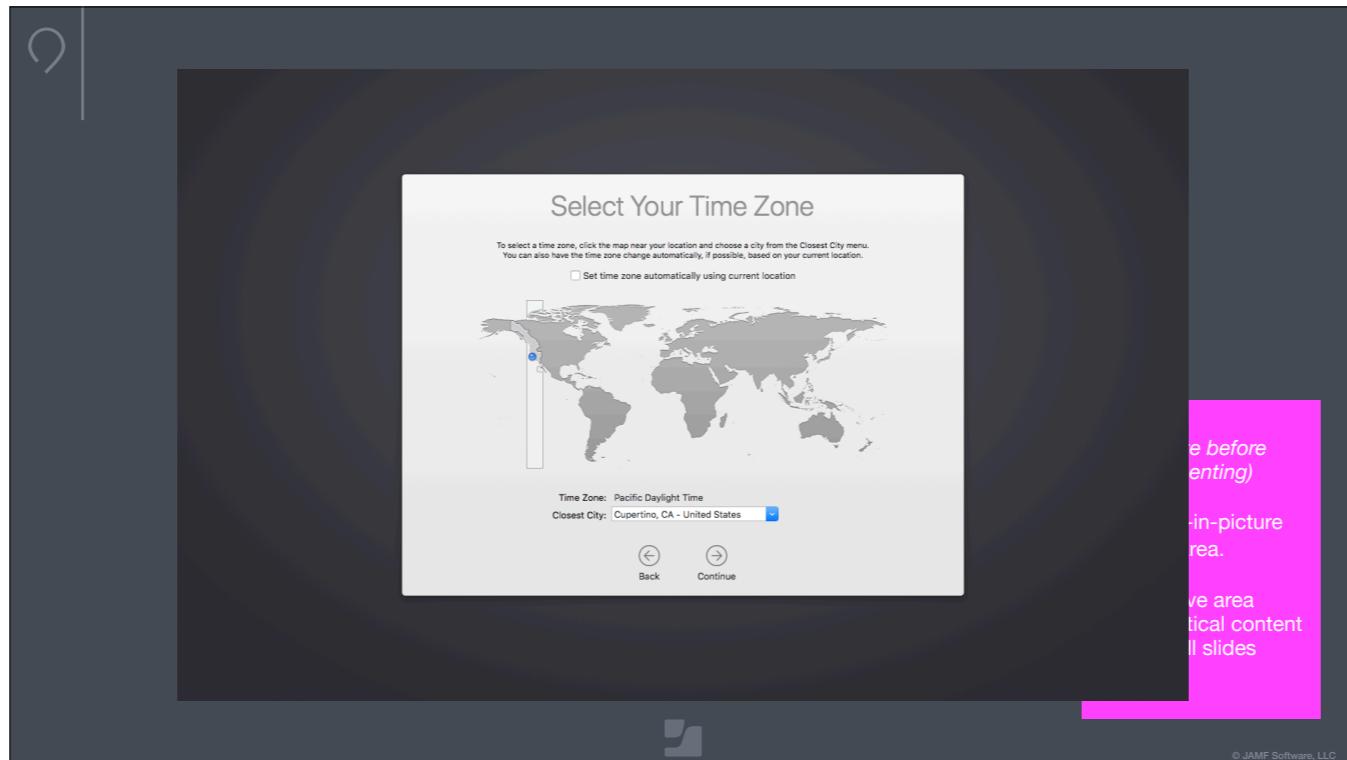
Now, we know this screen, it means we've got a DEP machine that's tied to a server and we're ready to roll!
Clicking continue...



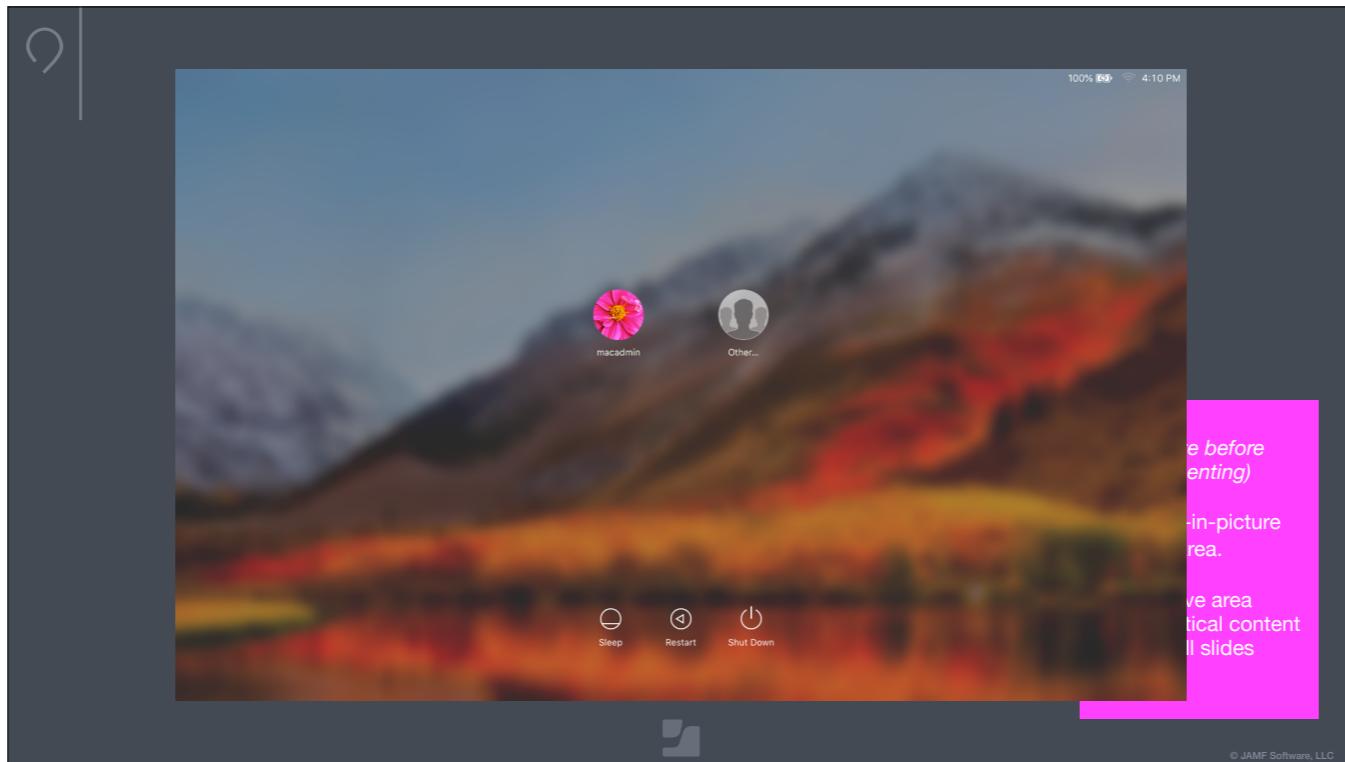
The machine gets configured from our prestage... and



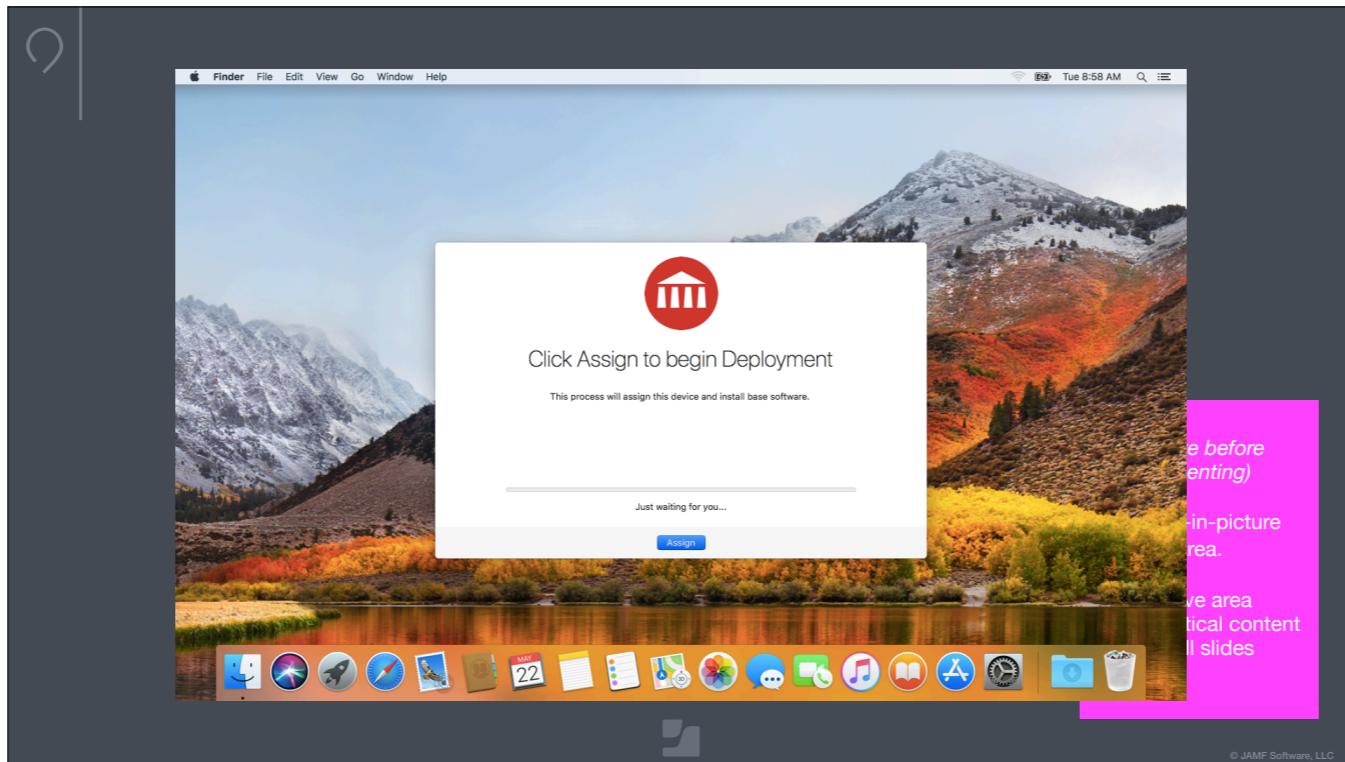
Right..another click..



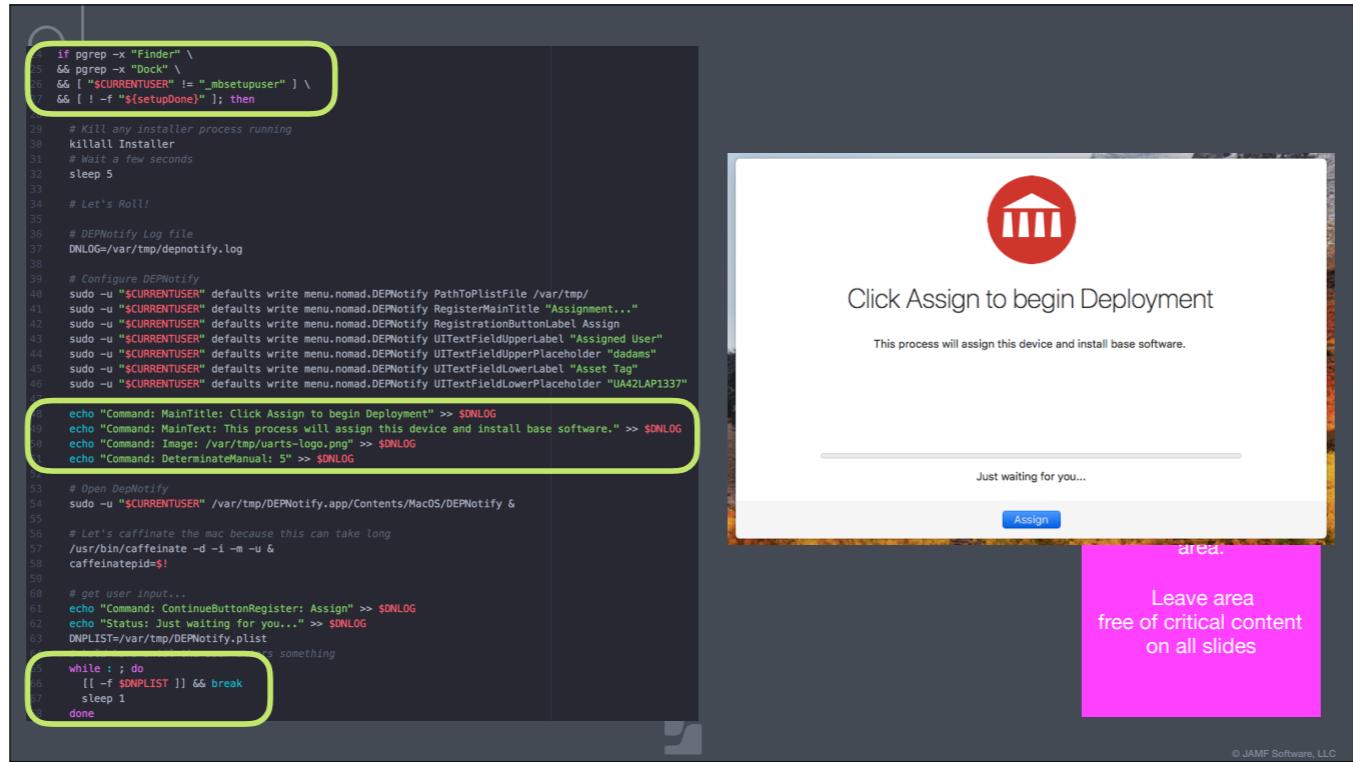
And another..



Finally!
Notice our macadmin account is there and it didn't ask for any authentication.
So we'll log in.



And this is what we're greeted to after login or after the enrollment policy runs AFTER logging in, our DEPNotify window. Let's see it with our code side-by side.

