

Warranty Reporting

Warranty information can be critical to maintaining a well-functioning fleet of Macs in an organization, which is typically obtained by using Apple's GSX service. However, because of the requirements needed to become a member of GSX, it's not always possible for an organization to jump on the GSX train. This workflow can be used in lieu of GSX for the purpose of obtaining the estimated AppleCare Warranty Expiration Date

You Will Need

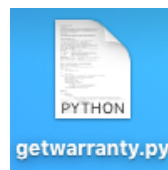
- Jamf Pro
 - Extension Attribute
 - Policy
 - Package
 - Script
- Composer
- Jamf Admin

Overview

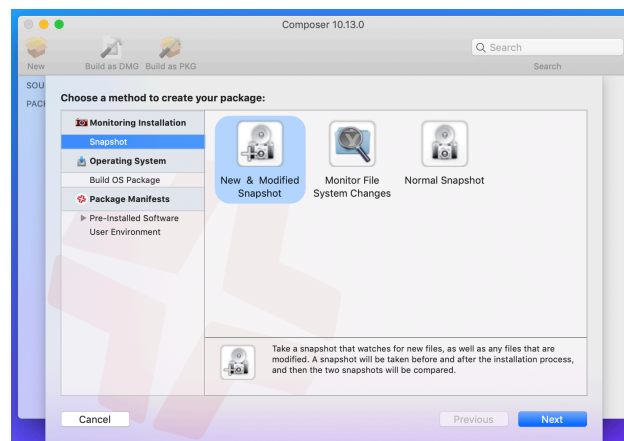
When everything is set up, we will run a Policy once on each computer that installs a Package (which contains `getwarranty.py` by Michael Lynn) and then executes a Script. The Script will utilize the contents of the Package to obtain the estimated AppleCare Warranty Expiration Date, and then use Jamf Pro's Classic API to update an Extension Attribute in Jamf Pro with the result.

Creating the Package with Composer

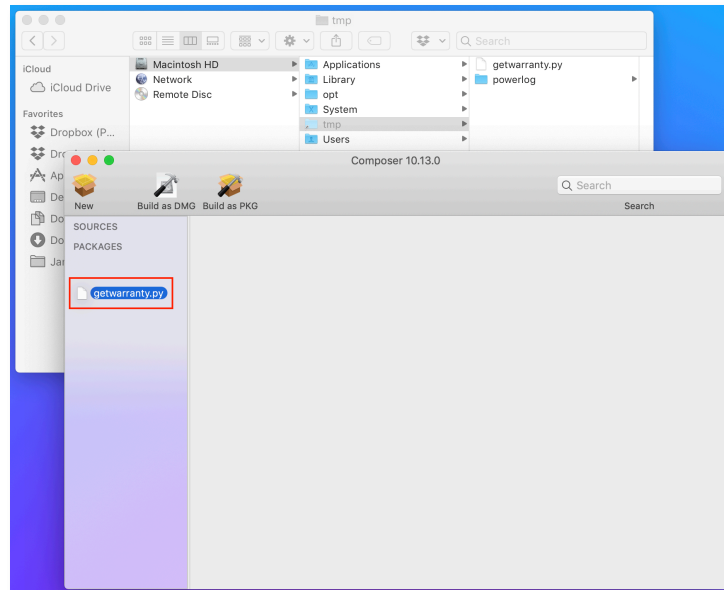
1. Obtain a copy of `getwarranty.py` from GitHub: <https://github.com/pudquick/pyMacWarranty/blob/master/getwarranty.py>
 - Copy and paste the script into a new text file on your computer, and make sure to save it as a file called `getwarranty.py`.



