Top Tips

Cool ideas, taking your admin game to the next level.

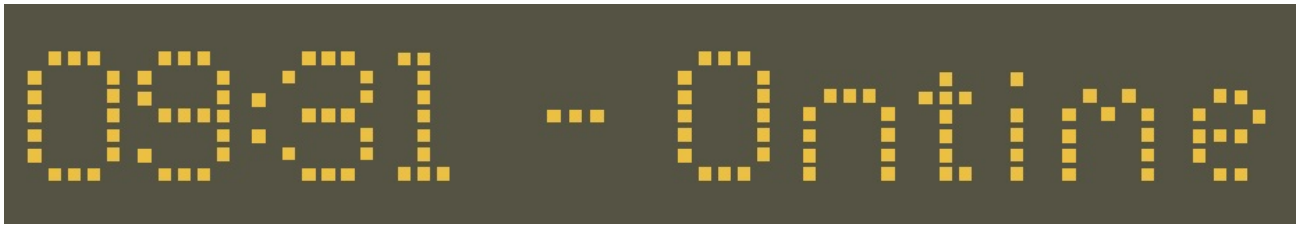
- Desktop A-Game
  - Desktop Fileset Timed Events
  - Apple Profiles & Dependencies
  - Policy Loops
  - Updating 3rd Party Software
  - Un-install Filesets
  - Inventory Items in Scripts
  - Script Logging
- Apple MDM
  - Profile Payload Planning
  - Inventory Items in Profiles
- Android EMM
  - Android Policy Planning

Desktop A-Game

Desktop Fileset Timed Events

Description

Filesets have the option to set an activation time, but what about items based upon timing of day, for example, rather than a dedicated date.



Policy Blocker Scripts are really designed to pause management of clients. However, with some clever use, a Policy Blocker Script can provide us with some assistance. This script type runs every 5 minutes on clients. Although the intention is to pause management until the script reports an exit status of 0, the 5 minute continual trigger can be leveraged.

Extending this with Custom Fields, it is possible to build out a desired outcome.

Ingredients

- One (or more) Custom Fields; 3 provided - Active Time Custom Fields.customfields
- Policy Blocker Scripts

macOS	Windows
Policy - Timed Event macOS.fileset.zip	Policy - Timed Event Windows.fileset.zip

Directions

- Download and import the provided Custom Fields
- Download and import the appropriate Fileset(s) (macOS and/or Windows)

Alter the Custom Fields values for Inactive and Active times to suit. Time is set using hours, minutes and seconds. E.g.

Desired Time (HH:MM)	Custom Field Value (HHMMSS not including leading zeros)
09:30	93000
12:55	125500
18:00	180000

> ⚠ If changing the Custom Field value for a device which is already running the policy, 2 subsequent Model Updates will need to be received by the client, if looking for a more 'immediate' result. The blocker script holds the client until success. This means, during a Model Update, the blocker will run before the new Custom Field Value will be read by the device. As such, only after a subsequent update (or inventory) will the blocker script be aware of the new Custom Field value. Custom Fields will naturally update on devices with inventory, but this is less frequent.

Time Order

Times can be either way around.

Examples:

Consider working hours to begin at 08:30 and end at 18:00

Example 1

An item should only be considered outside of standard working hours.

- Disable Active Time: 83000
- Enable Active Time: 180000

Display Name	Internal Name	Field Type	Association Count	Field Value
Enable Active Time	enable_active_time	Integer	All Clients	180000
Disable Active Time	disable_active_time	Integer	All Clients	83000

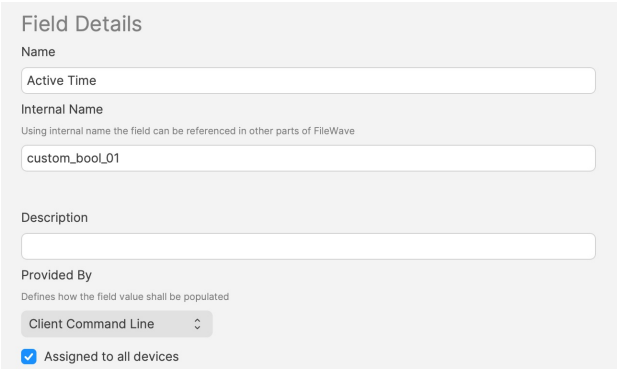
Example 2

Alternatively, an item should be considered during working hours:

- Enable Active Time: 83000
- Disable Active Time: 180000

Display Name	Internal Name	Field Type	Association Count	Field Value
Disable Active Time	disable_active_time	Integer	All Clients	180000
Enable Active Time	enable_active_time	Integer	All Clients	83000

Imported Custom Fields are disabled for all devices by default. Once tested, consider using the option to assign to all devices for each Custom Field imported. The file provided contains all 3 Custom Fields.



Smart Groups

A third Custom Field holds a true/false value. This value may be used with a Smart Group query, to determine if an item should be associated at this time or not.

> This Custom Field is set to use custom_bool_01. If this is already in use, an alternate Custom Bool number should be utilised instead. This is editable through the Scripts Environment Variables.

The script uses the following method to set these values.

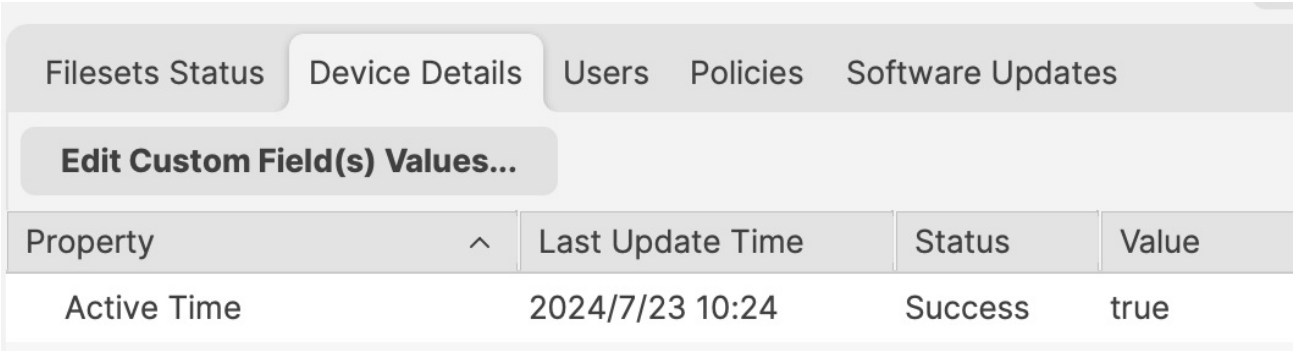
https://kb.filewave.com/books/custom-fields/page/add-filewave-custom-inventory-fields-remotely-using-a-fileset

Example

An update to Firefox needs to occur after 17:00 or before 09:30

- Disable Active Time = 93000
- Enable Active Time = 170000

Between these times, the 3rd Custom Field ‘Active Time’ should be False/0. Outside of these times, the ‘Active Time’ Custom Field should be True/1.



Smart Group can be based upon the following:

- Is Active Time True/1

- Is the version of Firefox matching that within the Fileset

> Do not just use the active time, unless intentional. Devices will continually enter and leave the Smart Group if this is set to only use the Active Time, each day. If an item is associated in this way, associated Filesets will trigger every time the device enters the group.

Deeper Dive

The Policy Blocker script has 2 considerations initially:

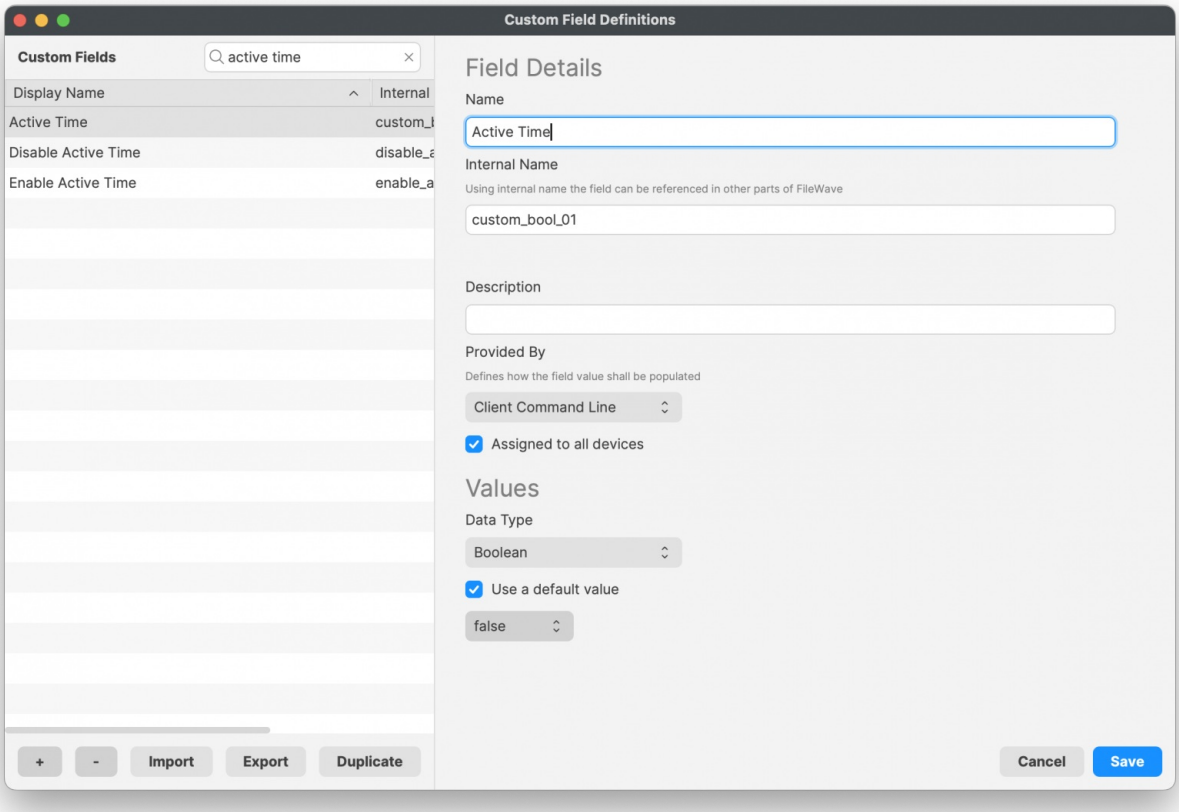
- The hours between which the timed event should occur
- The current time

This means there is a time beyond which the desired action may occur and a time beyond which the action should not occur and this needs to be compared with the current time.