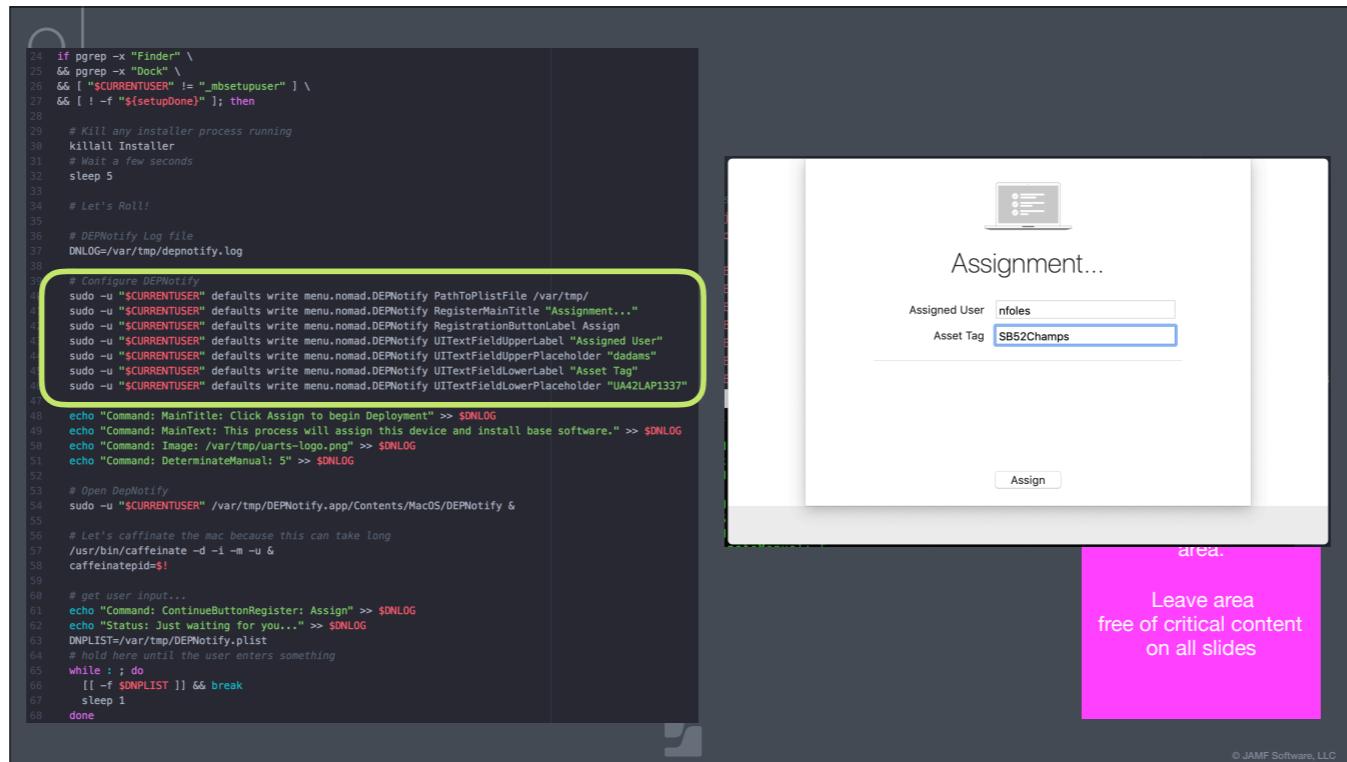


Here's our code. Now, DEPNotify was only launched because C. We waited for the dock and finder to be loaded and made sure we're not in the mbsetupuser account... we also are checking for this setup done file, more on that later.

C The window looks the way it does because we sent these commands to the deponify log file. Then we launched it AFTER sending that to the log.

Now the interesting thing is that this is now just waiting for the user to hit the "Assign" button. I set it up this way so our tech's don't have to wait around until it's done installing things, nothing will happen until they start. So that's down here C.

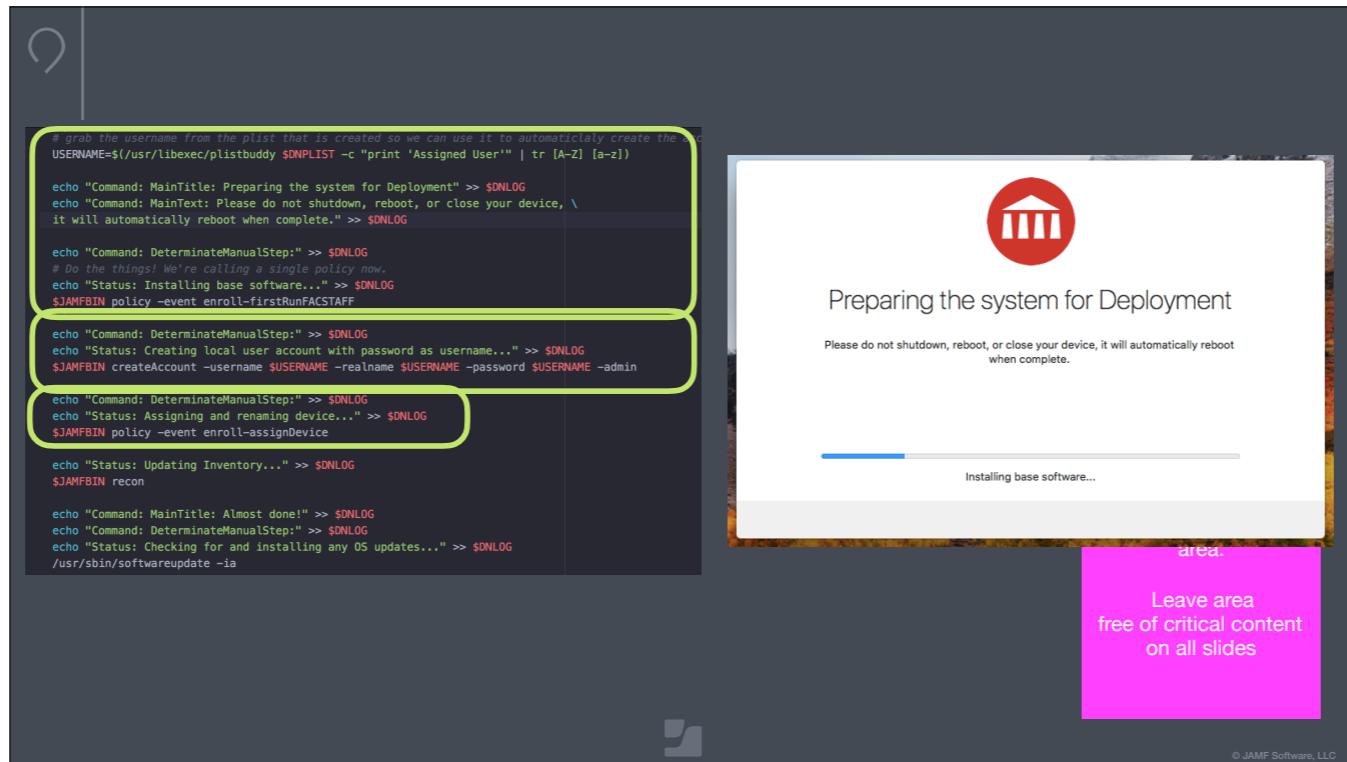
It's just a while loop waiting for the "DNPLIST" which is a file that gets dropped after input is complete. Which is next up...



When the tech clicks “Assign” the registration window drops down.

That is populated by this _C_ section of code. Notice that the code was run before DEPNotify was open because it’s actually just setting some prefs up. You can also create a config file for this, but we liked just coding it in.

Now, we only need two text fields, but you can also add a dropdown menu or two and a “sensitive data” checkbox. New features are being added all of the time so there are probably other things you can do now.



So after the user input is done we can begin the tasks.

C First we're grabbing the username from the plist that was created after registration and holding it for creating the local account later. Then we're passing more commands into the DEPNotify log file, so now our window has changed. And then we run our first run policy.

C After the policy is run we are creating the local account based on the assigned user and setting the password to the username (temporarily).

C Then we're going to assign and rename the device with another policy. This policy runs a script...

```

41 SERIAL=$(ioreg -rd1 -c IOPlatformExpertDevice | awk -F'\" /IOPlatformSerialNumber/{print $4}')
42 MODEL=$(system_profiler SPHardwareDataType | awk '/Model Identifier/ {print $3}')
43
44 DNPLIST=/var/tmp/DEPNotify.plist
45 USERNAME=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Assigned User'" | tr [A-Z] [a-z])
46 ASSETTAG=$(/usr/libexec/plistbuddy $DNPLIST -c "print 'Asset Tag'" | tr [A-Z] [a-z])
47
48 # Create xml
49 cat << EOF > /var/tmp/tempInfo.xml
50 <computer>
51   <general>
52     <asset_tag>$ASSETTAG</asset_tag>
53   </general>
54   <location>
55     <username>$USERNAME</username>
56   </location>
57 </computer>
58 EOF
59 ## Upload the xml file
60 /usr/bin/curl -sfku "${APIUSER}:${APIPASS}" "${JSSURL}JSSResource/computers/serialnumber/$SERIAL" -X PUT -T /var/tmp/tempInfo.xml
61
62 if echo "$MODEL" | grep -q 'MacBookAir'
63 then
64   PREFIX='MBA'
65 elif echo "$MODEL" | grep -q 'MacBookPro'
66 then
67   PREFIX='MBP'
68 else
69   echo "No model identifier found."
70   PREFIX=''
71 fi
72
73 # rename the computer
74 COMPUTERNAME="${USERNAME}-${PREFIX}"
75 COMPUTERNAME=$(echo ${COMPUTERNAME}:0:15)
76 /usr/local/jamf/bin/jamf setComputerName -name $COMPUTERNAME
77
78 # update our extension attribute
79 mkdir /Library/JAMF\ DM
80 mkdir /Library/JAMF\ DM/ComputerName
81 chflags hidden /Library/JAMF\ DM
82 echo $COMPUTERNAME > /Library/JAMF\ DM/ComputerName/ComputerName.txt
83 rm -rf /var/tmp/tempInfo.xml
84
85

```

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

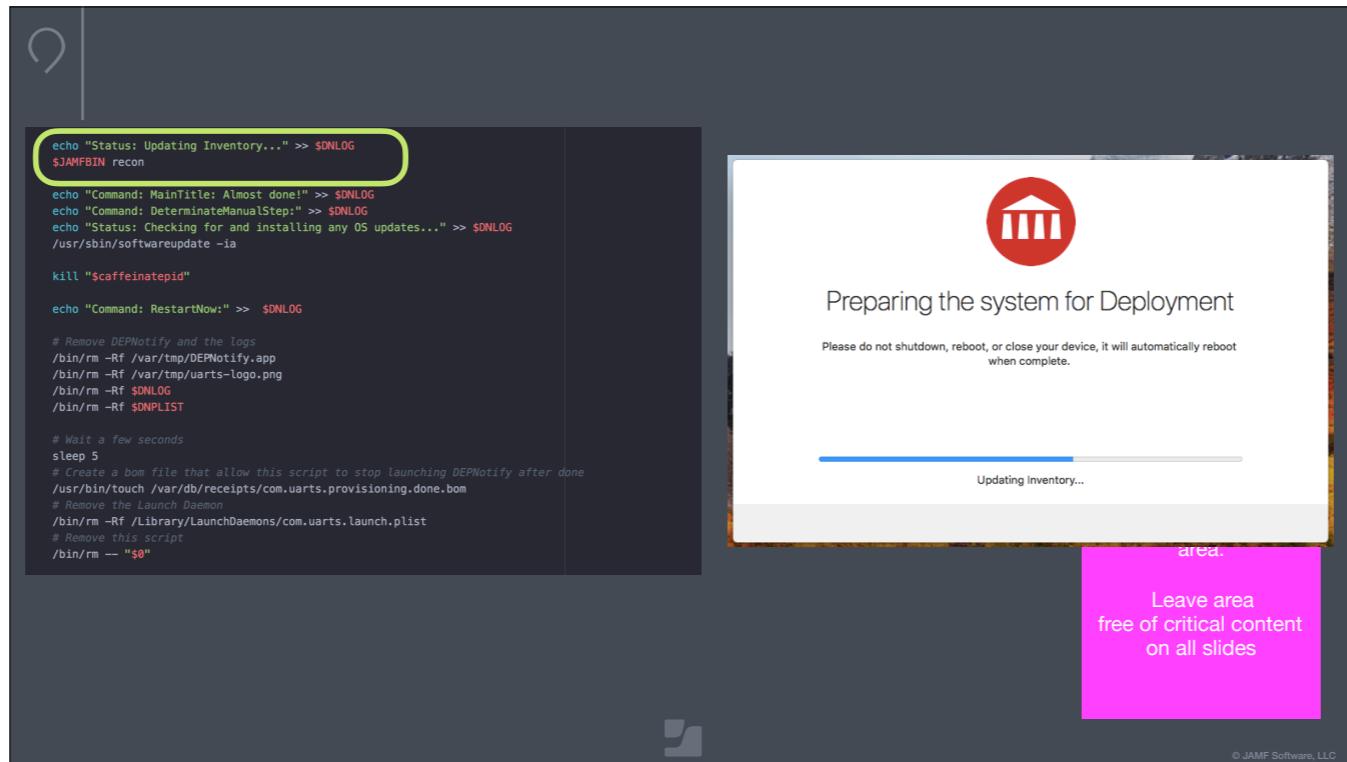
It just reads _C_ the plist file that was dropped at registration and sets variables for user and asset tag.