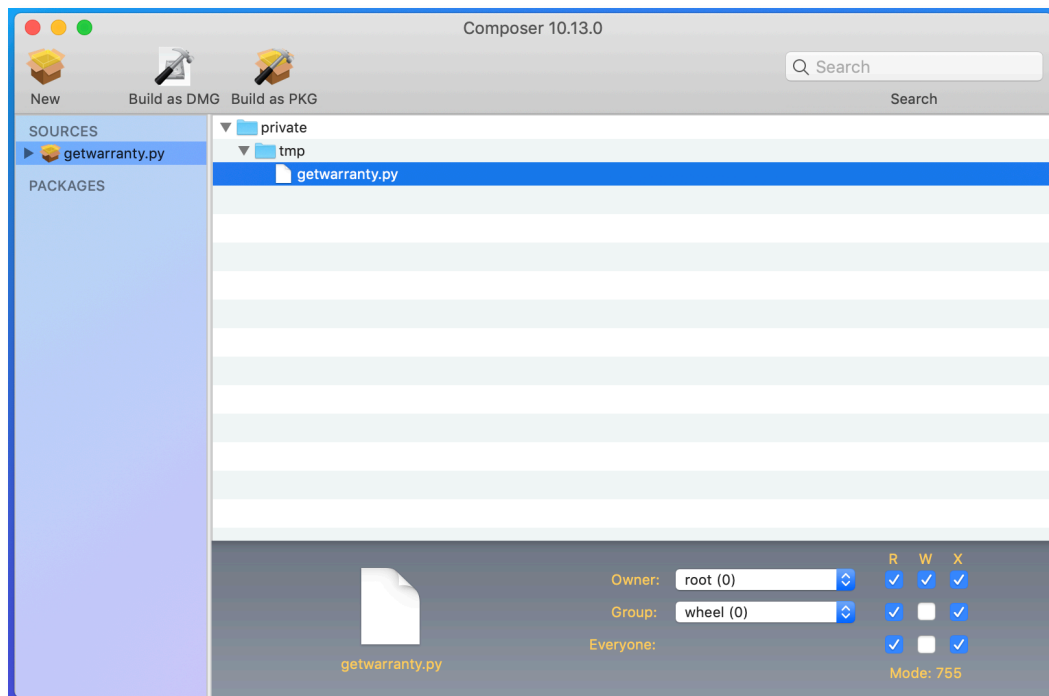
2. Place the newly-saved `getwarranty.py` file in `/tmp`
3. Open Composer.
 - If you are offered a welcome screen like the following, click Cancel in the lower left:



4. Drag the `getwarranty.py` script from `/tmp` into the left-hand side of Composer:



5. Expand the “private” and “tmp” folders within Composer to reveal `getwarranty.py`. Select it so that it is highlighted, and then use the dropdown menus and checkboxes to configure the Owner as “root”, the Group as “wheel”, and the Mode to **755**:



6. Click Build as PKG and save the Package (with any name) to your Mac
7. Upload the Package to your distribution point using Jamf Admin or the Jamf Pro web application

Setting up Encrypted Credentials

We will be utilizing the Classic API as a part of this workflow, which will require authentication from a Jamf Pro User Account. It is **never** a good idea to send an account password over the internet in clear text, so we're going to encrypt the password first.

1. Create a new Jamf Pro User Account that **only** has **Update** permissions on **Computers** and **Users** Jamf Pro Objects.
 - These are the minimum permissions required for the script's API calls to work
 - Set the password of this account to anything you'd like
2. Open the following GitHub page: https://github.com/jamf/Encrypted-Script-Parameters/blob/master/EncryptedStrings_Bash.sh
 - Similar to what we did for getwarranty.py, copy **lines 8 through 16** and paste them into a new text file on your computer.
3. Add a new line to the bottom of the script that says:
 - `GenerateEncryptedString "newAccountPassword"`
 - Swap "newAccountPassword" with the password for the account created in Step 1 of this section, but maintain the double quotes around it.
 - Here's an example of how this could look, with a password of myAw3som3P@ssword!

```
function GenerateEncryptedString() {  
    # Usage ~$ GenerateEncryptedString "String"  
    local STRING="${1}"  
    local SALT=$(openssl rand -hex 8)  
    local K=$(openssl rand -hex 12)  
    local ENCRYPTED=$(echo "${STRING}" | openssl enc -aes256 -a -A -S "${SALT}" -k "${K}")  
    echo "Encrypted String: ${ENCRYPTED}"  
    echo "Salt: ${SALT} | Passphrase: ${K}"  
}
```

```
GenerateEncryptedString "myAw3som3P@ssword!"
```

4. Save this file to your Desktop, titled "encrypted.sh"
5. In Terminal, execute the following command:
 - `sh ~/Desktop/encrypted.sh`
 - The output should look similar (but not exact, since you're using a different password) to this:

```
Encrypted String: U2FsdGVkX1/CDD2w/SFd+EhoMftu0oE+TpSvFqJGefZNqzmUYKFrIN06uoqvbz6p  
Salt: c20c3db0fd215df8 | Passphrase: 8d59dada81a8da163466b83a
```