The enable/disable active times are provided by way of Executable Environments. Taking this a step further, these times are defined using Administrator Custom Fields. This way the times can more easily be altered if required.

A third Custom Field is being used to indicate if the current time is one of activity allowance or not, but this time a Client Command Line Custom Field.

Client Command Line Custom Fields are stored locally on the device and then this value is available to the server, both for visibility, but can also be used in queries, for Smart Groups.



> Client Command Liine Custom Fields may be altered in FileWave Central Admin App, however, as soon as the device checks back inventory, the value from the client will be pushed back to the server.

Apple Profiles & Dependencies

Description

Dependencies offer a structure for Fileset installations, ensuring one or more Filesets are installed prior to one or more other Filesets. This works great, apart from where Apple MDM Filesets are involved.

Fileset Activation Quick Re-cap

Standard Fileset

1. Client checks-in.
2. Manifest is observed
3. New items are pushed to device and activate

Apple MDM Fileset (Profiles)

1. APNs request sent to Apple
2. Device pulls queued APNs requests from Apple
3. For each APNs request, device reaches out to relevant servers, for MDM requests, this is the FileWave Server
4. Device checks-in
5. Queued MDM commands are pushed to device, e.g. InstallProfile
6. Profile instals

For standard Filesets, FileWave is in control of the communication. However, for Apple MDM, there is an unknown amount of delay until the Profile is installed.

The Issue

Since Filesets are installed sequentially, if a Fileset were allowed to depend upon an Apple MDM Fileset, the client would be held waiting for an unknown period time, preventing other Filesets and configuration from actioning. For this reason, Apple MDM Filesets can only depend on a different Fileset type and not the other way around.

Requirement Scripts

Requirement Scripts allow a Fileset to fail, let the client continue and then 2 minutes later try the requirement again. The Requirement Script will continue with this process, whilst there is a non-zero exit code. By way of this process, the Requirement Script gives the ability to delay the installation of the Fileset, until any required Profiles are installed beforehand.

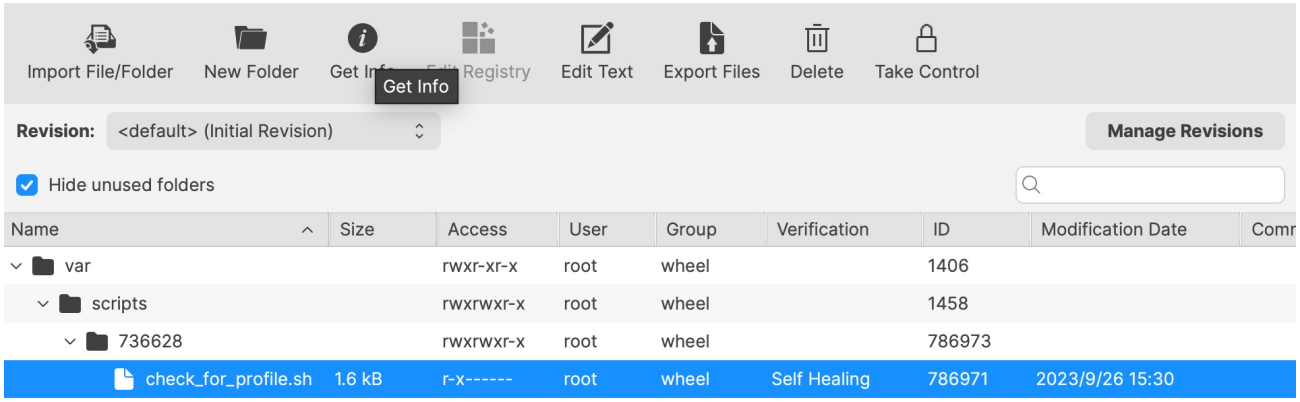
Ingredients

- Fileset designed to use a Requirement Script to ensure Profile is installed prior to activation
- Associated Profile ID(s)

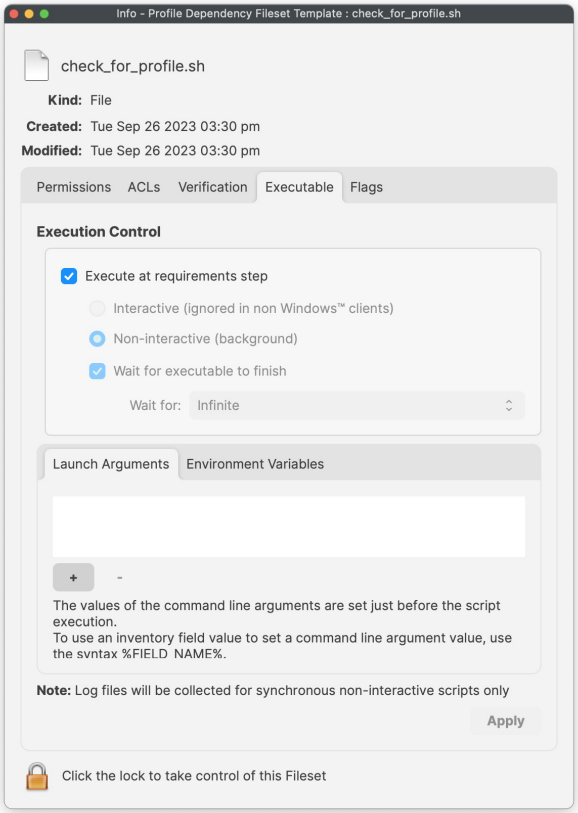
Profile Dependency Fileset Template.fileset.zip

Directions

Download the Fileset, import into FileWave and edit to match requirements. Select the 'check_for_profile.sh' script and click 'Get Info':

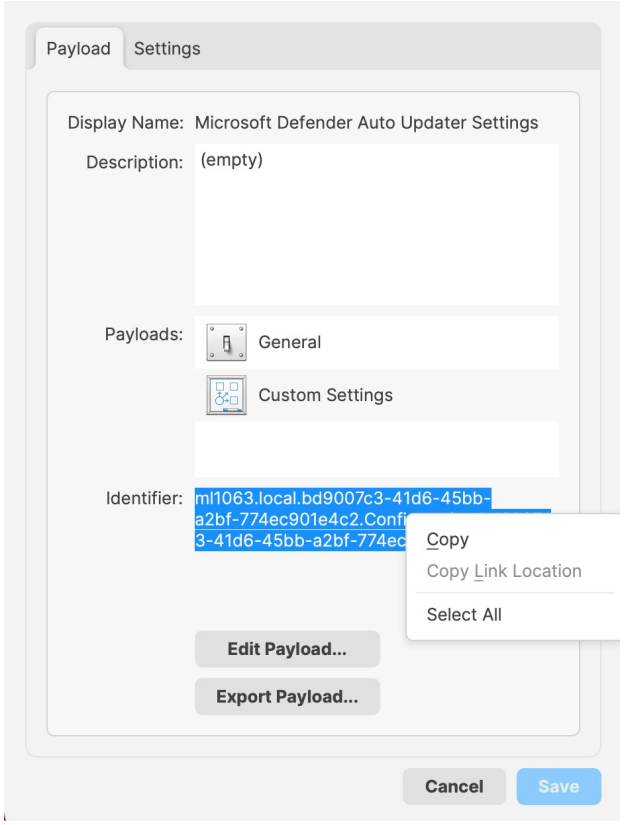


The Launch Arguments will initially appear empty.

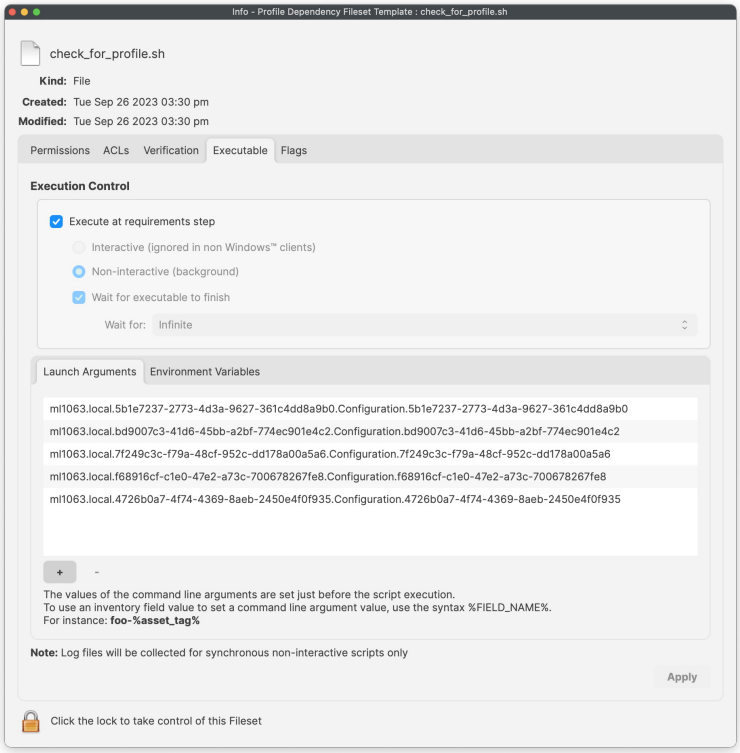


For each Profile that needs to be considered for installation prior to this Fileset, its ID must be added to the list of Launch Arguments (one entry per Profile ID). Profile IDs can be obtained from within the Payload details of the Profile Filesets.