C sets up an xml file and passes it to the jamf API

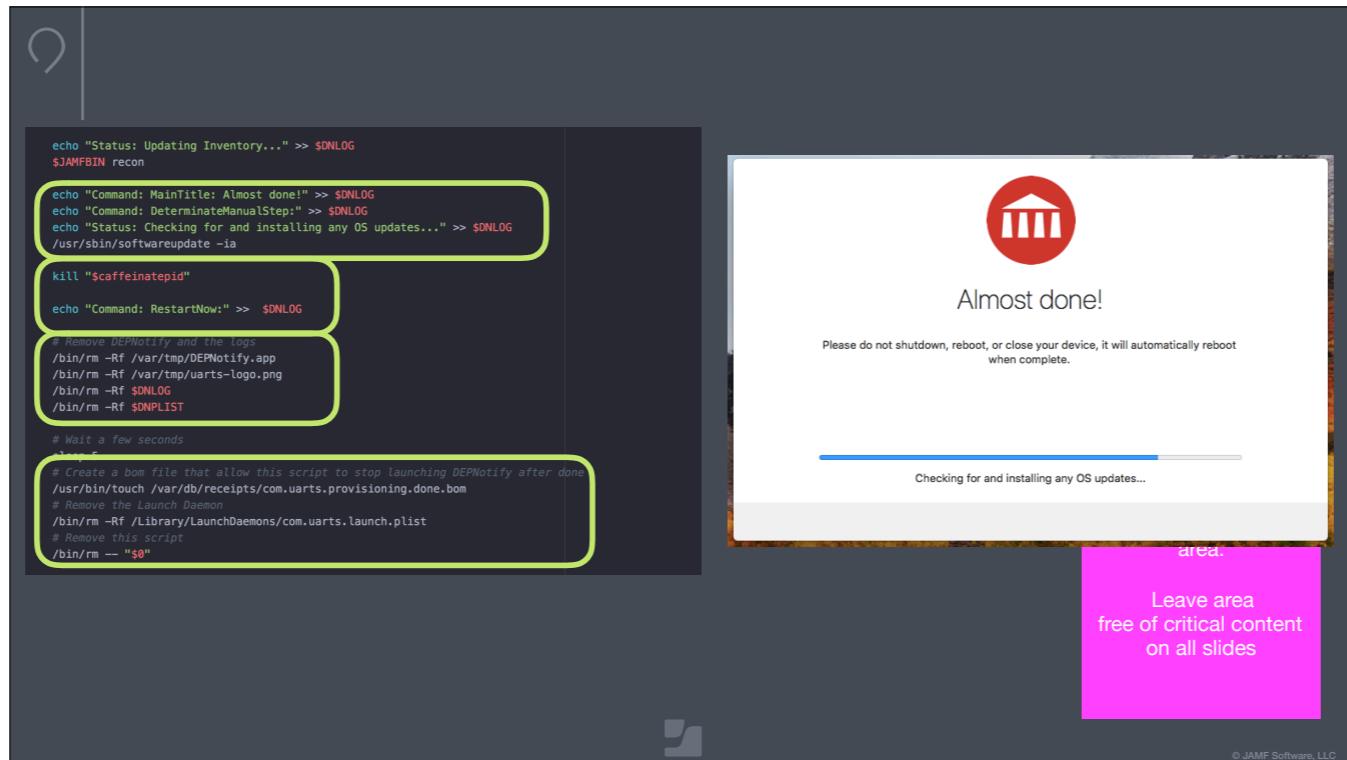
C then it renames the machine using the assigned username with a prefix for MBA or MBP.

C Finally, we update our extension attribute receipts on the machine and remove the temp XML file.

If you've ever done any work with the jamf API, this is all pretty straightforward.



Back in our provisioning script, well, at the bottom of it, we run a recon to get all of that new info into our server



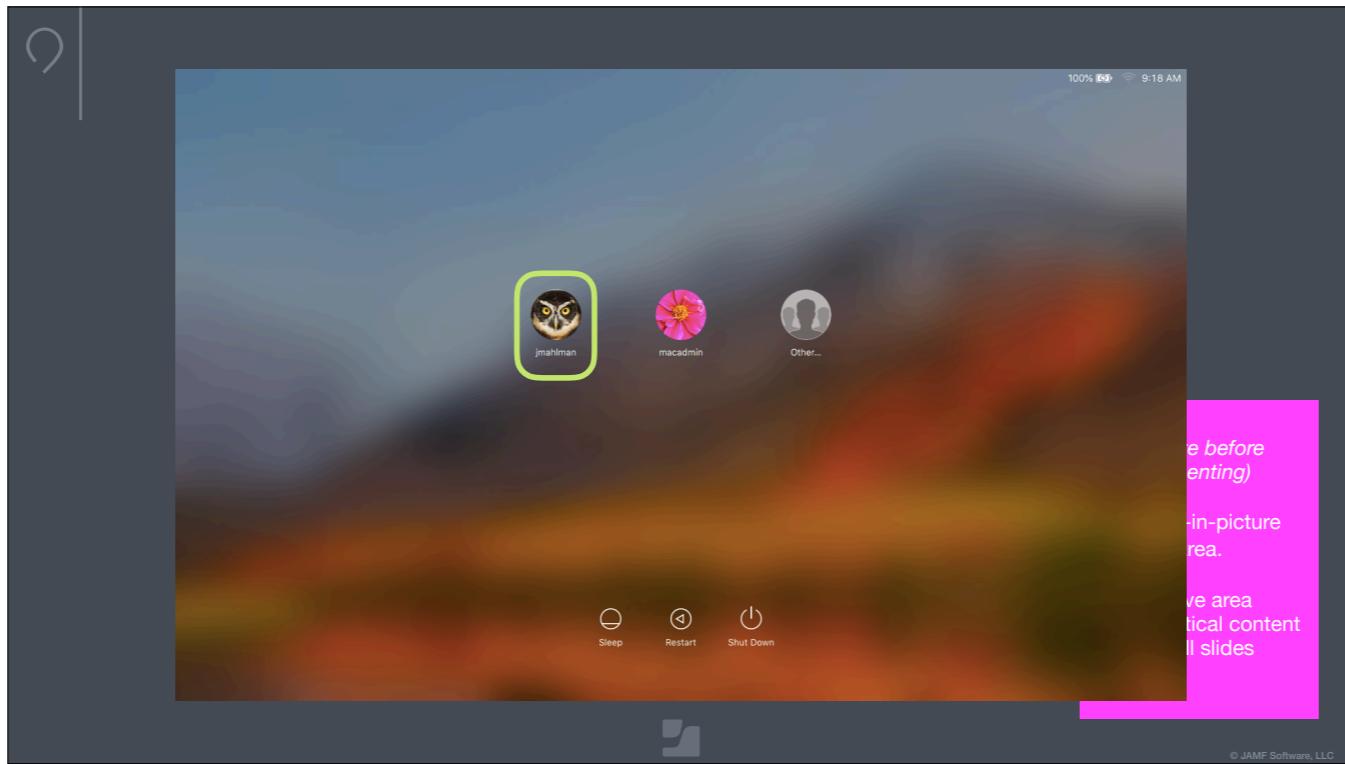
And next we run software updates..we're also changing the text in our DEPNotify window just to tell the tech that it's almost done.

C After software updates install we send the command to DEPNotify to restart the machine right away, you can ask the user to hit "OK" to reboot if you'd like, there is also a quit or logout action.

C Right as that happens we clean up the files that we installed (DEPNotify, our logo, and some of our logs).

C Finally, we create the setup done bom file to tell us that this ran and if the launchdaemon doesn't get removed for some reason it won't accidentally run provisioning again. It's also very useful if the enrollment triggers again for some reason. We are also removing the launch daemon here.

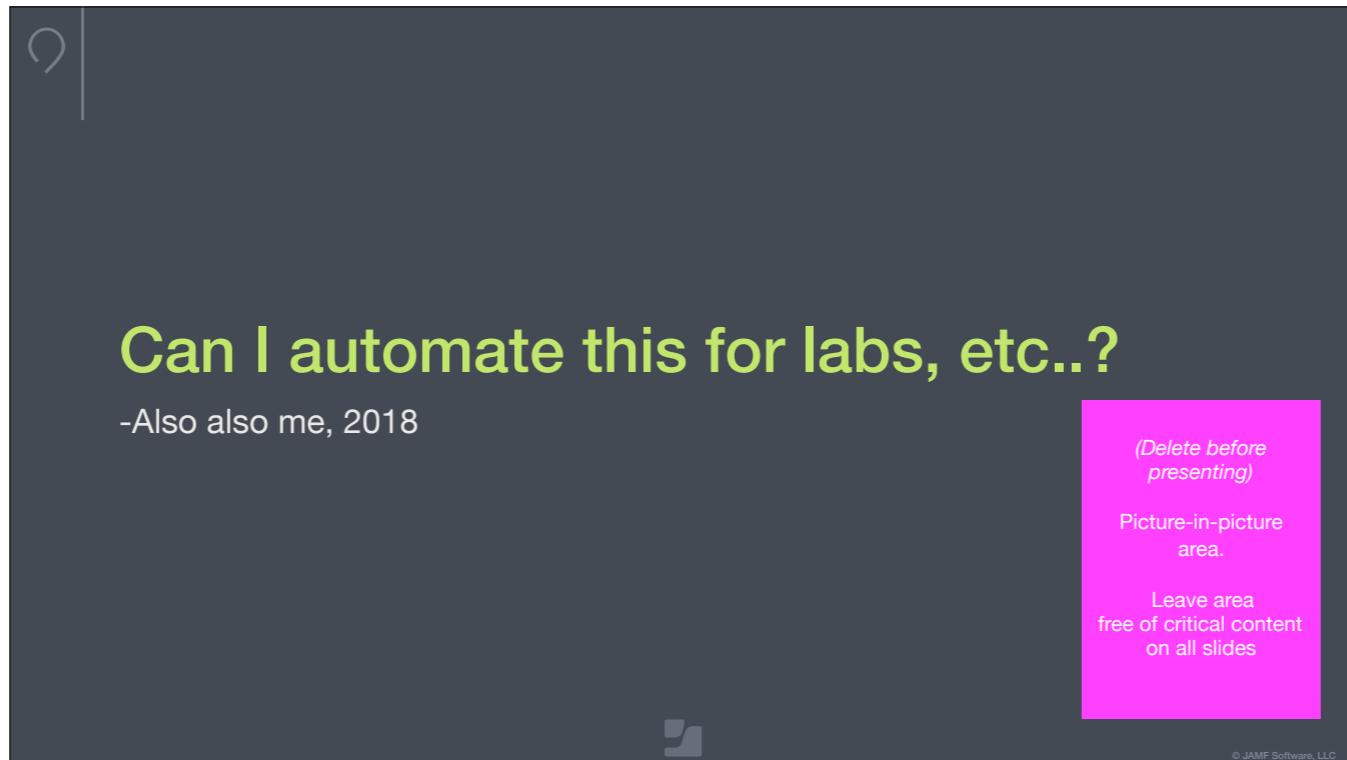
Note that we're not unloading it, we had issues with trying to unload it so I decided to just to remove it since the machine was going to reboot anyway.



Reboot complete and

C we now have our new local user and everything installed.

I was so happy this worked but It still had a lot of steps..so I began to think...



Can I automate this?

Well, around the exact same time I was working on this process I came across Neil Martin's jamf nation road show presentation where he did exactly what I wanted..so I decided to try it for myself in our environment so... _C_

HECK YEAH!

-Me, 3 months ago

Neil Martin's JNRS presentation:
<https://github.com/neilmartin83/Jamf-Nation-Roadshow-London-2018>

*(Delete before
presenting)*

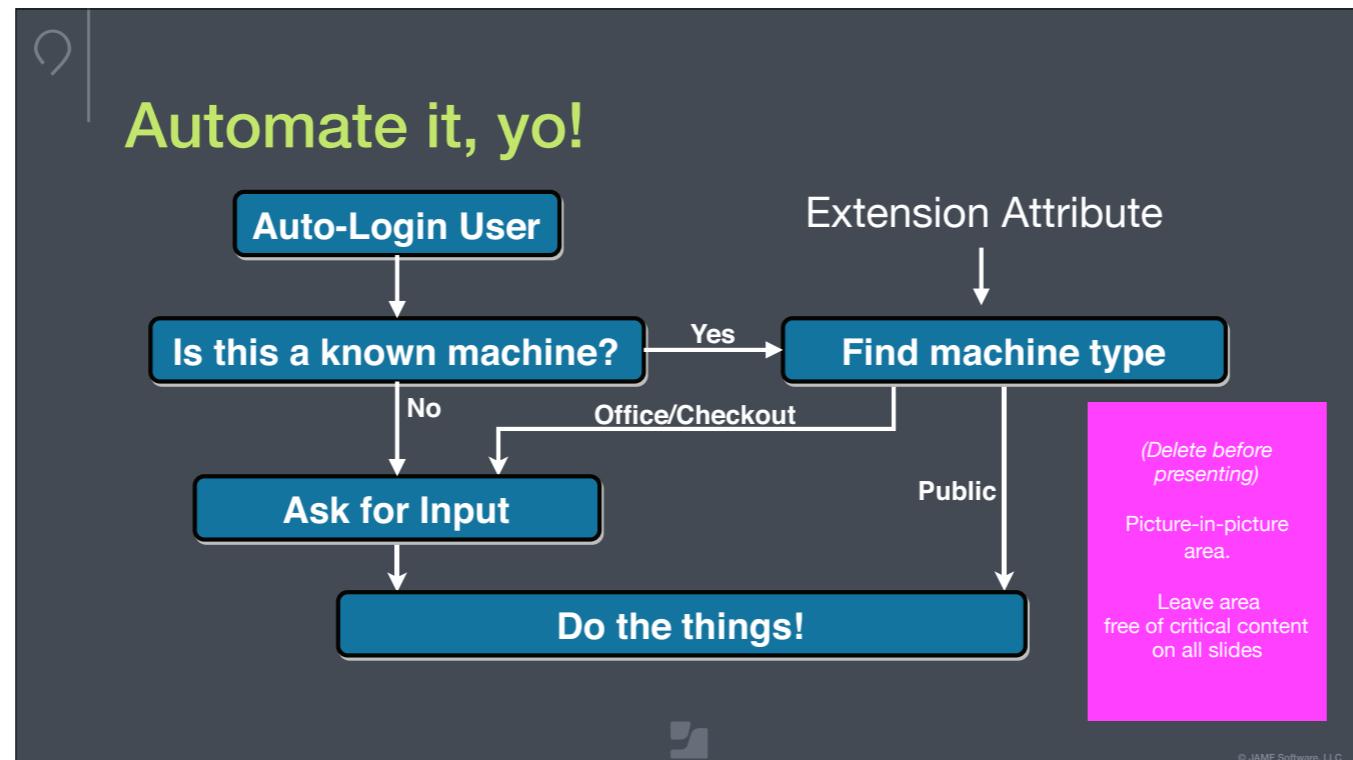
Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

With scripting we can do anything!

(Thanks Neil!)