6. Obtain iMatthewCM's `getWarrantyInformation.sh` script from GitHub: <https://github.com/iMatthewCM/Jamf-Scripts/blob/master/macOS/getWarrantyInformation.sh>
7. Open the obtained `getWarrantyInformation.sh` in a text editor. Scroll down to the section entitled "ENCRYPTED CREDENTIALS CONFIGURATION"

8. Edit lines 51 and 54 with the Salt and Passphrase values that Terminal just displayed:

```

42 #####
43 #
44 # ENCRYPTED CREDENTIALS CONFIGURATION
45 #
46 #####
47
48 #Follow the steps in the "Setting up Encrypted Credentials" section of the documentation to
   #configure the following:
49
50 #Enter the SALT value:
51 SALT="c20c3db0fd215df8"
52
53 #Enter the PASSPHRASE value:
54 PASSPHRASE="8d59dada81a8da163466b83a"
55

```

9. Make sure to keep the Encrypted String that was returned from Step 5 handy - we'll need that when we configure our Policy.

Modifying Script Variables

Now we'll finish up with the rest of the component configuration by creating and configuring our Extension Attribute, and pointing the getWarrantyInformation.sh script at our Jamf Pro server.

1. Go to Jamf Pro > Settings > Computer Management > Extension Attributes > New
2. Configure the Extension Attribute to reflect the following:

DISPLAY NAME Display name for the extension attribute

AppleCare Warranty Expiration Date

☒ Enabled (script input type only)

DESCRIPTION Description for the extension attribute

This EA utilizes iMatthewCM's getWarrantyInformation.sh workflow in conjunction with pudquick's getwarranty.py script to report the estimated AppleCare Warranty Expiration Date.

DATA TYPE Type of data being collected

Date (YYYY-MM-DD hh:mm:ss) ▼

INVENTORY DISPLAY Category in which to display the extension attribute in Jamf Pro

Purchasing ▼

INPUT TYPE Input type to use to populate the extension attribute

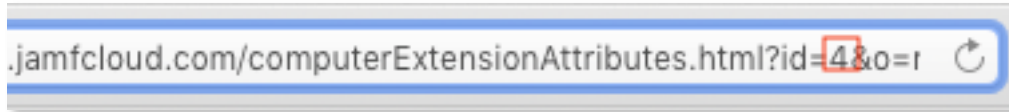
Text Field ▼

RECON DISPLAY Pane on which to display the extension attribute in Recon

Purchasing ▼

- The most important settings are the Data Type and the Input Type menus. These **must** be set to "Date" and "Text Field", respectively, for this workflow to function properly.

3. Save the Extension Attribute
4. While still looking at the page showing the configured Extension Attribute options, click into the URL box in your browser. Towards the end of the URL there will be a section that says "?id=" with a number following. Take note of that number, we will need it momentarily.