For eah Profile that must be installed, open the Profile for editing, highlight the Identifier and copy.



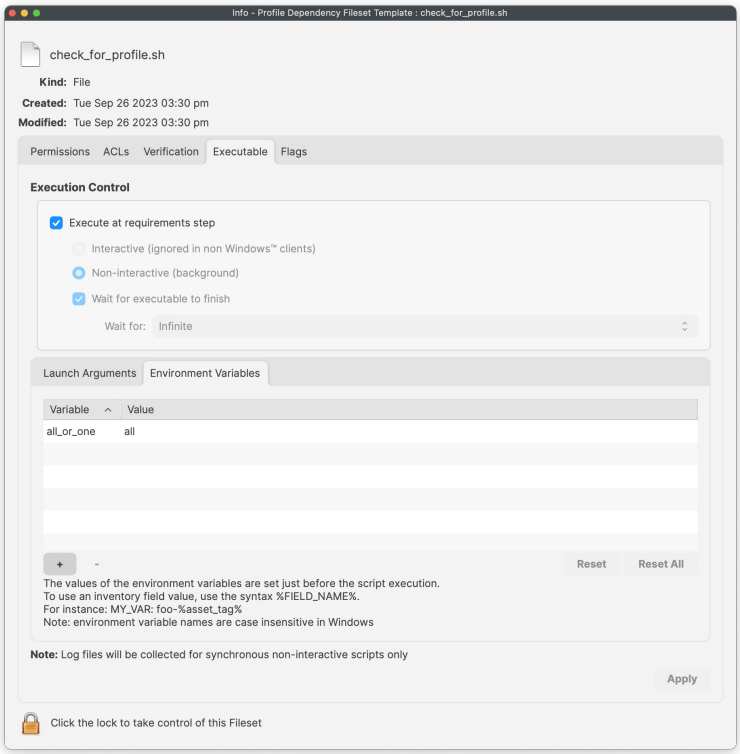
Copy these IDs into the LaunchArguments of the Fileset. Example below shows 5 Profile IDs added for a Microsoft Defender Installer.



The script allows for the idea of either by way of an Environment Variable (all_or_one). Set the value appropriately:

all	All of the listed Profiles must be installed prior to the Fileset becoming active.
one	At least one of the included Profiles must be installed prior to the Fileset becoming active

The below, shows a Fileset set to require all Profiles are installed, before Fileset activation for the same Microsoft Defender example:



With this set, add any additional installers into the Fileset, that would need to be installed, once the provided Profiles are installed.

Create a Fileset Group and add this Fileset and all necessary Profiles to the same group (not necessary, but somewhat neater to manage)

For example:

- Microsoft Defender Associated
  - MicroSoft Defender Installer (macOS)
  - Microsoft Defender Uninstaller
  - Profile - Microsoft Defender - Kernel Extension
  - Profile - Microsoft Defender - System Extension
  - Profile - Microsoft Defender - Auto Updater Settings
  - Profile - Microsoft Defender - Notifications
  - Profile - Microsoft Defender - TCC
  - Profile - Microsoft Defender - Web Content Filter

Associate the Fileset Group, test and then rollout to more devices once happy.

> This Fileset is particular useful with Apple TCC Privacy Settings Profiles. Privacy settings provide access permissions for software to function. However, typically these Profiles need to be installed before the process that they are allowing is started. This means, if the software is allowed to instal before the Profile is installed, the software process would need restarting after the Profile is installed. The above Fileset offers the solution around this, where the Fileset will only attempt download and installation once the Profile is in place.

Policy Loops

What

Smart Groups provide extensive power to Fileset management. By way of preset criteria, devices enter and leave groups appropriately. However, it is possible for this to go wrong and get unexpected experiences. Policy Loops are an example of such behaviour.

Name	Location
Windows TeamViewer	/
Windows Firefox	/
macOS TeamViewer	/

When/Why

Time to explain, with examples.

Example 1

Imagine a PKG macOS installer Fileset for an app called CLU.app, version 1.0

This is associated to all devices based upon two criteria:

- Device OS is macOS
- Device does not have CLU.app version 1.0 installed

Once the software is installed, the devices no longer belong to the group, since version 1.0 I now installed.

A new version of the software is released, version 1.1. A new association is created with a differing group, with Criteria:

- Device OS is macOS
- Device does not have CLU.app version 1.1 installed

Perhaps the apparent issue is already obvious.

- Devices running version 1.0 will be included in the new Smart Group at the next refresh time.
- The Fileset will activate and the software will transfer from version 1.0 to 1.1
- Devices will leave this new group for version 1.1

Clearly, all should be done now, until the next new version is released. However, there is an issue.

When devices check back that version 1.1. is installed, at the next Smart Group refresh, devices will be added back into the group for version 1.0, since this version is no longer installed. If this older version PKG is allowed to instal over the newer version, the software will be downgraded back to version 1.0.

But hang on. If 1.0 is now installed, the device will be added back into the Smart Group for version 1.1 at next refresh, with the consequence of installing 1.1. again.

This is an example of a Policy Loop. The device will continually be adding itself in and out of groups, installing and removing software as it goes.

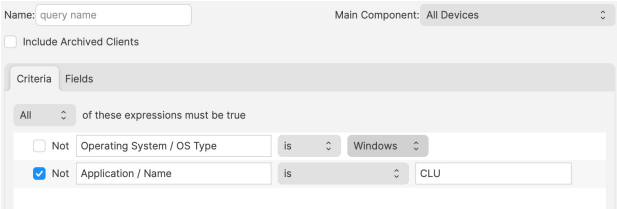
Example 2

Here is another example, but with just one group.

Same principle, but this time with a file level self-healing Fileset of this same application for Windows. This time CLU.exe, version 1.0

The criteria for the Smart Group association this time has been set as:

- Device OS is Windows
- Device does not have this software installed



Windows devices without this software will enter the Smart Group, receive the Fileset, adding the exe and any other supporting files to the designated folder. Subsequently, the device will check back in, reporting the software is now installed. At next Smart Group refresh, the device no longer meets criteria and the device leaves the Smart Group.

This is now where the issue occurs. As a self-healing Fileset, the software will be removed on disassociation of the Fileset. The user will lose the software, but at next refresh, the device will re-enter the group, causing the software to instal once more.

Again, this will continue to occur, with the software constantly being removed and re-added.

> For each of these examples, should certain features not be used, for example, self-healing in the latter example. Absolutely not. Self-healing is a key aspect to FileWave Fileset deployment. Instead, care should be taken, when considering the criteria of Smart Groups, to be sure that Policy Loops do not occur.

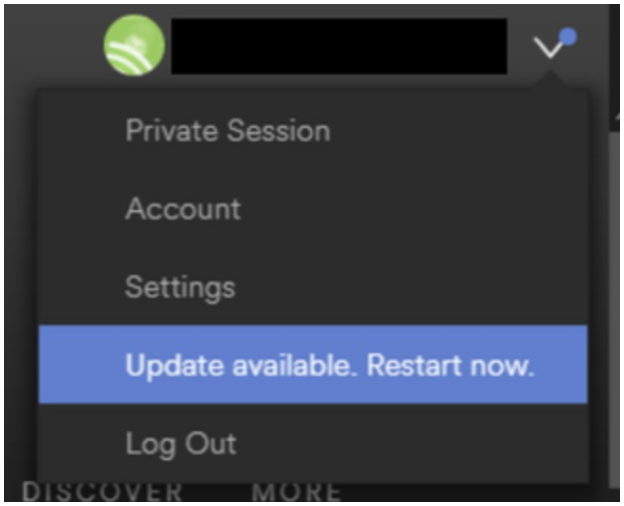
Updating 3rd Party Software

What

Naturally devices require software and that software needs updating. The question is how.

For managed software, e.g. Apple VPP Apps, updates occur automatically, but other software deployed using PKG, MSI, EXE or file level Filesets then what happens. Essentially, there are a couple of key choices.

Some software attempts to auto update, which may or may not work, in particularly when users are not admins, whilst other software will always require updates pushed out.



Why

Back to the choice. Allow software to auto update or prevent such activity and choose to build new Fileets to push out updates. Making that choice, though, can be impacted by other factors.

For example:

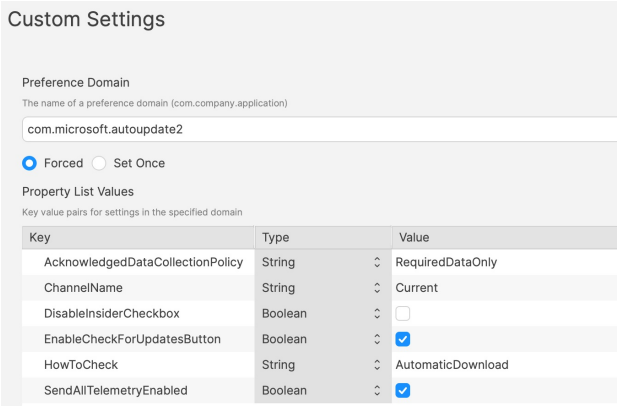
- Is the software being deployed critical to business
- Are there company restrictions that prevent software being updated before approval
- Does the software even have an auto updater
- How easy is it to prevent the software from updating, where an auto update does exist
- Do you trust the software supplier enough to allow updates to occur without prior testing
- What impact could occur if an update went wrong and what is the rollback option
- Is a reboot required after the update

How

Those are some considerations. Now to consider some finer details.