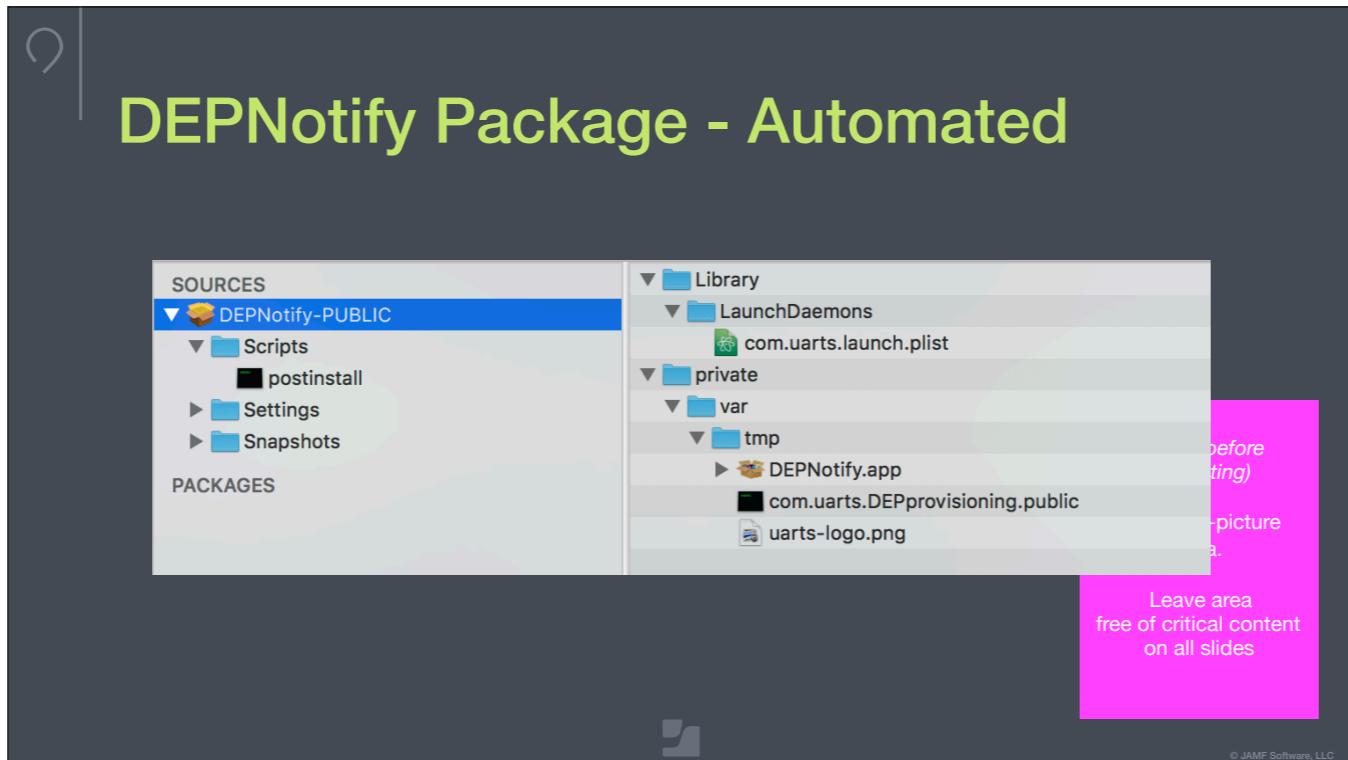
This was my ::ahem:: flow of how I wanted things to work. I like flow charts.

I wanted the machine to auto-login so DEPNotify would actually open, see if it's a “known machine” to the server and if it was, what kind? Office, lab, checkout laptop...

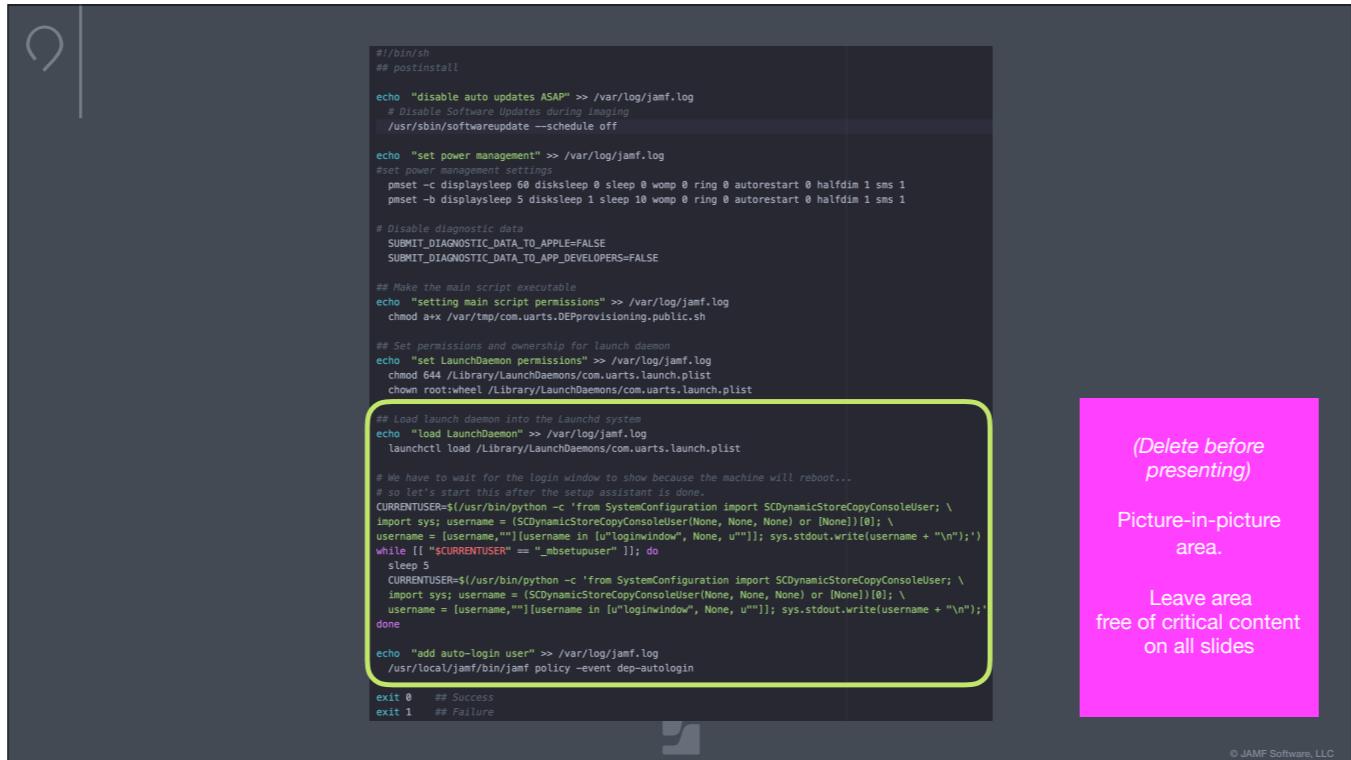
If it was not known to the server or if it was an “Office” or “Checkout” machine I wanted it to always ask the user for input, then do everything. If it’s a public machine, just do everything.

We find out the machine type `_C_` with a text extension attribute that we fill at image/deployment time and have on the machines in a hidden receipt which we call a Cohort. It never changes unless we specifically change it. Every machine in our environment has a Cohort.

DEPNotify Package - Automated



So let's make our package. It's the same package except the post install script has some changes...



```

#!/bin/sh
## postinstall

echo "disable auto updates ASAP" >> /var/log/jamf.log
# Disable Software Updates during imaging
/usr/sbin/softwareupdate --schedule off

echo "set power management" >> /var/log/jamf.log
#set power management settings
pmset -c displaysleep 60 disksleep 0 sleep 0 womp 0 ring 0 autorestart 0 halfdim 1 sms 1
pmset -b displaysleep 5 disksleep 1 sleep 10 womp 0 ring 0 autorestart 0 halfdim 1 sms 1

# Disable diagnostic data
SUMMIT_DIAGNOSTIC_DATA_TO_APPLE=FALSE
SUMMIT_DIAGNOSTIC_DATA_TO_APP_DEVELOPERS=FALSE

## Make the main script executable
echo "setting main script permissions" >> /var/log/jamf.log
chmod a+x /var/tmp/com.uarts.DEPProvisioning.public.sh

## Set permissions and ownership for launch daemon
echo "set LaunchDaemon permissions" >> /var/log/jamf.log
chmod 644 /Library/LaunchDaemons/com.uarts.launch.plist
chown root:wheel /Library/LaunchDaemons/com.uarts.launch.plist

## Load launch daemon into the Launchd system
echo "load LaunchDaemon" >> /var/log/jamf.log
launchctl load /Library/LaunchDaemons/com.uarts.launch.plist

# We have to wait for the login window to show because the machine will reboot...
# so let's start this after the setup assistant is done.
CURRENTUSER=$(/usr/bin/python -c 'from SystemConfiguration import SCDynamicStoreCopyConsoleUser; \
import sys; username = [SCDynamicStoreCopyConsoleUser(None, None, None) or [None]][0]; \
username = [username,""]るusername in ["loginwindow", None, u""]]; sys.stdout.write(username + "\n");')
while [[ "$CURRENTUSER" == "__bsetupuser" ]]; do
    sleep 5
    CURRENTUSER=$(/usr/bin/python -c 'from SystemConfiguration import SCDynamicStoreCopyConsoleUser; \
import sys; username = [SCDynamicStoreCopyConsoleUser(None, None, None) or [None]][0]; \
username = [username,""]るusername in ["loginwindow", None, u""]]; sys.stdout.write(username + "\n");')
done

echo "add auto-login user" >> /var/log/jamf.log
/usr/local/jamf/bin/jamf policy -event dep-autologin

exit 0    ## Success
exit 1    ## Failure

```

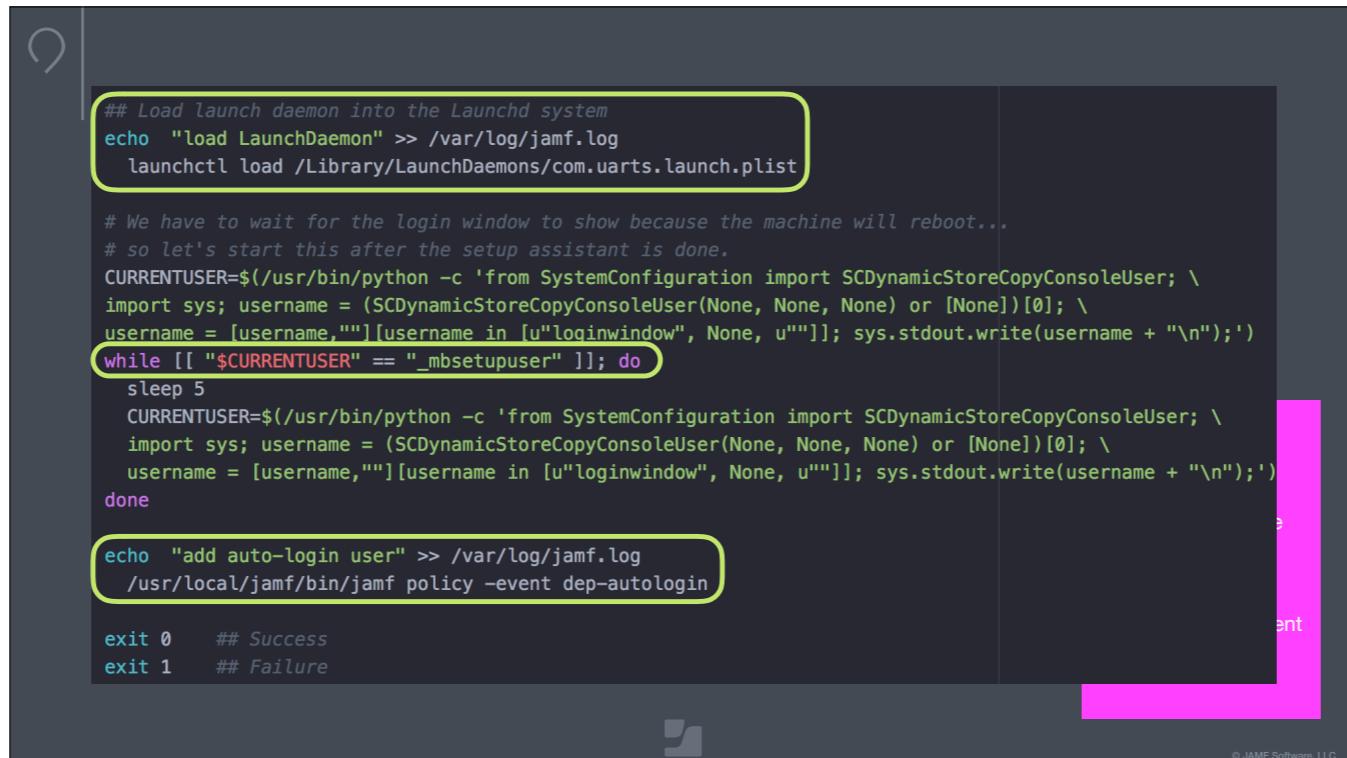
(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