5. Head back to getWarrantyInformation.sh (where we just configured the Salt and Passphrase for our encrypted credentials) and scroll down to the section entitled "DEFINE VARIABLES"
6. Modify line 67 (JAMF_PRO_URL) to reflect your Jamf Pro server's URL
7. Modifying line 71 (API_USERNAME) to reflect the username of the Jamf Pro User Account we created during Step 1 of the "Setting up Encrypted Credentials" section of this documentation.
8. Modify line 76 (WARRANTY_EA_ID) to reflect the Extension Attribute Object ID we just collected from the URL
9. Modify line 80 (WARRANTY_EA_NAME) to reflect the **exact** name of the Extension Attribute configured in Step 2.

```

56 #####
57 #
58 # DEFINE VARIABLES
59 #
60 #####
61
62 #Follow the steps in the "Modifying Script Variables" section of the documentation to configure the
63 following:
64
65 #Enter your Jamf Pro URL - include any necessary ports but do NOT include a trailing /
66 #On-Prem Example: https://myjss.com:8443"
67 #Jamf Cloud Example: https://myjss.jamfcloud.com"
68 JAMF_PRO_URL="https://myjss.jamfcloud.com"
69
70 #Username for the API call
71 #This user only needs permissions to UPDATE Computer and User objects, and should be entered in
72 clear text
73 API_USERNAME="warrantyAPIuser"
74
75 #The Jamf Pro Object ID of the EA we're going to update
76 #You can get this from clicking into the EA in the GUI and checking the URL
77 #There will be a section that says id=X - write in the value for X here:
78 WARRANTY_EA_ID=4
79
80 #This is the display name for the EA - write this exactly as it appears in Jamf Pro
81 #Capitalization matters, and make sure to keep the quotes around it as seen in the example
82 WARRANTY_EA_NAME="AppleCare Warranty Expiration Date"

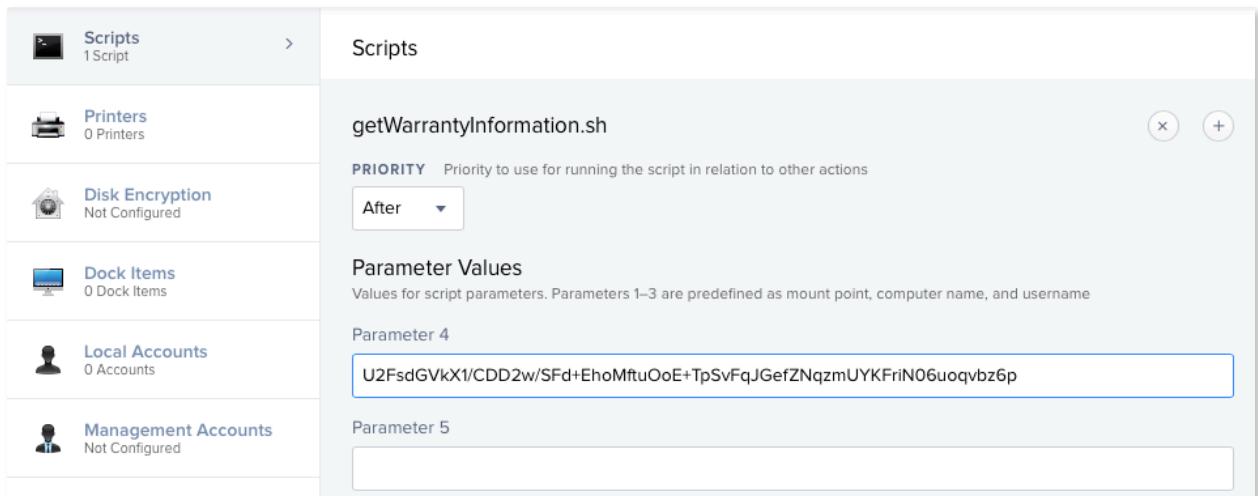
```

10. Save the script - configuration is now complete!
11. Upload this script to Jamf Pro
 - Jamf Admin works great for this
 - Alternatively, go to Jamf Pro > Settings > Computer Management > Scripts > New, then copy and paste the contents of your configured script into Jamf Pro

Deploying the Workflow

Now that we have all of the individual pieces set up and uploaded to Jamf Pro, we can create our Policy to deploy everything to our managed Macs.

1. Go to Jamf Pro > Computers > Policies > New
2. Add a **Display Name** of your choice
3. For the **Trigger**, select **Recurring Check-in**
4. Set the **Execution Frequency** to **Once Per Computer**
5. Configure the **Packages payload** and add the Package that was uploaded in Step 7 of the “Creating the Package with Composer” section of this documentation
 - Ensure “**Action**” is set to “**Install**” in the dropdown menu
6. Configure the **Scripts payload** and add the Script that was uploaded in Step 11 of the “Modifying Script Variables” section of this documentation
 - Ensure “**Priority**” is set to “**After**” in the dropdown menu
 - This is because we need the Package to install first, since the script we’re deploying is dependent on the contents of the Package being available on the computer.
7. In the text box labeled “**Parameter 4**”, paste in the **value for the Encrypted String** that we obtained in Step 5 of the “Setting up Encrypted Credentials” section of this documentation.
8. The configured Scripts payload should now look similar to the following:



9. Click on the **Scope** tab, and set “**Target Computers**” to “**All Computers**”
 - If you don’t want warranty information for every single managed Mac in your fleet, feel free to modify the scope as you see fit.
10. Save the Policy

And that’s it! The workflow is now configured. As computers start checking in and trigger the Policy, the Package will be installed putting getwarranty.py into /tmp. The getWarrantyInformation.sh script will then run and update the Jamf Pro Extension Attribute with the estimated AppleCare Warranty Expiration Date, and then the getwarranty.sh script laid down by the Package will be deleted.

Please note that since the Extension Attribute was formatted with a Data Type of “Date”, Smart Computer Groups may be leveraged using the “Before YYYY-MM-DD” or “After YYYY-MM-DD” operators to sort your fleet by their warranty expiration dates!