Denying AutoUpdates

For software that has no autoupdates, this is already a consideration, but denying updates takes some greater work. Firstly, a requirement to locate how the update works and then how to prevent it.



Most software vendors are likely to have either a Windows registry entry or a macOS plist preference file that can be configured to prevent the updates. Identifying the file to alter and the values to set, in some instances can be easy to address. In fact, many other Admins often post these settings or they may be available from vendors. However, sometimes this information isn't readily available.

Name	Type	Data
(Default)	REG_SZ	(value not set)
ACFullAccessOnLoginScreen	REG_DWORD	0x00000000 (0)
Always_Online	REG_DWORD	0x00000001 (1)
Apply_Blacklist_Or_Whitelist_On_Meeting	REG_DWORD	0x00000000 (0)
AutorecordRemoteControlEnforced	REG_DWORD	0x00000000 (0)
AutoUpdateMode	REG_DWORD	0x00000000 (0)

Various methods exist, but generally, the process is look at files before and after making preference changes made available through the software GUI if available. Fileset Magic is one method to assist with this process. This option of the FileWave application takes a snapshot of the device and then after changes are made, a second snapshot is taken. It is then a case of comparing the before and after to see what has changed.

Allowing AutoUpdates

One key question is, does micro management of updates of all applications really improve management of devices. Many applications are not business critical or a bad update could easily have minimal impact. With that in mind, why not allow updates. Indeed, Apple VPP Apps leave little choice. Of course, just making that decision does not mean auto updates are on by default. As such, the same process to calculate how to disable updates may need to be actioned to work out how to enable updates.

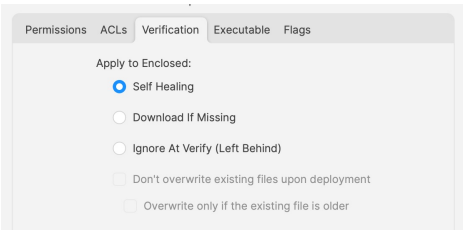
Add to this, as eluded to prior, will the auto update work if the user is not an Admin. This needs to be tested, but if not, then the same process used for denying updates would likely be required.

Considerations

For either method, there are some additional considerations, which mostly centre around self-healing.

Denying Autoupdates

When using a file level Fileset to deploy software, files should be set as self-healing.



Not only does this ensure the most efficient delivery of files from server to devices, it adds some greater benefits. When the association of the new version is associated, in the same Model, the older version should be disassociated.

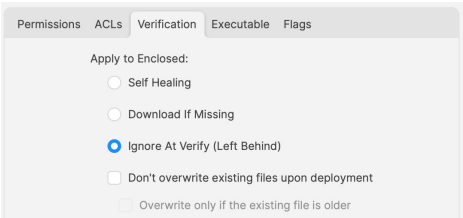
If both Filesets are left associated, updated files will be replaced, whilst new files will be pushed to devices, however what about files that the software no longer uses. If the older Fileset is not disassociated, these files will be left behind. Although this may seem harmless enough, actually they can be very damaging. Developers of software would not expect those files to be in existence with the new application and with thousands of lines of code, it could be easy enough that these files still have references and could cause havoc with the newer version.

Allowing AutoUpdates

So, how about handling software where the autoupdates is allowed to occur. In this instance, if file level Filesets are used to deploy the application, self-healing would be completely the wrong choice.

When software autoupdates, files will be altered. When a verification occurs, any altered files set for self-healing, will be replaced

with the older files. Although clearly undesirable, this isn't the same as downgrading the software, such that it would still function. Self-healing will also return any files that were removed by the updater. This brings back the condition of files unexpectedly installed, which again could cause the software to act irregular or not even start. As such, Ignore at Verify would be the ideal selection.



Ignore at Verify brings about 2 additional items for attention.

Un-installing.

Un-installers can come in differing forms, one of course is by way of self-healing. However, using the allowed auto-updater example, self-healing is not an option. This means an alternate method would be required to remove the software. Of course, FileWave can be used to achieve this, for example, with un-installer scripts.

Rollback

Where software is auto updating, the only version available in FileWave will likely be the same version originally pushed (unless updated more recently as a Fileset). Therefore, if there was a need to rollback to a prior version, som additional work would be required, which would take time before being deployable.

Overview

Each method has its own merits, but being aware of the pros and cons and how to deal with these, provides the armoury for successful application management.

Un-install Filesets

What

There will likely come a time when software installed is no longer required. What options are there for removing installed software.

Options

Self-Healing

The verification settings ‘Self Healing’ and ‘Download if Missing’ ensure that any files included within the Fileset are removed on disassociation from the devices, so naturally files will be removed.

MSI

Windows MSI Filesets are unique. They provide an un-install option for developers and FileWave benefits from this feature, if enabled per Fileset. On disassociation, the MSI un-installer feature may be triggered.

PKG/EXE

Unlike MSI installers, PKG and EXE installers have no built-in un-installer and the files installed are not part of the Fileset, just the PKG or EXE itself can be set with verification settings. In this instance, if the developer of the software does not offer an additional PKG/EXE or script designed to remove the software, it is usual to self-build out some kind of script to remove those items installed.

Apple VPP & Android Play Store

Disassociation of these Filesets triggers a command to remove the Application

Other Files

When software is opened by the user, additional files can be created, which are not part of the original software or included in any of the above installer types. If desired to remove these, then again some kind of self-build method would be required.

Why

Removing software and supporting files keeps devices clean and helps ensure the users are productive, using chosen software and making the sharing of files is simplified; along with other reasons, like security, etc.

When

Identifying and building un-installers is one part of this process, but when the un-installer runs is key. It may seem obvious that the un-installer should run when Filesets are no longer associated, but it is not quite that clear cut.

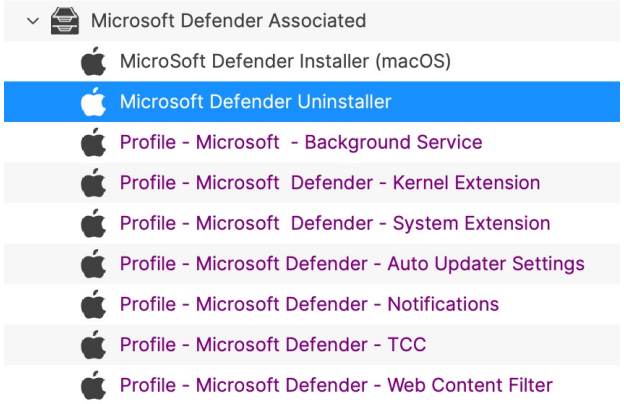
For example, consider the idea of writing a script to remove an application. Typically, a pre or post un-installer script could be set within a Fileset. It may seem natural to add this script within the same Fileset as the installer of the software. However, this can raise concerns.

Over time, developers will provide updates to software. Where autoupdating of software is prevented, new Filesets or Fileset Revisions will be built to push out these updates. Self-Healing handles swapping between Fileset or Revisions, since any files that match between the two differing Filesets remain untouched. However, any containing pre or post un-installer scripts will run at this time, which is likely undesirable.

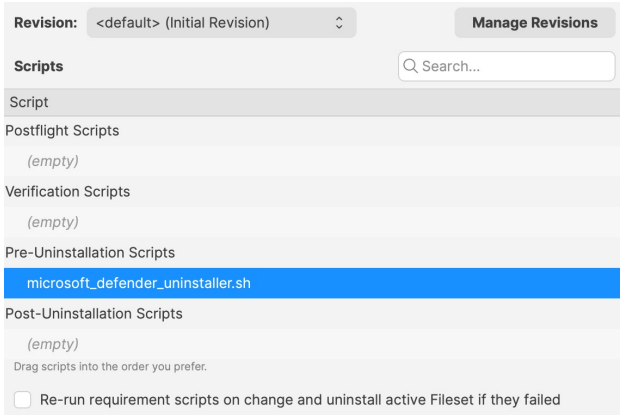
How

There is an installer Fileset, a method to uninstall the Fileset, so how to make sure these run only when desired. One such method is the use of Fileset Groups.

Using an example, the following diagram shows a Fileset Group for Microsoft Defender, which includes an Installer Fileset, some necessary Profiles and an Un-installer Fileset. Association will be with this Fileset Group.



The un-installer is scripted:

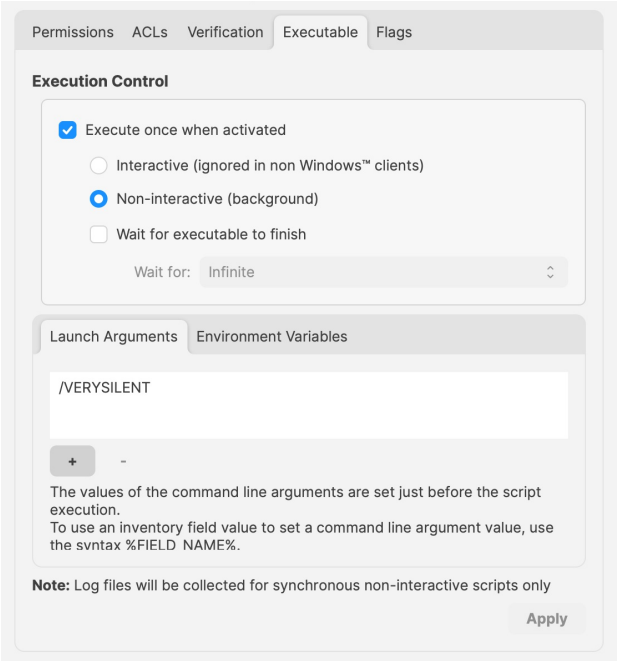


Since the Fileset Group is associated with devices, the installer can easily be updated. Either a new Fileset can be dropped into this group and the current Installer Fileset removed or the Installer could have an alternate Revision added and swapped over. However, when removing the Fileset Group association with devices, this will trigger this Pre-Uninstaller Script and only at this point in time will the software be removed.