It's mainly the same except for this _C_ section here... _C_



```
## Load launch daemon into the Launchd system
echo "load LaunchDaemon" >> /var/log/jamf.log
launchctl load /Library/LaunchDaemons/com.uarts.launch.plist

# We have to wait for the login window to show because the machine will reboot...
# so let's start this after the setup assistant is done.
CURRENTUSER=$(/usr/bin/python -c 'from SystemConfiguration import SCDynamicStoreCopyConsoleUser; \
import sys; username = (SCDynamicStoreCopyConsoleUser(None, None, None) or [None])[0]; \
username = [username,""]的文化 in [u"loginwindow", None, u""]]; sys.stdout.write(username + "\n");')
while [[ "$CURRENTUSER" == "_mbsetupuser" ]]; do
    sleep 5
    CURRENTUSER=$(/usr/bin/python -c 'from SystemConfiguration import SCDynamicStoreCopyConsoleUser; \
import sys; username = (SCDynamicStoreCopyConsoleUser(None, None, None) or [None])[0]; \
username = [username,""]文化 in [u"loginwindow", None, u""]]; sys.stdout.write(username + "\n");')
done

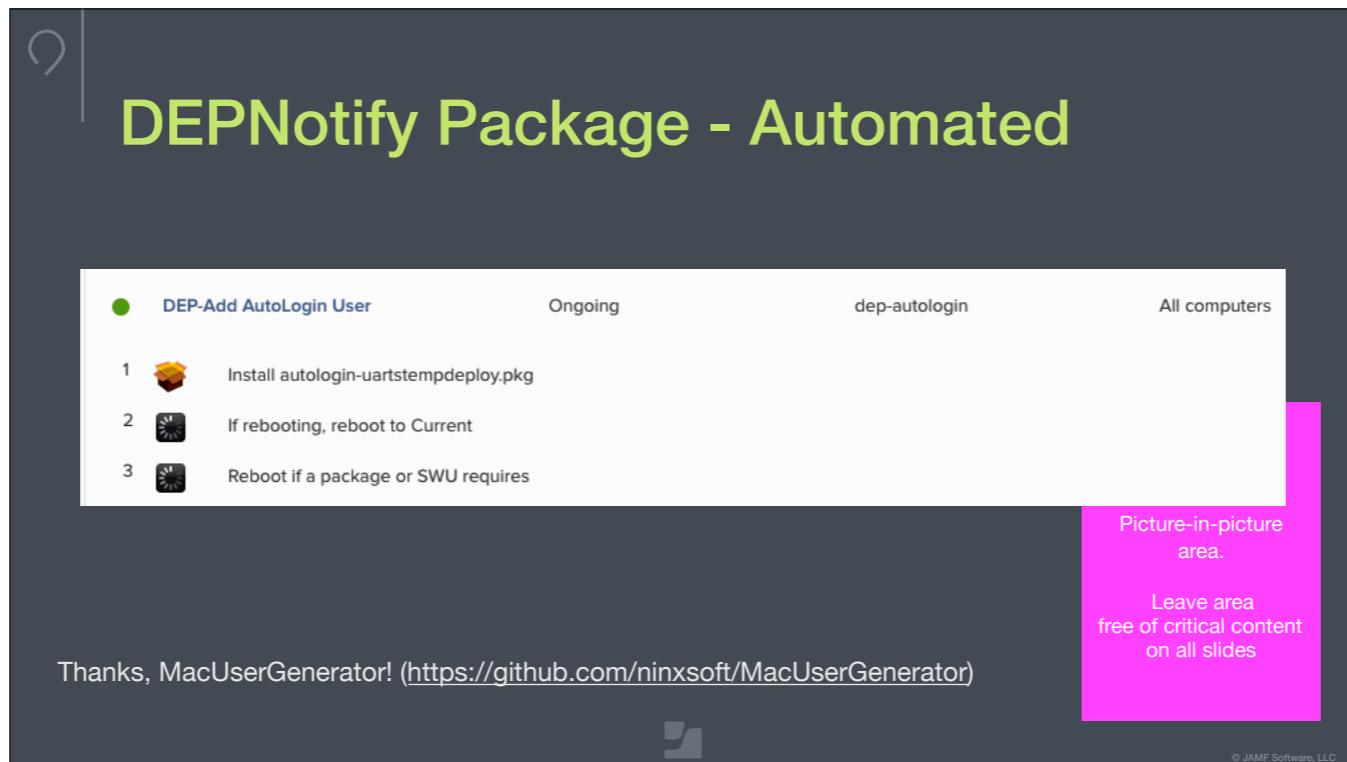
echo "add auto-login user" >> /var/log/jamf.log
/usr/local/jamf/bin/jamf policy -event dep-autologin

exit 0 ## Success
exit 1 ## Failure
```

We're still loading our launch daemon C but this ugly portion below is doing us a big favor.

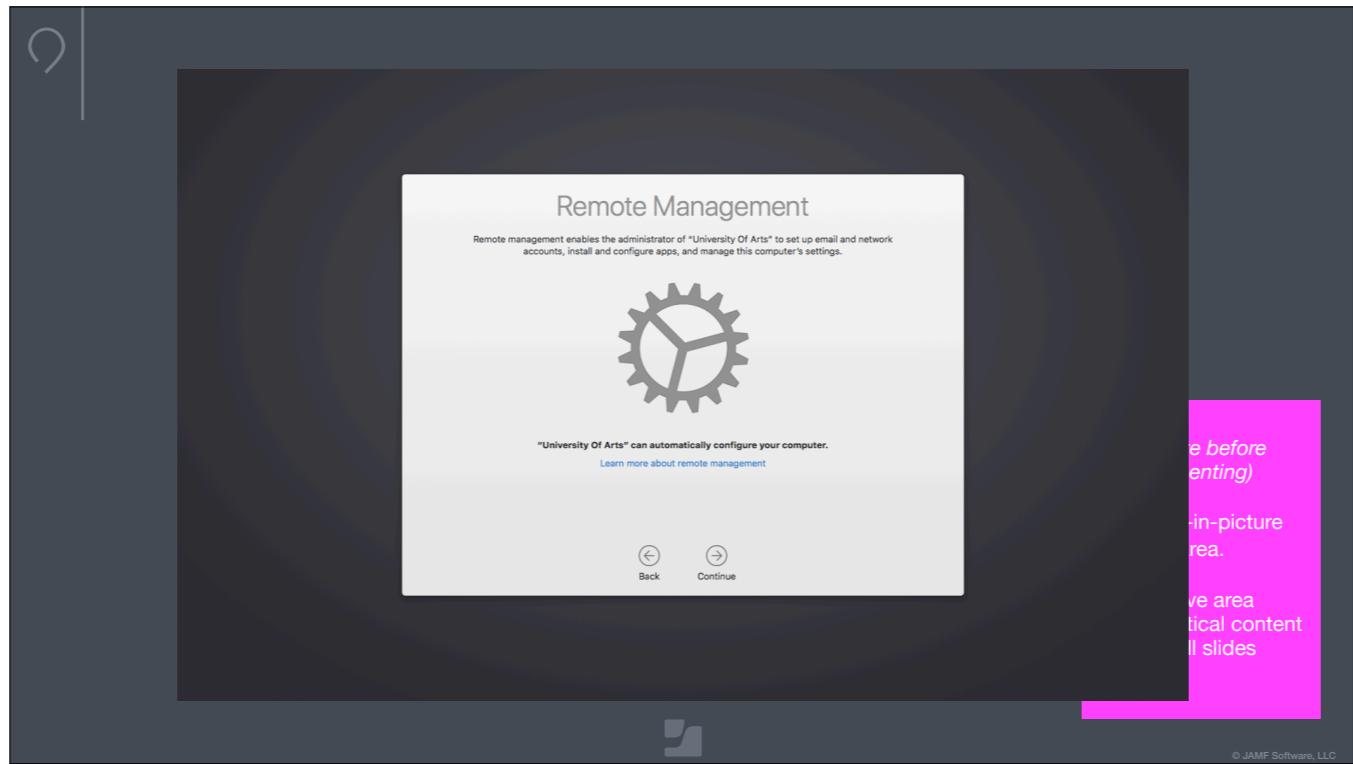
C While we're in the _mbsetupuser we don't want the next line to run. That's because this line C is calling a policy that installs a package to create an auto-login user and automatically reboots the machine.

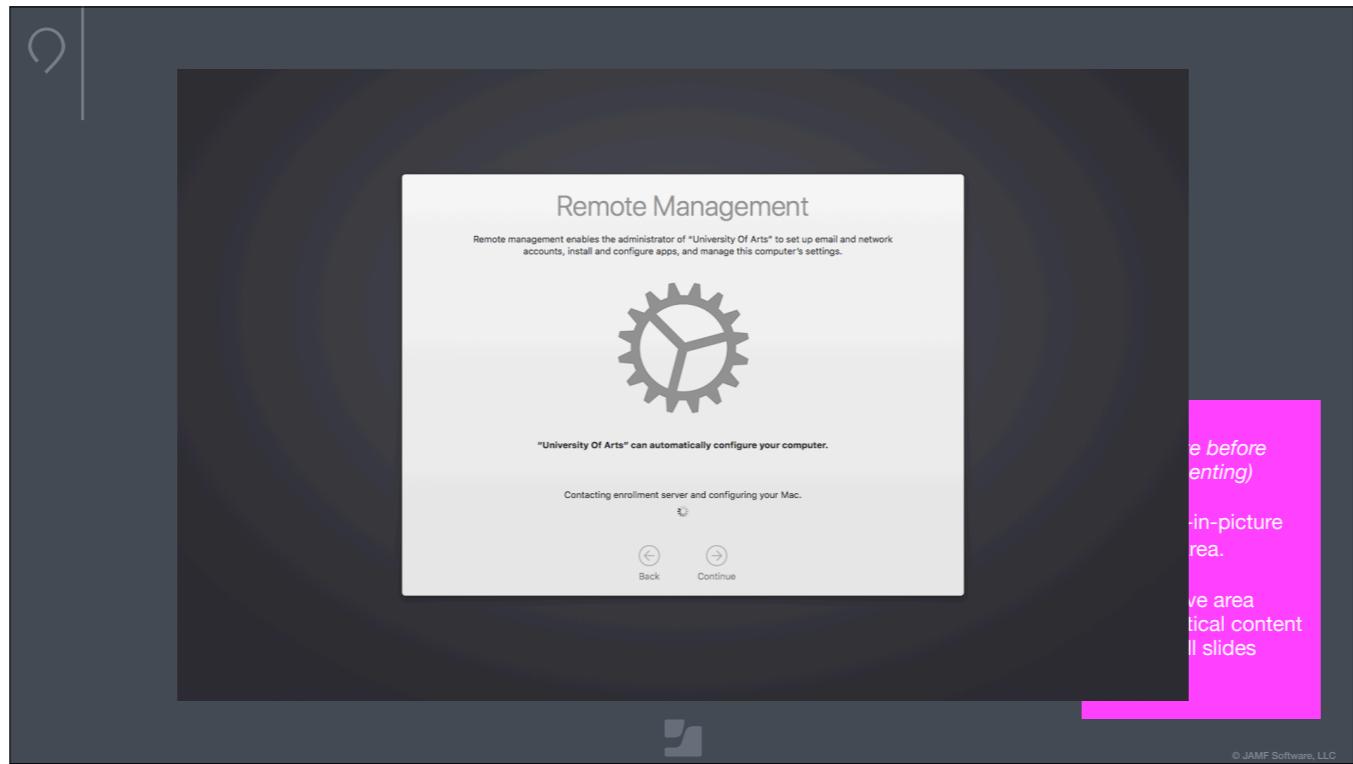
If we don't wait the machine may reboot before you get to the login window and finish setup assistant. Not the end of the world, but we didn't want that.

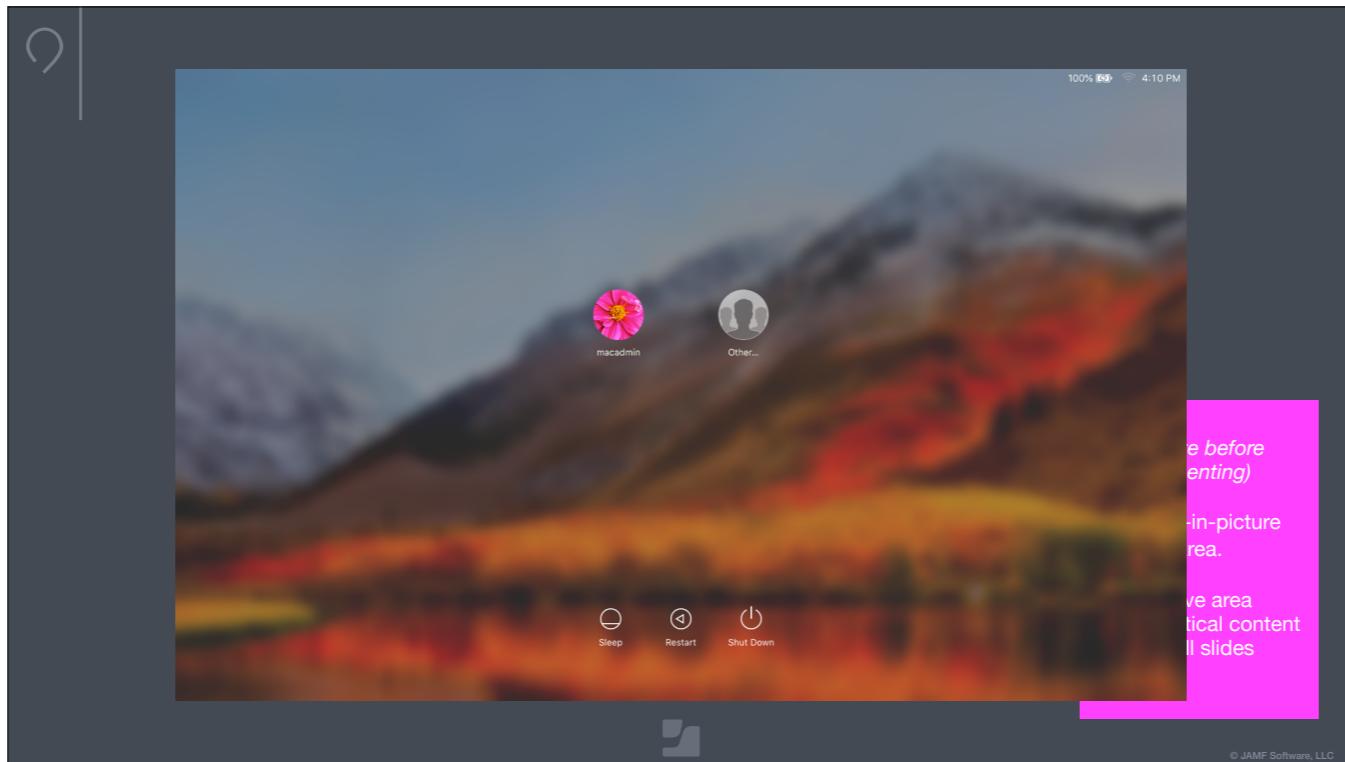


This is the auto-login user policy. It's just installing a package created with MacUserGenerator which is an app based on CreateUserPackage.

So now that all of that's done..let's see that in action!







Now we're back at the login screen and the machine will automatically reboot here and auto login to the temp account we created. When it reboots, our provisioning script will kick off.

```

OSVERSION=$sw_vers -productVersion
serial=$(ioreg -rd1 -c IOPlatformExpertDevice | awk -F'"/' '/IOPlatformSerialNumber/{print $4}')
exml=$(curl "$JSSURL/JSSResource/computers/$serial"/subset/extension_attributes -u "$APIUSER":"$APIPASS" -H "Content-Type: application/xml")
jssMacName=$(echo "$exml" | xpath '//extension_attribute[name="New Computer Name"]' | awk -F'<value>'<"/value>" '{print $2}')
jssCohort=$(echo "$exml" | xpath '//extension_attribute[name="New Cohort"]' | awk -F'<value>'<"/value>" '{print $2}')

JAMFBIN=/usr/local/bin/jamf
# Get the logged in user
CURRENTUSER=$(/usr/bin/python -c 'from SystemConfiguration import SCDynamicStoreCopyConsoleUser; \nimport sys; username = (SCDynamicStoreCopyConsoleUser(None, None, None) or [None])[0]; \nusername = [username,""]るusername in ["logwindow", None, u""]; sys.stdout.write(username + "\n");')
# Setup Done File
setupDone="/var/db/receipts/com.uarts.provisioning.done.bom"

# This is where we wait until a user is completely logged in
if pprep -x "Finder" \
&& pprep -x "Dock" \
&& [ "$CURRENTUSER" != "_mbsetupuser" ] \
&& [ ! -f "$setupDone" ]; then

    # Kill any installer process running
    killall Installer
    # Wait a few seconds
    sleep 5

    # Let's Roll
    # DEPMNotify Log file
    DNLOG=/var/tmp/depmnotify.log

    # If the computer is NOT in the jss or if it's an OFFICE machine
    # we want to get user input because most likely this is being reprovisioned.

    if [[ "$jssMacName" == "" ]] || [[ "$jssCohort" == "" ]] || [[ "$jssCohort" == "OFFICE" ]]; then
        # Configure DEPMNotify registration window
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify PathToPlistFile /var/tmp/
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify RegisterMainTitle "Setup..."
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify RegisterButtonLabel Setup
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldUpperPlaceholder "T1337-M01"
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldUpperLabel "Computer Name"
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldLowerPlaceholder "UAA2DSK1337"
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UITextFieldLowerLabel "Asset Tag"
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UIPopUpMenuUpper -array \
        'Lab' 'Studio' 'Smart-Classroom' 'Office' 'Checkout' 'Kiosk'
        sudo -u "$CURRENTUSER" defaults write menu.nomad.DEPNotify UIPopUpMenuUpperLabel "Cohort"
    fi
fi

```

(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

This is the beginning of the script. First, we want to see if we recognize this machine. C_ Let's zoom in..

```
# Set basic variables
OSVERSION=$(sw_vers -productVersion)
serial=$(ioreg -rd1 -c IOPlatformExpertDevice | awk -F''' '/IOPlatformSerialNumber/{print $4}')
eaxml=$(curl "$JSSURL"JSSResource/computers/serialnumber/"$serial"/subset/extension_attributes -u "$APIUSER":"$APIPASS" -H "/
jssMacName=$(echo "$eaxml" | xpath '/extension_attribute[name="New Computer Name"]' | awk -F'<value>|</value>' '{print $2}')
jssCohort=$(echo "$eaxml" | xpath '//extension_attribute[name="New Cohort"]' awk -F'<value>|</value>' '{print $2}')
```

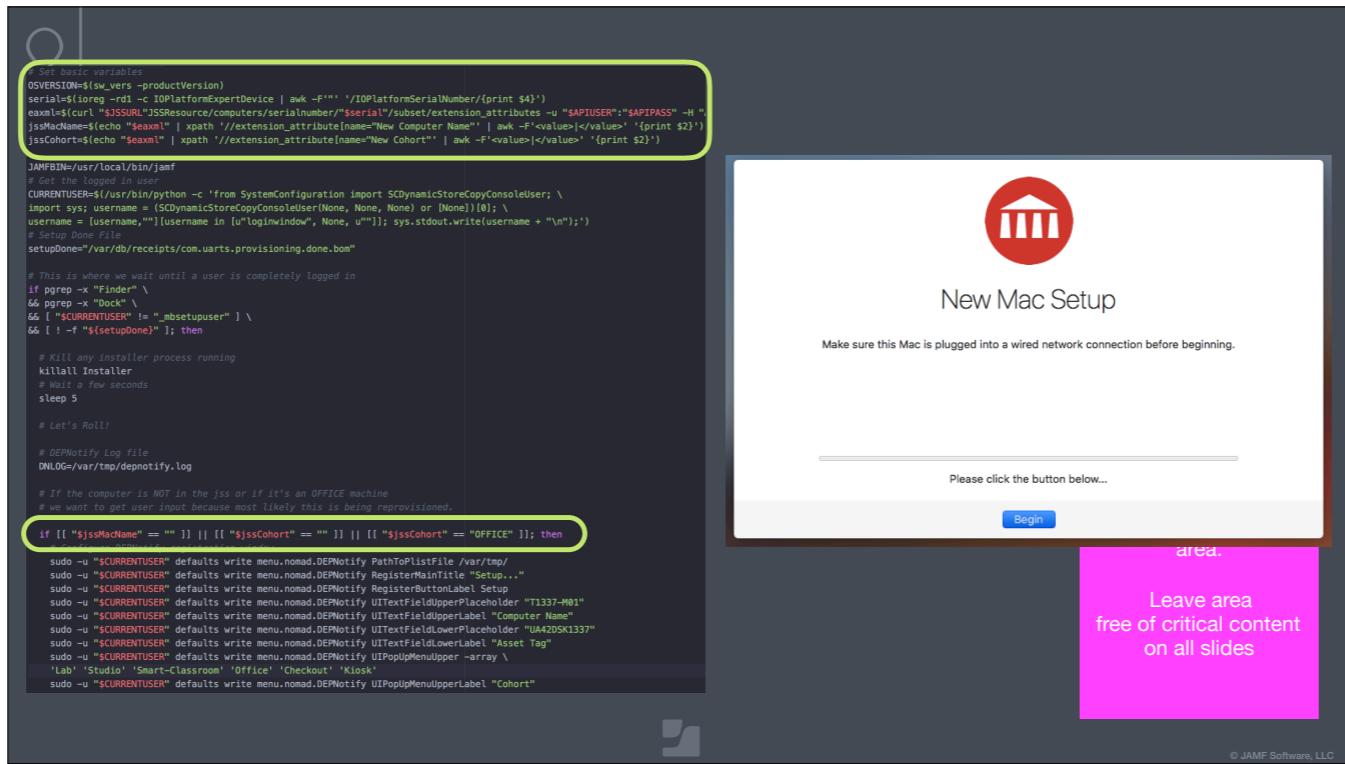
preserung)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

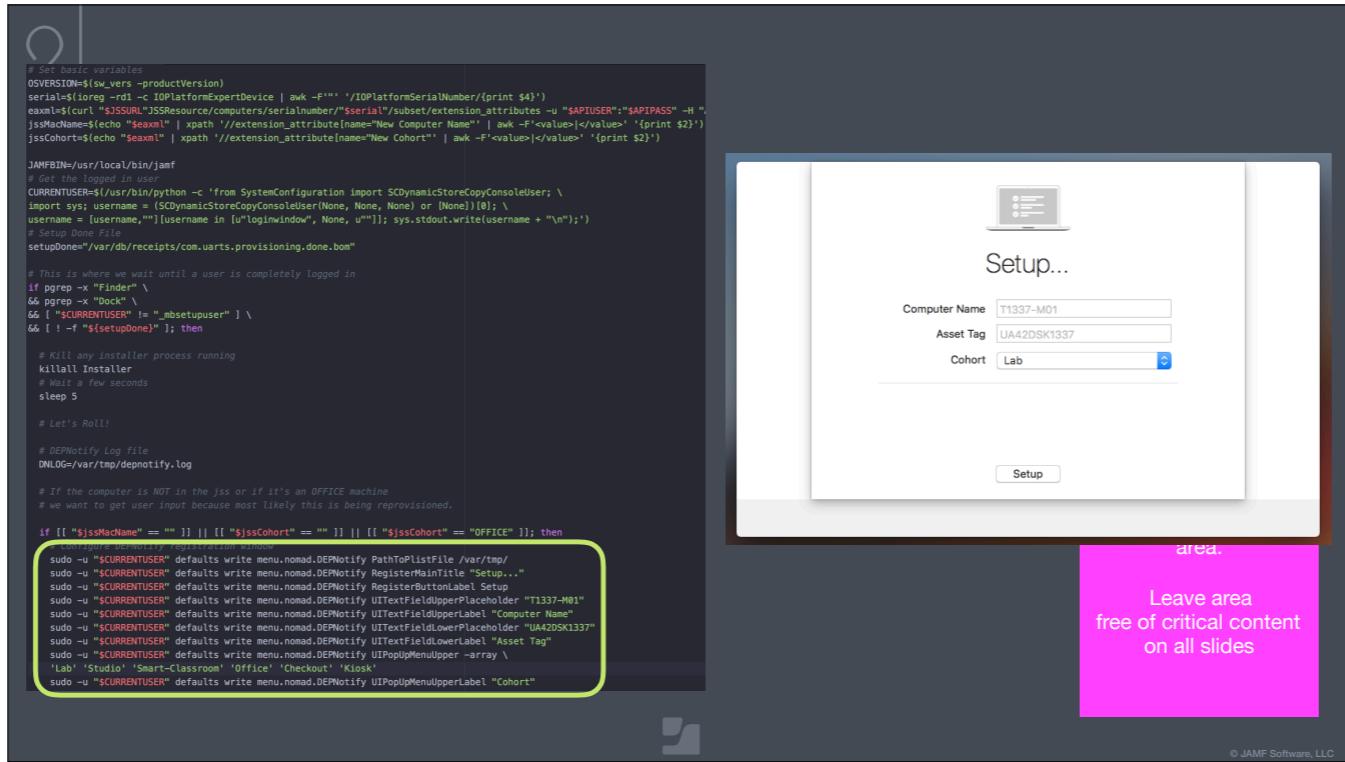
We pull the computer info from the jamf server api and get our Extension attributes _C_ “New Computer name” and _C_ “New Cohort”. The names are just because we have an older extension attribute for each of them and instead of changing that one (since a lot of smart groups use them) we just made new ones.



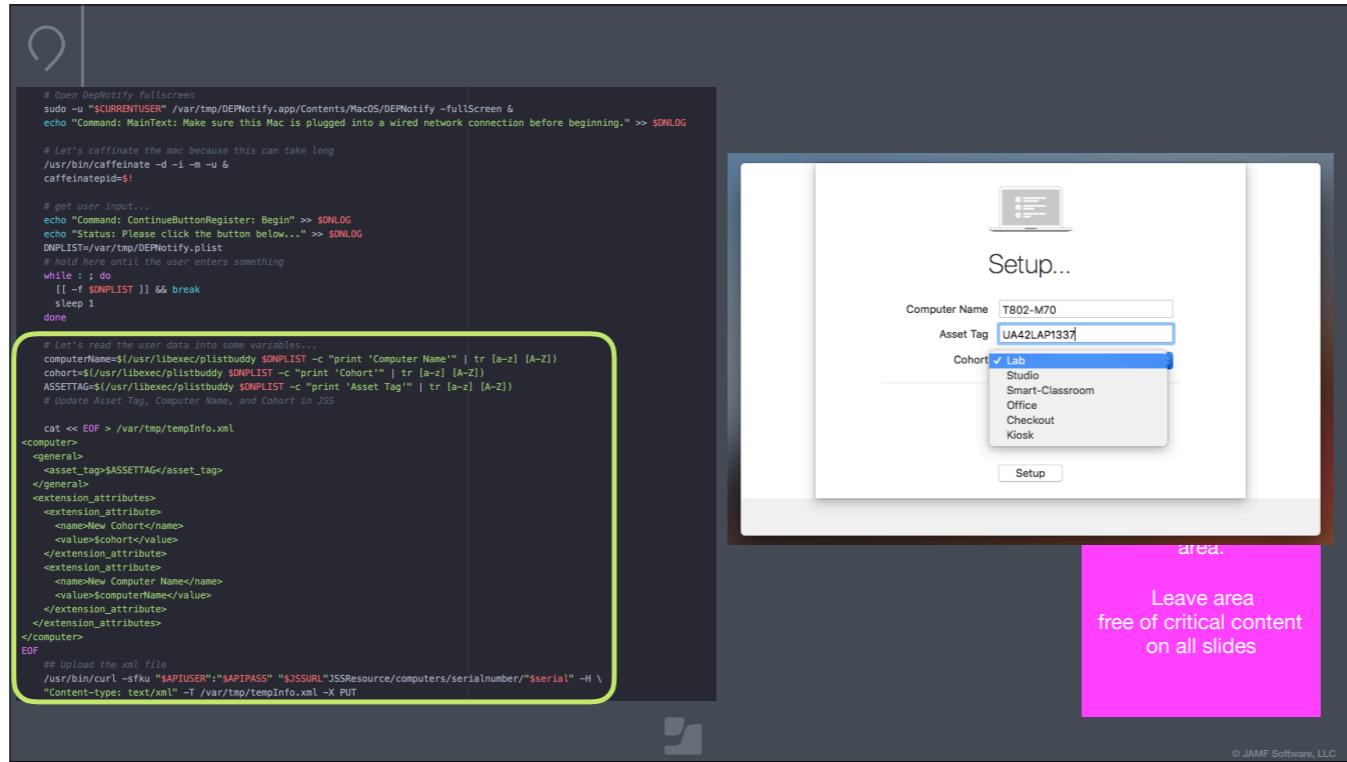
Side by side again. We get that info _C_

Then _C_ we check if either of those EAs are blank or if it's an OFFICE machine

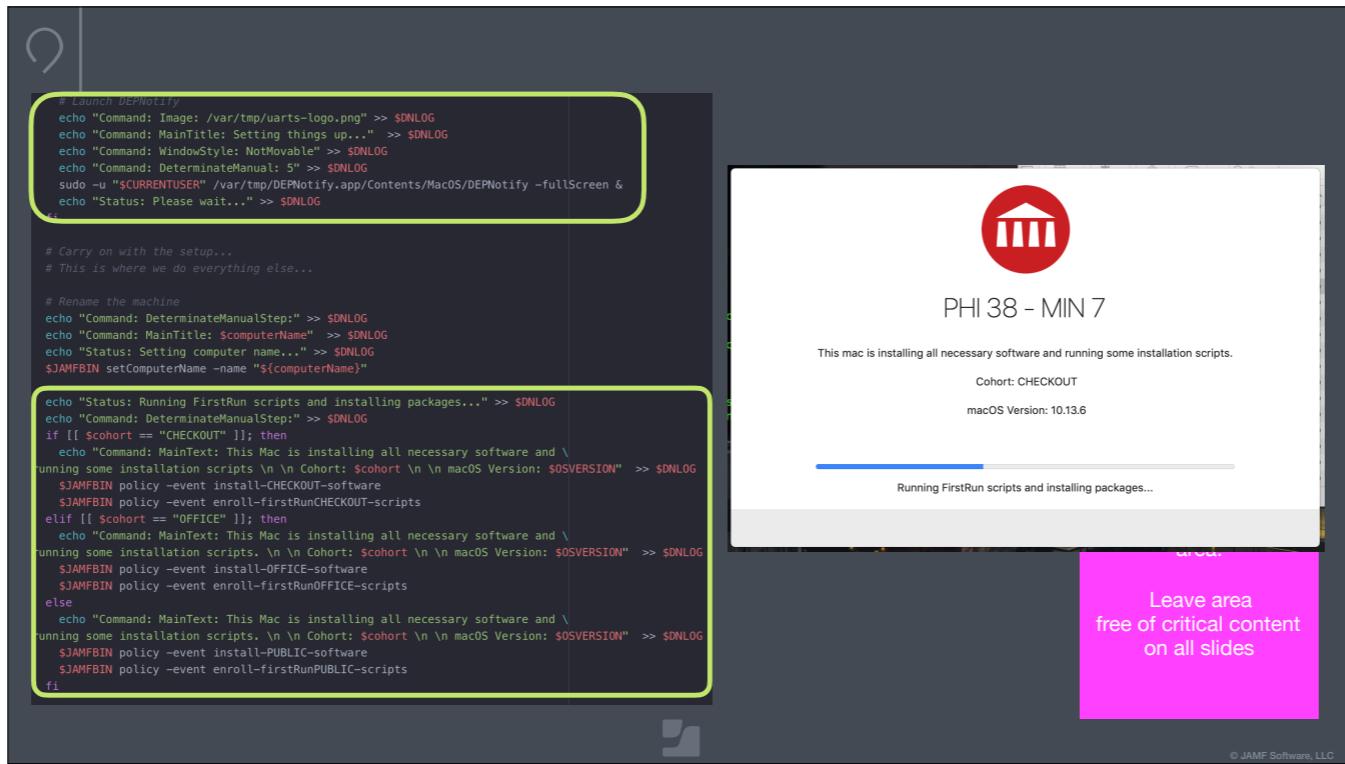
In this case, the machine is an OFFICE machine so it's going to sit there and wait for someone to click "Begin" and enter information just like before.