PKG/EXE Un-installers

Where developers supply un-installers, pre-packaged in an installer PKG, building out auto running PKG Filesets would require the association to this un-installer only be assigned when disassociation occurs from the installer. Planning this becomes very complex. Instead, the PKG can be added to an empty Fileset and an un-installer script can be used to trigger the PKG file, akin to the above Microsoft Defender example.

Similarly, EXE installers are usually triggered from the Fileset Executable options:



EXEs built to un-install software however have the same issue as PKG Filesets. Pre or Post Un-installer Scripts yet can be used instead though. Again, upload the un-installer EXE into an Empty Fileset and add an un-installer script inside this same Fileset to trigger this EXE; creating a separate un-installer Fileset that can be used in a Fileset Group as demonstrated above.

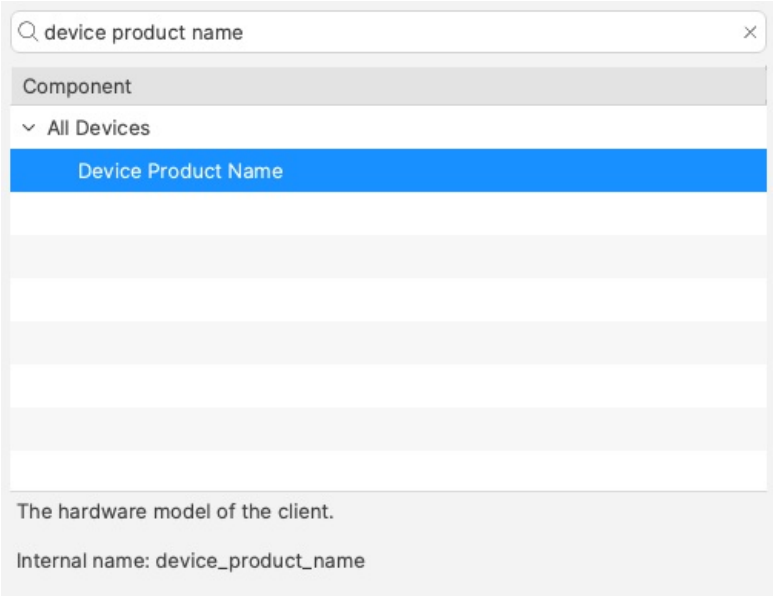
Inventory Items in Scripts

What

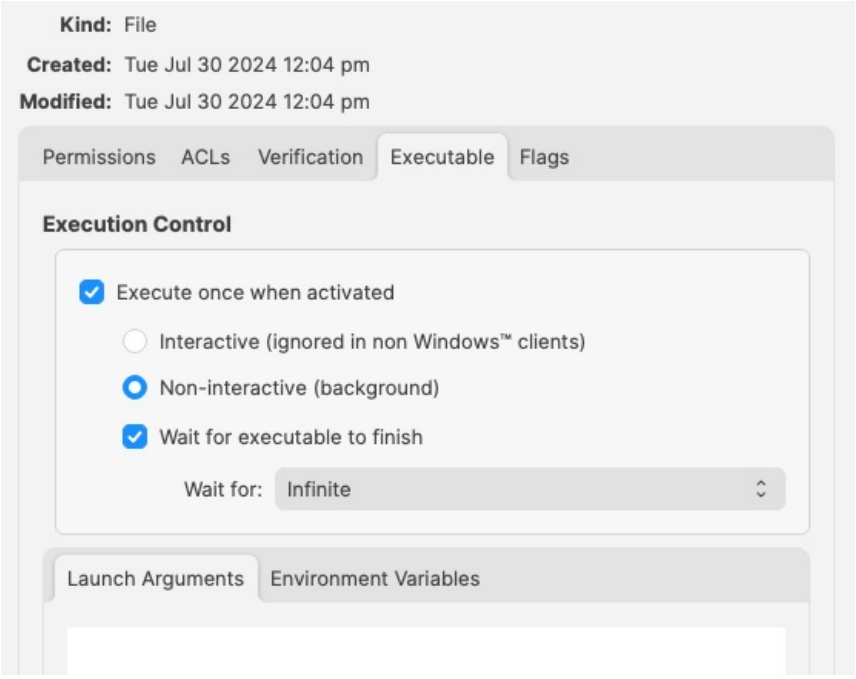
- Each Inventory Items has an Internal Name, including Custom Fields which provide extended inventory
- The Internal Name can be used to reference an Inventory Item in Scripts
- These Internal Names should be added to either the Launch Arguments or Environment Variables of the Script
- This applies to all script types, be that other Custom Fields, Policy Blocker Scripts or Fileset Scripts

When

Internal Name of an Inventory Item may be located from the Inventory Query Editor. Example shows the Internal Name: ‘device_product_name’



This may then be added into a Script, by way of either a Launch Argument or Environment Variable



But, which should be used?

How

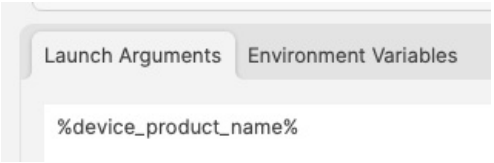
In some respects it does not matter which is used, however, for easy reference consider the following:

- Launch Arguments are referenced by their numerical position
- Environment Variables are referenced by a chosen name
- Custom Fields have an abbreviated name and a full name. Custom Field names could overlap with a built-in Inventory Item.

Built-In Inventory

In general, recommendation here is that of Environment Variables. This makes reading the script easier without having to redefine new names within the script for Launch Argument positions.

For example:



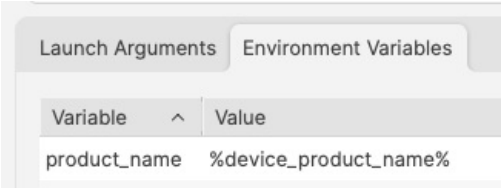
Could be referenced in a script as:

macOS shell	echo $1
Windows Powershell	echo $args[0]

But to make the parameters more easily recognisable for anyone reading the script, it could be desirable to name them:

macOS shell	`product_name="$1"` `echo $product_name`
Windows Powershell	`$product_name="$args[0]"` `echo $product_name`

References to the provided inventory parameters in the script now makes more sense, but as mentioned, Environment Variables take this a step further:



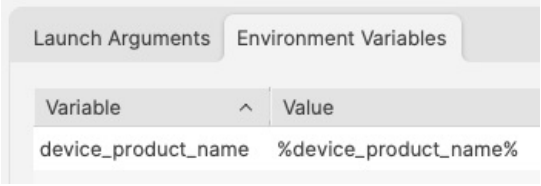
A variable name is already defined and this can be referenced in the script directly

```
echo $product_name
```

Improvements

First Improvement

To improve the readability of the script further, consider setting the variable name to match the value, e.g:

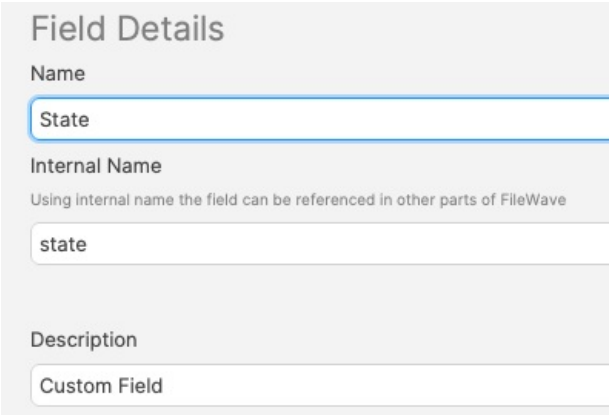



```
echo $device_product_name
```

Second Improvement

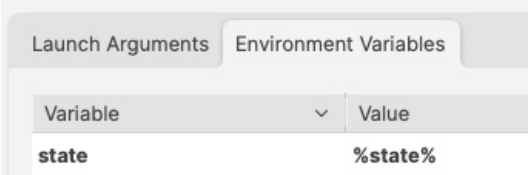
When referencing a Custom Field in a script, it could be referenced in one of two ways.

Example Custom Field: State



> Note, the description has been used to indicate this is a Custom Field. Inventory Query editor shows Description.

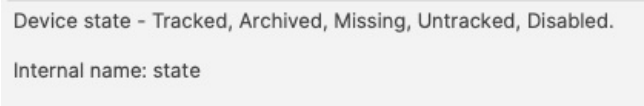
This could be referenced with:



and

```
echo $state
```

However, there is a built-in Inventory Item called State. So there are now two Internal Names of ‘state’

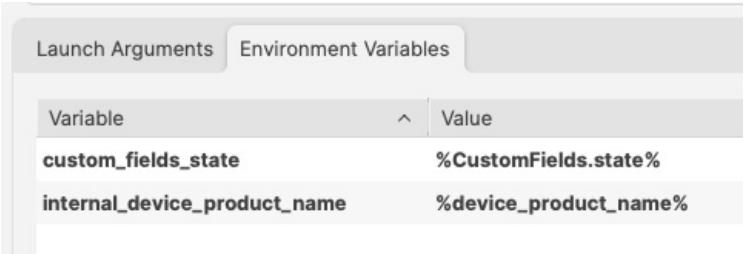


The above scripted example for 'state' would actually report the built-in value, not the Custom Field. There is, though, a hidden prefix that can be used.