So we setup our prefs for the registration window to enter asset tag and computer name and the computer type with a dropdown.

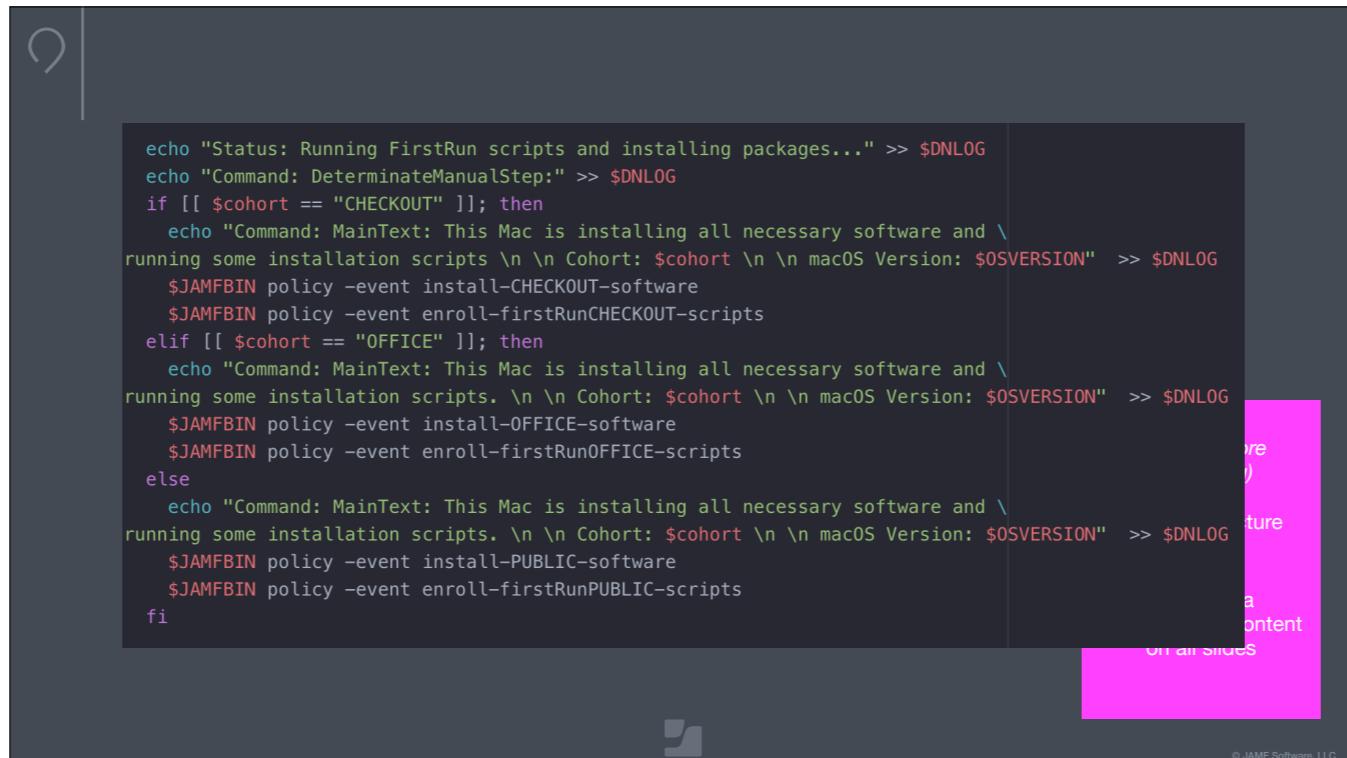


When the tech finishes the registration window we send that new info to the jamf server via API again and continue with setup like every other machine.



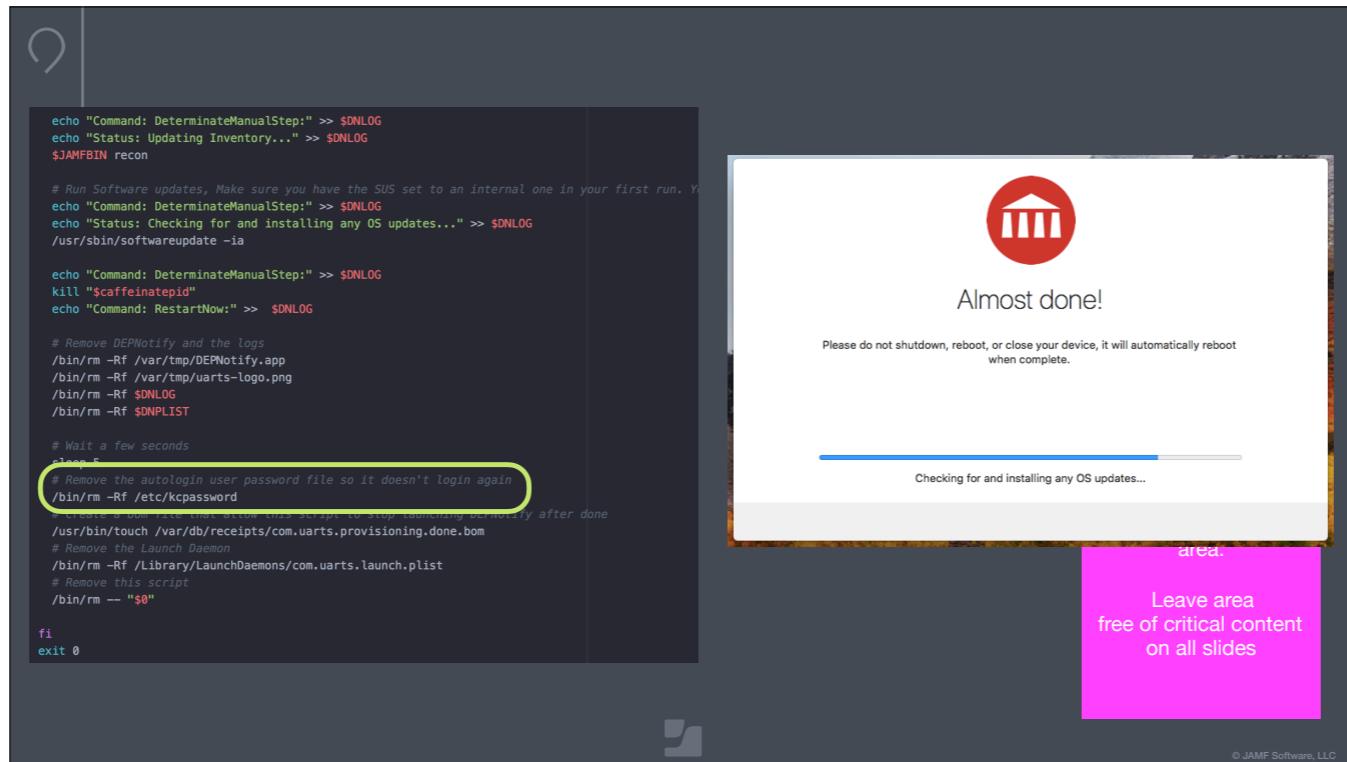
Now, if the machine IS known to the server and falls into one of our automated groups _C_ we setup our just DEPNotify window and launch it. Then just _C_ run commands based on the type of machine.

This section is also where office and unknown systems do everything.



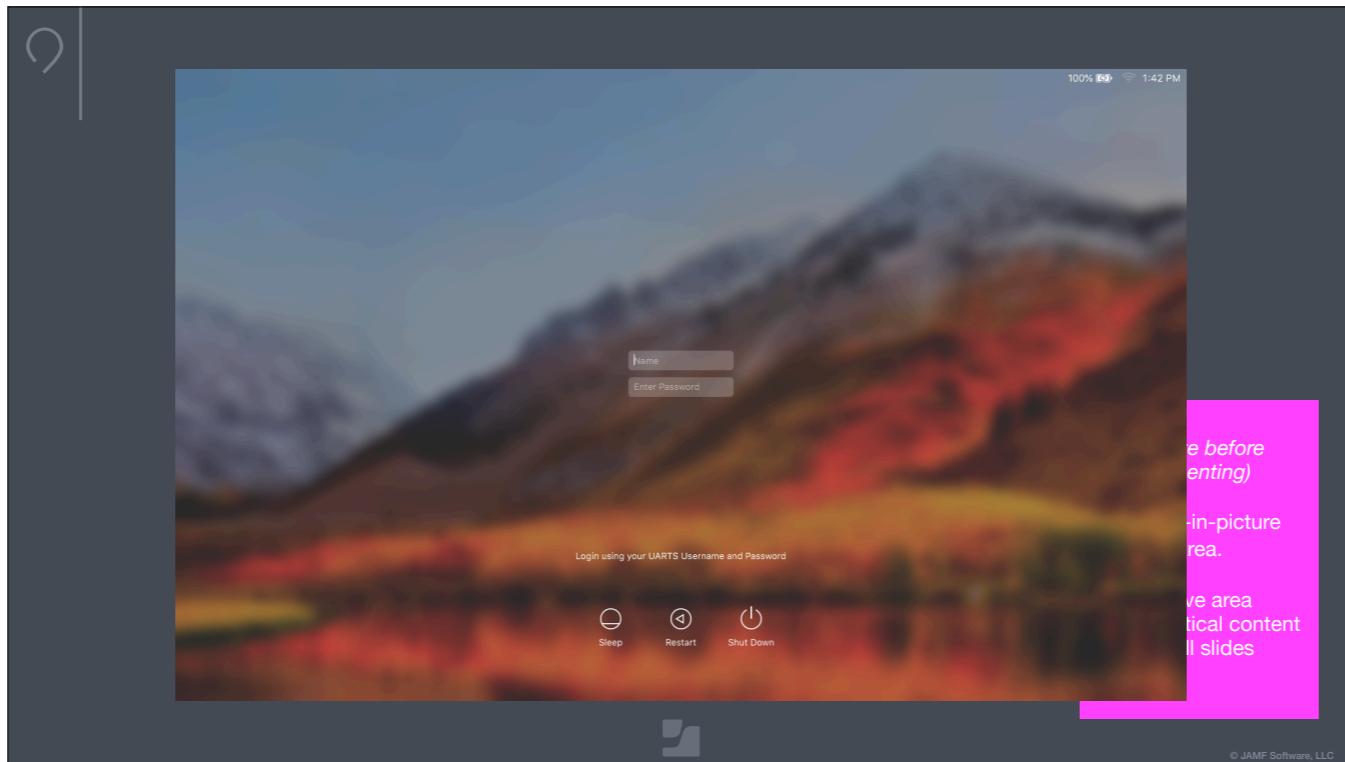
```
echo "Status: Running FirstRun scripts and installing packages..." >> $DNLOG
echo "Command: DeterminateManualStep:" >> $DNLOG
if [[ $cohort == "CHECKOUT" ]]; then
    echo "Command: MainText: This Mac is installing all necessary software and \
running some installation scripts \n \n Cohort: $cohort \n \n macOS Version: $OSVERSION" >> $DNLOG
    $JAMFBIN policy -event install-CHECKOUT-software
    $JAMFBIN policy -event enroll-firstRunCHECKOUT-scripts
elif [[ $cohort == "OFFICE" ]]; then
    echo "Command: MainText: This Mac is installing all necessary software and \
running some installation scripts. \n \n Cohort: $cohort \n \n macOS Version: $OSVERSION" >> $DNLOG
    $JAMFBIN policy -event install-OFFICE-software
    $JAMFBIN policy -event enroll-firstRunOFFICE-scripts
else
    echo "Command: MainText: This Mac is installing all necessary software and \
running some installation scripts. \n \n Cohort: $cohort \n \n macOS Version: $OSVERSION" >> $DNLOG
    $JAMFBIN policy -event install-PUBLIC-software
    $JAMFBIN policy -event enroll-firstRunPUBLIC-scripts
fi
```

Here's that section blown up a bit. You can see we're just running 2 policies for each type. One takes care of scripts and the other is software..which is just another script that calls out software triggers so we can easily add and remove software to our deployments in the jamf webapp

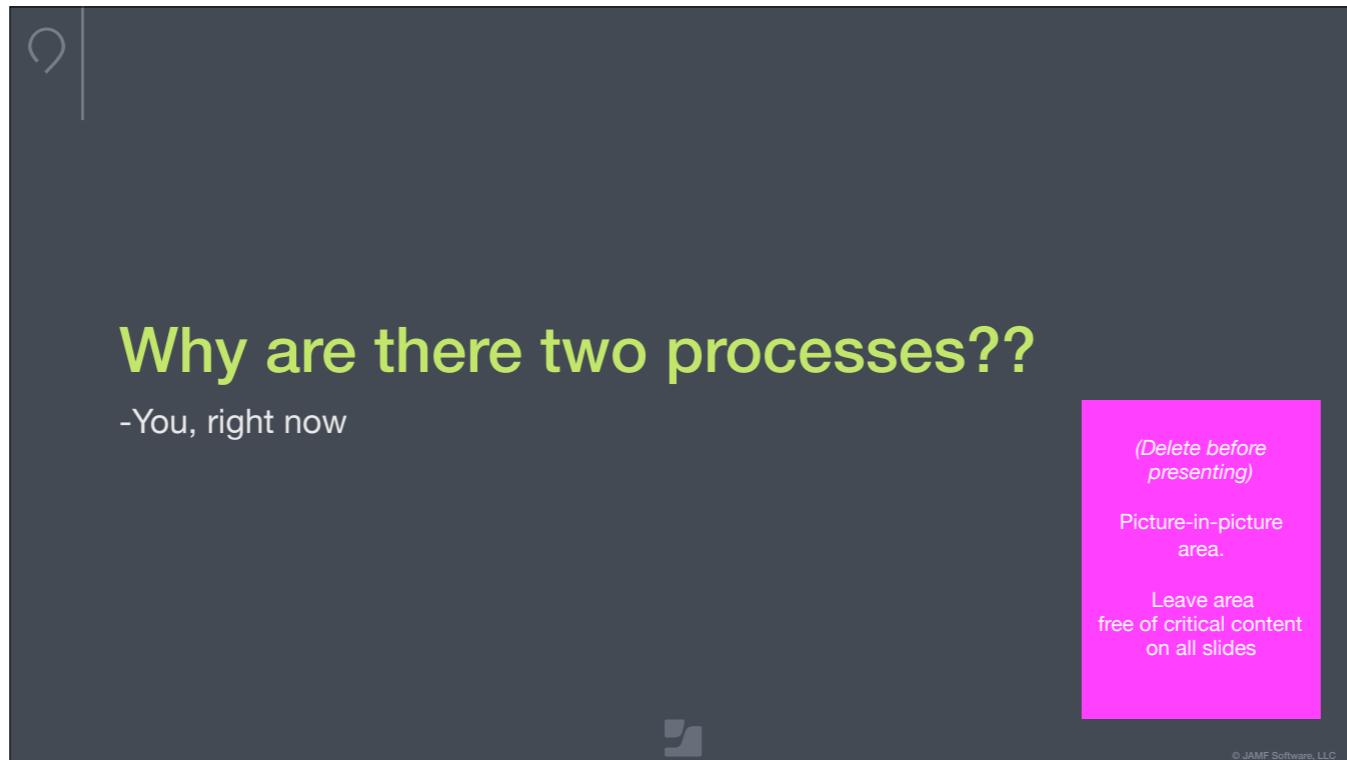


The end of our automation script is the same as the other except this one key line _C_ here. We want to remove the auto login user password file so it doesn't automatically login again after rebooting.

We reboot...and..



We're greeted by our login window. And we'done.



Some of you may be thinking, why do you have two different processes? This is a really good question and the really good answer is that we don't anymore!

The first process was my original proof of concept that we were testing for months before the automated one was done. I wanted to make sure the process worked and our techs were seeing it already before I rolled it out 100%.

We also didn't have a need for the automated setup until the summer when we got PUBLIC new systems in, so I took care of our faculty and staff laptops first since we had a few trickling in throughout the spring.

I also wanted to go through the evolution of our process.

There is actually an updated set of scripts on my Git page to take care of all automation.



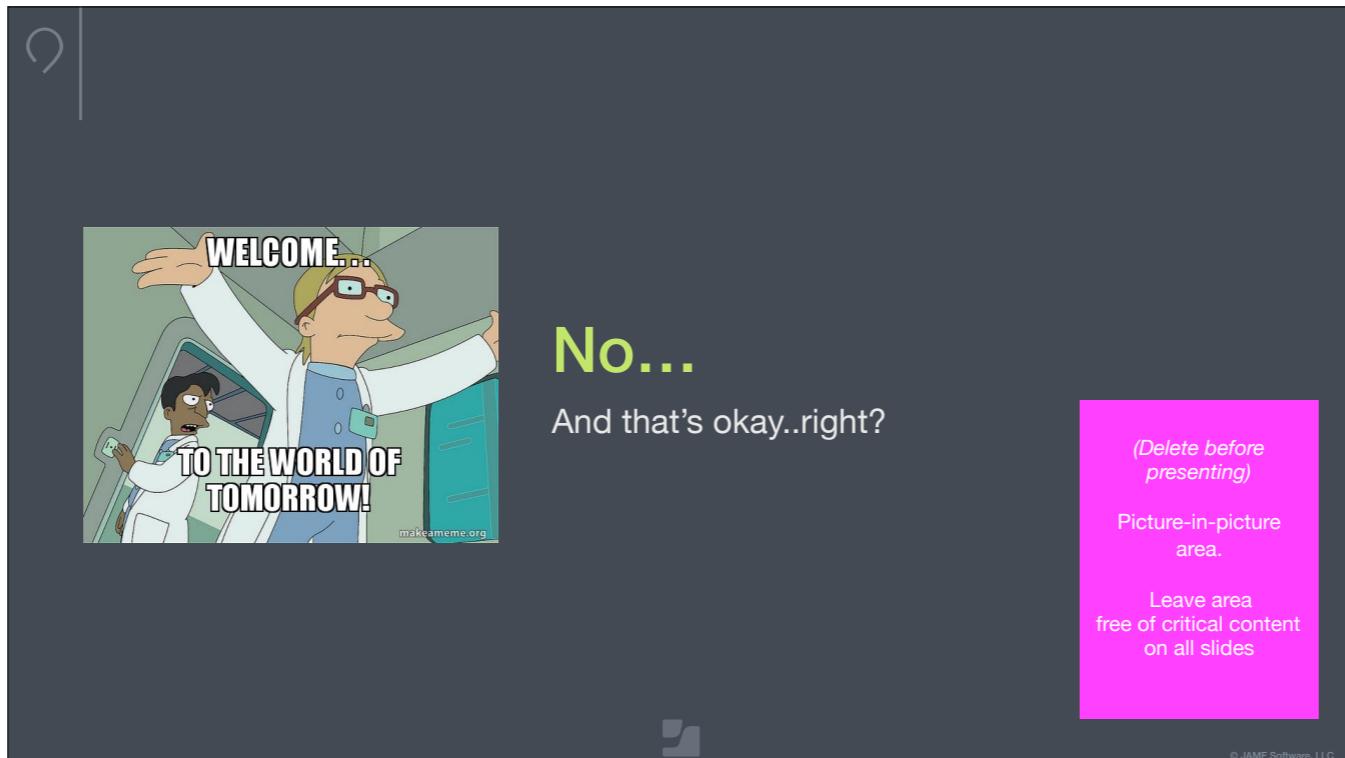
(Delete before
presenting)

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC

So this was totally me after I got my process working. But I have to remind myself... _C_



It's not. Well, not really.
And that's okay for now..

DEPNotify really helped us make the transition a lot smoother even though we started our transition way later than we should have. Hopefully the future will bring us closer to the old “hand off” methods we were used to. _C_ Welcome to the future!!

9

I'm not into scripting...any ideas?

-You, right now...maybe?

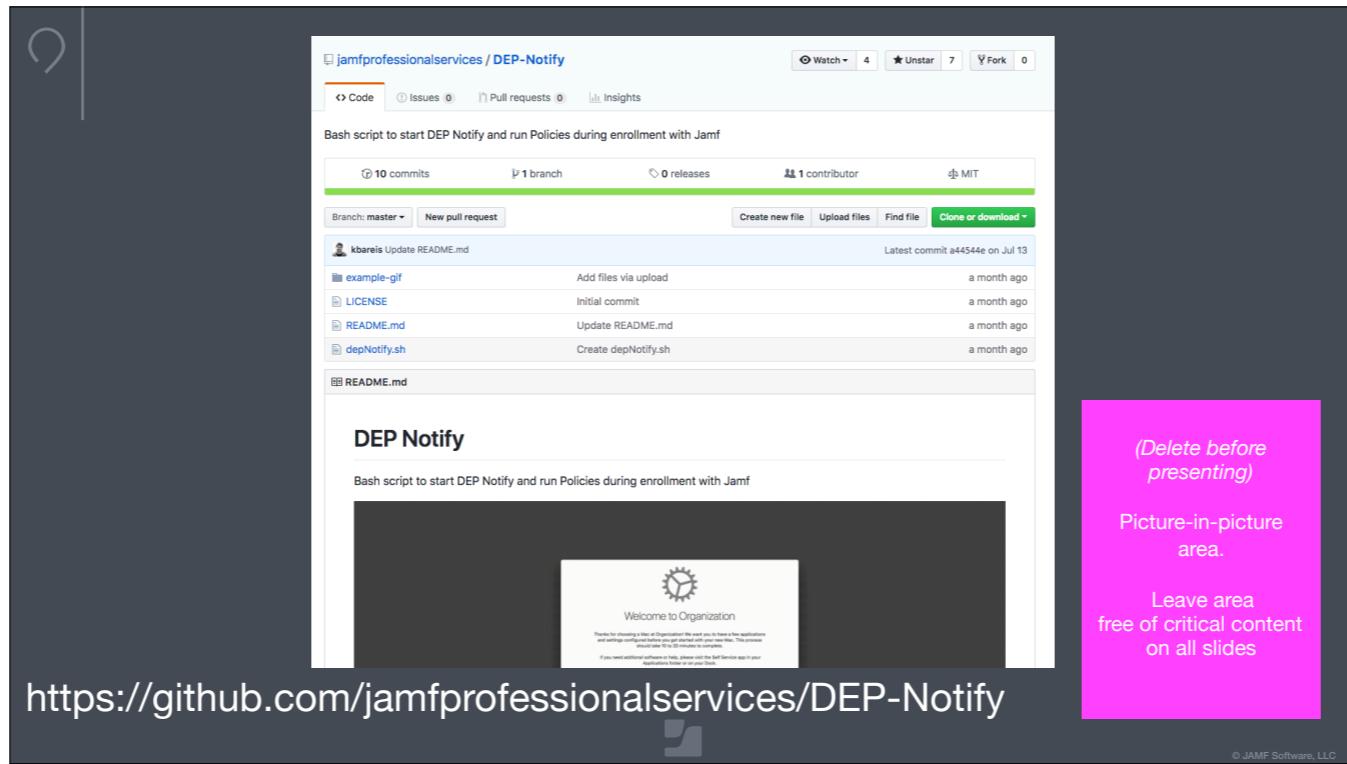
(Delete before presenting)

Picture-in-picture area.

Leave area free of critical content on all slides

© JAMF Software, LLC

There may be some here that don't really want to do a heck of a lot of scripting, or if you want something really easy to deploy. Well I do have a quick solution for you....



Jamf professional service has a git repo for DEPNotify which includes _C_

```
#####
# Variables to Modify
#####
# Testing flag will enable the following things to change:
# - Auto removal of BOM files to reduce errors
# - Sleep commands instead of policies being called
# - Quit Key set to command + control + x
TESTING_MODE=true # Set variable to true or false

# Flag the app to open fullscreen or as a window
FULLSCREEN=false # Set variable to true or false

# Banner image can be 600px wide by 100px high. Images will be scaled to fit
# If this variable is left blank, the generic image will appear
BANNER_IMAGE_PATH="/System/Library/CoreServices/Setup Assistant.app/Contents/Resources/remote_management.tiff"

# Main heading that will be displayed under the image
# If this variable is left blank, the generic banner will appear
BANNER_TITLE="Welcome to Organization"

# Paragraph text that will display under the main heading. For a new line, use \\n
# If this variable is left blank, the generic message will appear. Leave single
# quotes below as double quotes will break the new line.
MAIN_TEXT="Thanks for choosing a Mac at Organization! We want you to have a few applications and settings configured before you begin. You can always change them later if you need to. If you have any questions, please refer to the support page below. We hope you enjoy your new Mac!"

# URL for support or help that will open when the ? is clicked
# If this variable is left blank, the ? will not appear
SUPPORT_URL="https://support.apple.com"

# Initial Start Status text that shows as things are firing up
INITIAL_START_STATUS="Initial Configuration Starting..."

# EULA configuration
# CURRENTLY BROKEN - seeing issues with the EULA and continue buttons
EULA_ENABLED=false # Set variable to true or false
EULA_TXT_FILE="/var/tm/eula.txt"

# The policy array must be formatted "Progress Bar text,customTrigger". These will be
# run in order as they appear below.
POLICY_ARRAY=(
    "Installing Chrome,depNotifyChrome"
    "Installing Firefox,depNotifyFirefox"
)

# Text that will display in the progress bar
INSTALL_COMPLETE_TEXT="Configuration Complete!"
```

A script where you just fill your information into the variables and it will take care of the logic and installation policies for you. It's a very nice easy to use and read script if you need a simple deployment process.

The hopeful future!

- Hope that Apple gives us a way to have 100% zero-touch
 - --eraseinstall flag
 - Skip Setup Assistant?
 - Better use of snapshots?
- DEPNotify at login window
- See what Jamf comes up with

(Delete before presenting)
Picture-in-picture area.
Leave area free of critical content on all slides

© JAMF Software, LLC

So..looking ahead.

We've been getting new tools with every version of macOS.

C Let's Hope that apple gives us true zero-touch.

We're getting close with the erase install flag but we still have to go through setup assistant.
I would also like more control with snapshots.

C DEPNotify at login window would be awesome and I know there is work on this already especially with Nomad Login.

C and of course I would like to see what jamf comes up with for this. There is some work being done already on a DEP workflow that looks promising, but it may be some time off.

Resources

- My GitHub
 - <https://github.com/jmahlman/uarts-scripts/tree/master/DEP%20Scripts>
 - Updated process: <https://github.com/jmahlman/DEPNotify-automated>
- DEPNotify
 - <https://gitlab.com/Mactroll/DEPNotify>
- Neil Martin's Presentation/Code from JNRS
 - <https://github.com/neilmartin83/Jamf-Nation-Roadshow-London-2018>
- Jamf Professional Services DEPNotify repo
 - <https://github.com/jamfprofessionalservices/DEP-Notify>

(Delete before
presenting)

Picture-in-picture
area.

Leave area
free of critical content
on all slides

© JAMF Software, LLC



THANK YOU!

© JAMF Software, L

6 | Thank you for listening!

Give us feedback by
completing the 2-question
session survey in the JNUC
2018 app.

UP NEXT

Session title

Session time



© JAMF Software, LLC