This Custom Field could be referenced as either:

- %state%
- %CustomFields.state%

The latter prevents unexpected collusion with the matching Internal Name. Hence, to make the parameters more obvious when reading...




```
echo $custom_fields_state
echo $internal_device_product_name
```

Notice, despite no prefix existing for built-in Inventory Items, by including a prefix for both variables in the Environment Variables definitions, reading the script will be much clearer.

Anyone reading the script is now aware that state is a Custom Field, without having to cross reference anything. Likewise, the reader also is aware that the device_product_name also comes from Inventory, again, without any cross reference necessary.

Unknown Inventory

Not all Inventory Items are available as parameters.

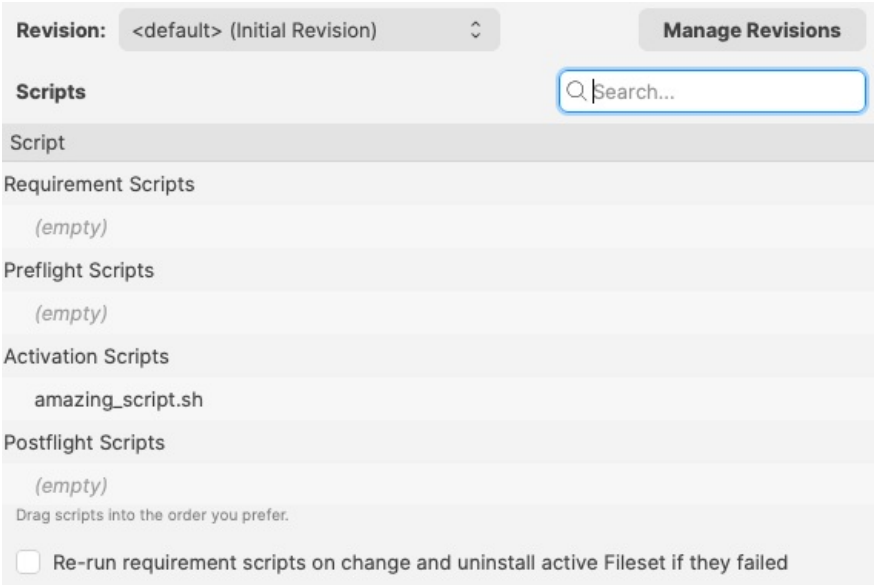
The FileWave Client builds out the report of items to inventory and return to server. Additionally, all Custom Fields, including those server-side (Administrator Custom Fields), are available to the client. However, inventory returned by MDM is not available, since the client is unaware of these values, they are pure server-side.

%CustomFields.location%

Script Logging

What

For scripts added to FileWave Filesets using the Script view, logging is enabled by default.



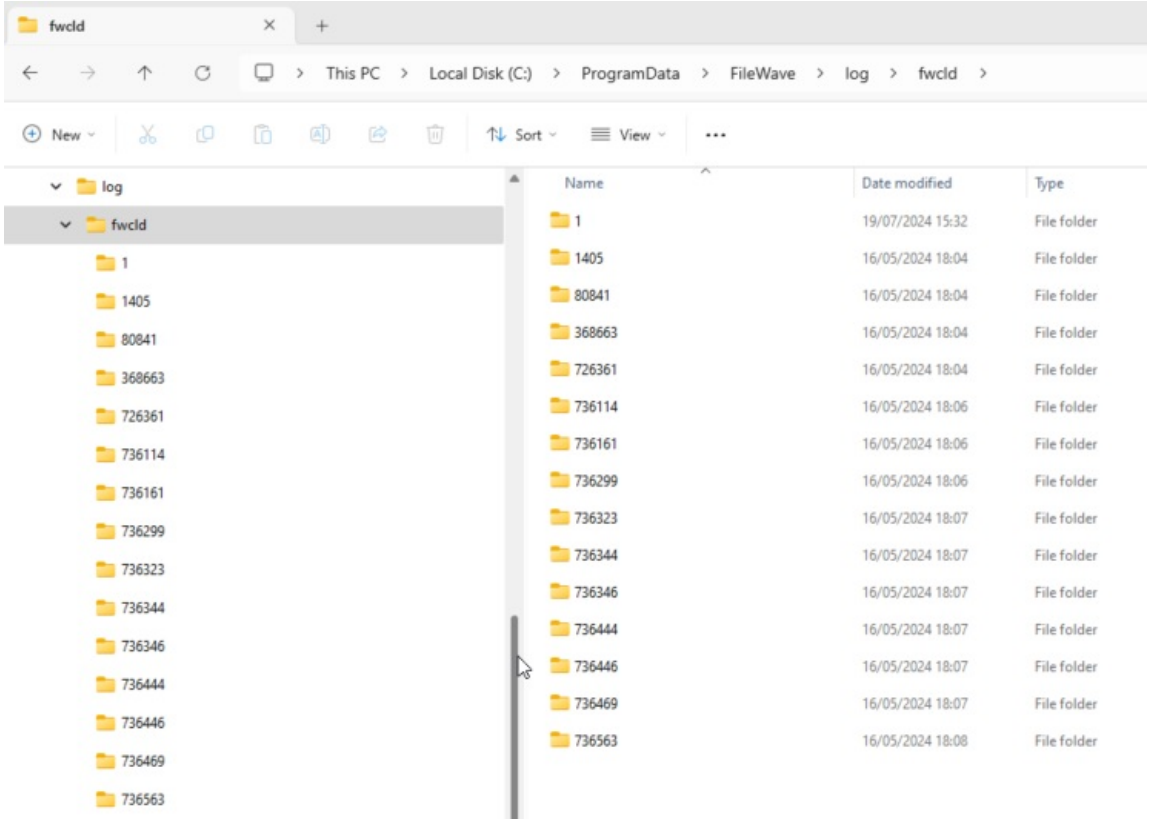
When

Each time a Script (as built above) is actioned on a device, a log file is created or added to, with anything that the script outputs. The logs of these Scripts are located in the following directories, within subfolders named after the Fileset ID:

macOS

```
# ls -al /private/var/log/fwcld/
total 0
drwxrwxrwx  17 root  wheel   544 Mar  5 16:01 .
drwxr-xr-x  83 root  wheel  2656 Jul 31 08:27 ..
drwxrwxrwx   3 root  wheel    96 Aug  1  2023 1
drwxrwxrwx   5 root  wheel   160 Sep 25  2023 54231
drwxrwxrwx   6 root  wheel   192 Sep 26  2023 54235
drwxrwxrwx   3 root  wheel    96 Nov  9  2023 54367
drwxrwxrwx   3 root  wheel    96 Nov  9  2023 54368
drwxrwxrwx   3 root  wheel    96 Nov  9  2023 54374
drwxrwxrwx   3 root  wheel    96 Nov  9  2023 54379
drwxrwxrwx   3 root  wheel    96 Nov 10  2023 54384
drwxrwxrwx   3 root  wheel    96 Nov 10  2023 54396
drwxrwxrwx   3 root  wheel    96 Nov 10  2023 54401
drwxrwxrwx   3 root  wheel    96 Nov 10  2023 54406
drwxrwxrwx   3 root  wheel    96 Dec 15  2023 54417
drwxrwxrwx   3 root  wheel    96 Dec 15  2023 54419
drwxrwxrwx   3 root  wheel    96 Dec 15  2023 54421
drwxrwxrwx   3 root  wheel    96 Mar  5 16:01 55188
```

Windows



Example:

macOS example, but the principle is the same for Windows.

Consider this simple shell script that runs a command to output the username that ran the command:

whoami.sh

```
#!/bin/zsh
whoami
exit 0
```

On running the command, the user running the command will be reported. For example, running this locally on a device might reply:

```
% whoami
sholden
```

Viewing the log generated by FileWave:

```
# cat /private/var/log/fwcld/54421/whoami.sh.log
----------------------- HEADER - Date: (Fri Dec 15 2023) - Time: (16:40:21) -----------------------
root
----------------------- FOOTER - Date: (Fri Dec 15 2023) - Time: (16:40:22) - Exit code: (0) ---------------------
--
```

The output presents:

- Header and Footer with timestamps
- Footer with exit code value
- Any output from the script between the Header and Footer

Improvement

The Script may or may not provide output, depending upon the command used. However, why not add additional echo commands (or similar) to output extra details to provide more information from the script running.

Here is an example of a Fileset Requirement Script, waiting for confirmation of a Profile to be installed before activating the Fileset:

```
#!/bin/zsh

found_profile=""

while [ $# -gt 0 ]
do
        found_profile=$(profiles list all | awk -v search=$1  '$0 ~ search {print $NF}')
        if [ ! -z $found_profile ]
        then
                echo "Found installed profile: $found_profile"
                exit 0
        else
                echo "Did not find $1"
        fi
        shift
done

exit 1
```

The script is outputting additional information, showing the ID of the Profile, found or not. On success, exit 0, else exit 1.

Requirement Scripts will retry every 2 minutes, until successful, unless coded otherwise

```
----------------------- HEADER - Date: (Thu Jul 31 2024) - Time: (11:03:12) -----------------------
Did not find ml1063.local.aa0bd493-960d-4dc0-9631-a3fea189191e.Configuration.aa0bd493-960d-4dc0-9631-a3fea189191e
Did not find ml1063.local.5a57bcb9-7293-4cba-a20b-126eb2660b25.Configuration.5a57bcb9-7293-4cba-a20b-126eb2660b25
----------------------- FOOTER - Date: (Thu Jul 31 2024) - Time: (11:03:12) - Exit code: (1) ---------------------
--

----------------------- HEADER - Date: (Thu Jul 31 2024) - Time: (11:05:12) -----------------------
Found installed profile:  ml1063.local.aa0bd493-960d-4dc0-9631-a3fea189191e.Configuration.aa0bd493-960d-4dc0-9631-
a3fea189191e
----------------------- FOOTER - Date: (Thu Jul 31 2024) - Time: (11:05:12) - Exit code: (0) ---------------------
--
```

On first attempt, the log shows two Profiles were searched and not found, with the script exiting a value of 1. On second attempt, the first Profile ID is now showing as installed and the script exited with a value of 0.

No Logs

Some scripts ran through FileWave, e.g. Policy Blocker Scripts, do not provide logs, with some mention in the Client Log alone, that the Script ran.

However, it is entirely possible to choose to create a custom log file within a script, for any script, and echo any output desired to provide additional logging.

> Consider how the script will grow and how to either overwrite or append appropriately.

Apple MDM

Profile Payload Planning

What

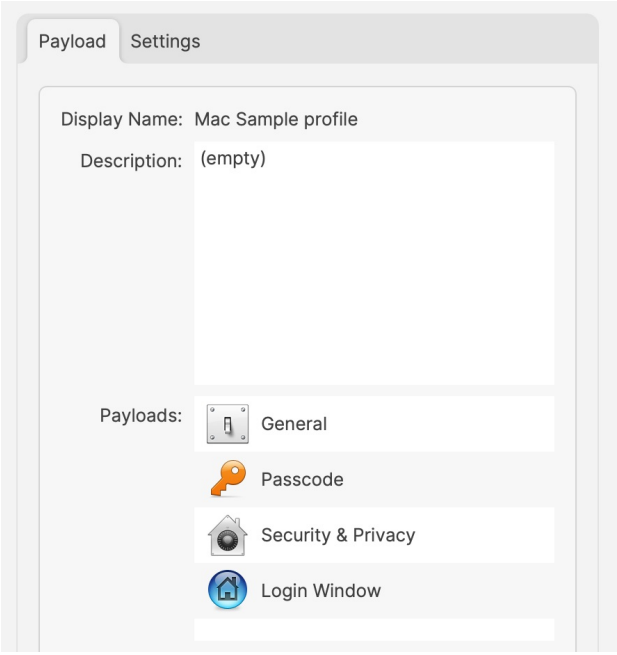
- Apple Profiles contain Payloads
- Payloads provide configuration options
- As of macOS 11, Profiles may only be delivered to devices which are also MDM enrolled. (MDM is the only enrolment option for iOS and similar OS types).

That's the fundamentals of Profiles in a nutshell, but there is more consideration.

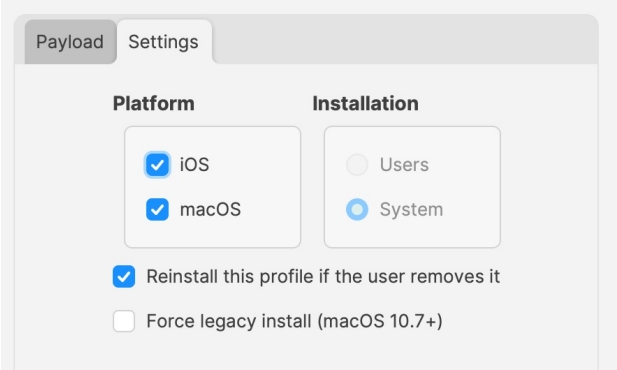
Key aspects:

- A Profile may contain multiple Payload types
- Multiple Profiles may be pushed to each device
- macOS devices have an additional option: should the Profile be set against the User or the System?

Example Profile with multiple Payloads:



For the same above Payload, the Settings show:



Apple's implementation is such, that only one local user can be managed (the first user after enrolment). However, any amount of directory users can be managed. This restriction applies to User set Profiles only.

Not all Payload types can be User or System. Some may only be User or only System, rather than the choice. From the screenshot above, the Settings show System is the only choice and is therefore greyed out.

Possibly, one of the most important consideration:

> Where multiple Profiles are assigned which contain the same Payload type (but differing settings), Apple do not guarantee the

experience. There used to be a suggestion for restrictive Payload settings, the most restrictive wins, but other Payload types have always had this mention.

How

Firstly, overlapping Payloads should be avoided, to ensure experience is by design, not luck.

Next, from the above, Profiles can contain many Payloads, but should they? Consider:

- Having an experience that is undesired in the Profile
- Needing to 'Force Reinstall' a Profile

Undesired Experience:

More items in the Profile makes it harder to identify anything occurring that is undesired.

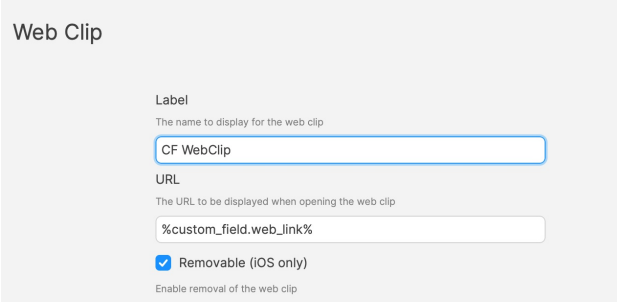
Removing the undesired experience temporarily, whilst identifying, involves removing the entire Profile, which could easily be undesirable in its own right. By creating multiple Profiles with different Payloads, instead of one massive Profile with lots of Payloads, makes identifying and resolving unexpected experiences, much more easily with less impact.

Force Reinstall

This option can be desirable for a few reasons, but consider this example:

- Profile contains a Payload with a value that is set by way of a Custom Field or Inventory item.
- Custom Field or Inventory Item is altered and devices need that new value to be applied within the Profile Payload

An example Web Clip Payload, using a Custom Field to populate the value:



When a Profile is altered, FileWave will note the Profile as Modified and the Profile will be redelivered with the new settings. However, when changing Custom Field or Inventory values, there is no change to the Profile. The Payload referencing the Custom Field or Inventory item is still referencing this, it is only during delivery that the value is noted and entered in the Profile. As such, if the referenced Custom Field or Inventory values are altered for devices, the current Profile will need to be reinstalled. A 'Force Reinstall' will ensure this occurs, but two things occur from this action. The current Profile is removed and the updated Profile is installed. Consider, what is the consequence of Profile removal?

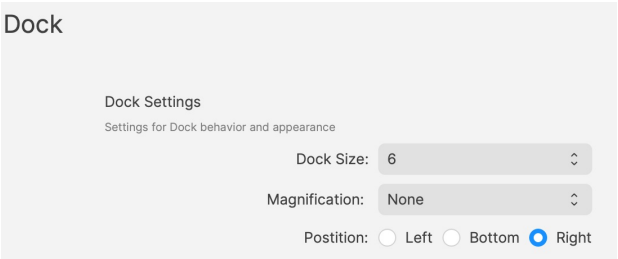
With the above in mind, always consider what is being included in a Profile and therefore keep each Profile lean in content; try not to overload too many Payloads into one Profile.

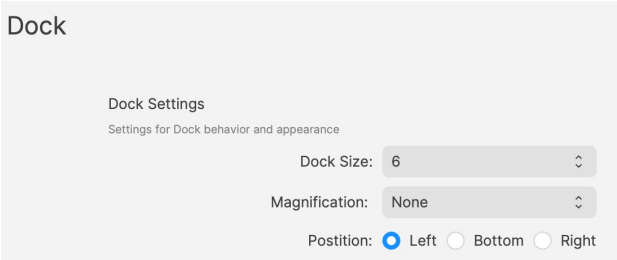
Overlapping Payloads

What is an overlapping Payload. This is when two or more Profiles are trying to manage the same thing, but with different settings. This shouldn't be confused with multiple allowed Payloads.

For example:

Profiles to manage the Dock. One Profile sets the dock on the right and the other on the left. This is overlapping and should be avoided:





If both of the above were assigned to a device, how could the device possible determine which should be obeyed?

Profile to provide certificates. One Profile provides one certificate and another Profile provides a different certificate. This isn't overlapping. Providing multiple certificates is desirable and need not be from one single Profile.

User vs System

Within the settings of Profiles is an option to define whether the Profile should apply to users or system. Some Payloads may be set as User or System only, but not either, whilst others may be either. A Profile must have a setting, so a default will be used when a Profile Payload is first added. Always check to confirm it is set as desired.

A Profile may only have one setting applied. FileWave will therefore prevent the addition of a User only Payload to a Profile already containing another Payload set as System. However, where Payloads may be either, if a Profile already contains a Payload, any additional Payloads that can be added will all be set with the same setting.

For example:

- Create a new Profile and add the Login Window Payload
- Save the Profile and re-open to observe the Settings (should be shown as System and greyed out)
- Create another new Profile and add a Dock Payload
- Save the Profile and re-open to observe the Settings (should show as either System or User, but defaulted to System).
- Change to User, save and re-observe the change

It can be seen that the Login Window is System only, yet the Dock Payload could be either

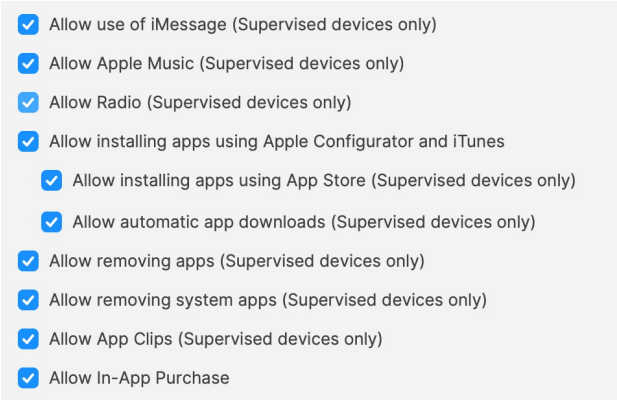
- Re-open the Login Window Profile created above and add the Dock Payload to this Profile
- Save and re-open to observe the Settings

The Settings remain as System and the applied Dock Payload will therefore be set for the System and not User. If a Dock Profile of User were required, it should not be included in a Profile that already contains a Payload that is set as System.

Why

Should all of the above be of consideration? Why would User be chosen over System? If System will work for all users, why not just set all Profiles as System where possible. However, what if the settings included were only for users, but not for a hidden Admin account. This local admin account is not managed by MDM. By setting System level, any Profiles built this way will impact this user, along with the managed local user and any directory users. This may be undesirable. Passcode policy could be an example.

Some Payload types require certain types of enrolment. Many Payload settings require Supervision, for example. macOS devices managed via User Enrolment, do not qualify as Supervised.



Planning

The above comes down to planning.

Profiles could contain multiple Payloads based upon functionality and for User or System determined targeting. Who needs to be managed? Local user (all or just managed) and/or directory users.

Consider the impact if a Profile were ‘Force Reinstalled’ or if it was deemed necessary to temporarily remove the association to one or more devices, for whatever reason.

Also give thought to how devices will be purchased and enrolled depending upon what needs to be managed. Using a BYOD scheme and only using User Enrolment will greatly reduce what can be managed, whist ADE(DEP) gives the maximum amount of control through Profiles.

Will the choice made, incur additional concerns over security, if desired management cannot be achieved?

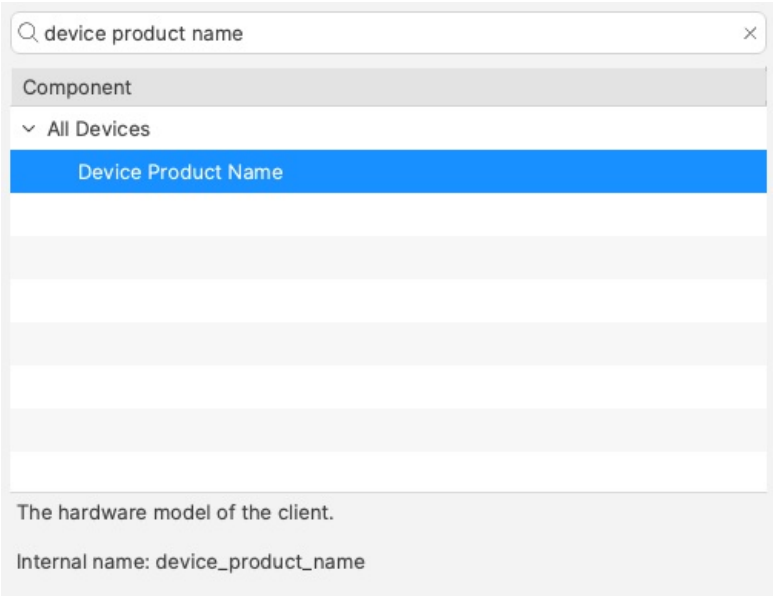
Inventory Items in Profiles

What

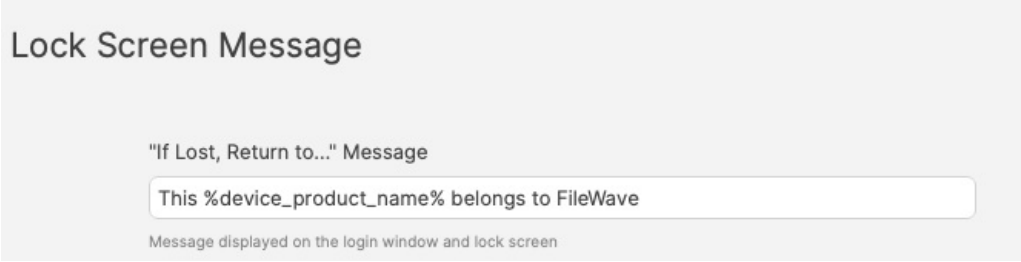
- Each Inventory Items has an Internal Name, including Custom Fields which provide extended inventory
- The Internal Name can be used to reference any Inventory Items in Profiles

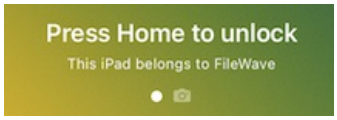
When

Internal Name of an Inventory Item may be located from the Inventory Query Editor. Example shows the Internal Name: ‘device_product_name’



This may be added into a Profile, effectively customising the Profile per device or user:

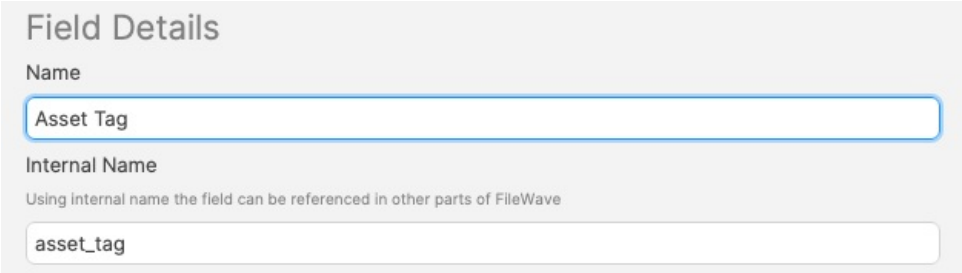




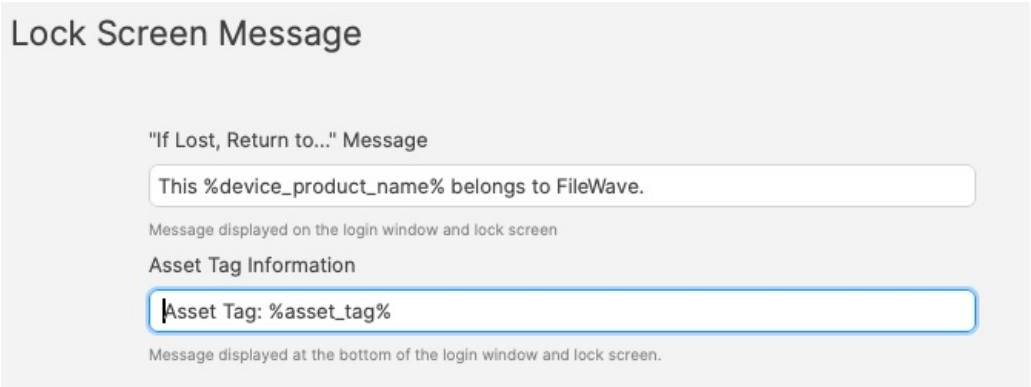
Custom Fields

Associated Custom Fields may also be used with Payloads settings of Profiles. Extending the above example, consider a Custom Field for Asset Tag:

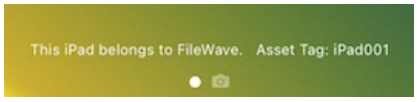
Custom Field Definition



Profile Payload



Lock Screen



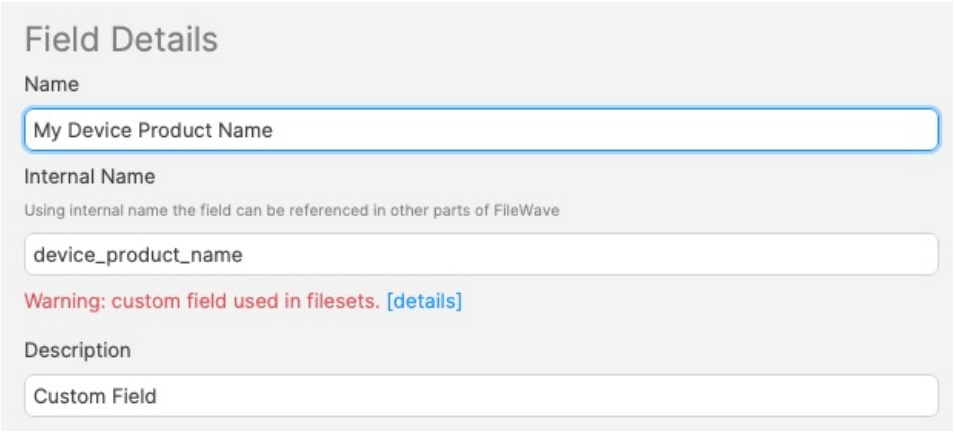
Improvement

When referencing a Custom Field in a Profile Payload, it could be referenced in one of two ways. From the above example, it could be either:

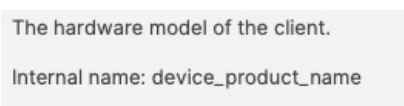
- %asset_tag%
- %custom_field.asset_tag%

The additional prefix indicates more clearly that this is a Custom Field Inventory Item. If there was an Inventory Item with a matching name provided by FileWave, the first item in the list would report the provided Inventory Item value for the device and not the Custom Field.

For demonstration, imagine creating a Custom Field called 'My Device Product Name' with Internal Name: device_product_name.



There are now two Inventory Items with the Internal Name: device_product_name:



and

Custom Field

Internal name: device_product_name

With the values:

Criteria | Fields | Dashboard

Drop here the fields you want to see in the query report ; change column order by moving column header.

My Device Product Name	Device Product Name
iPad 32GB	iPad

Altering the above example Lock Message to use both of these:

Lock Screen Message

"If Lost, Return to..." Message

FileWave Inventory: %device_product_name%

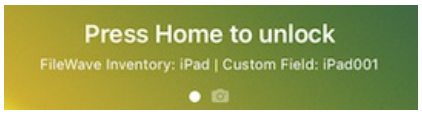
Message displayed on the login window and lock screen

Asset Tag Information

Custom Field: %custom_field.asset_tag%

Message displayed at the bottom of the login window and lock screen.

The device clearly demonstrates how the value without a prefix uses the FileWave provided Custom Field value:



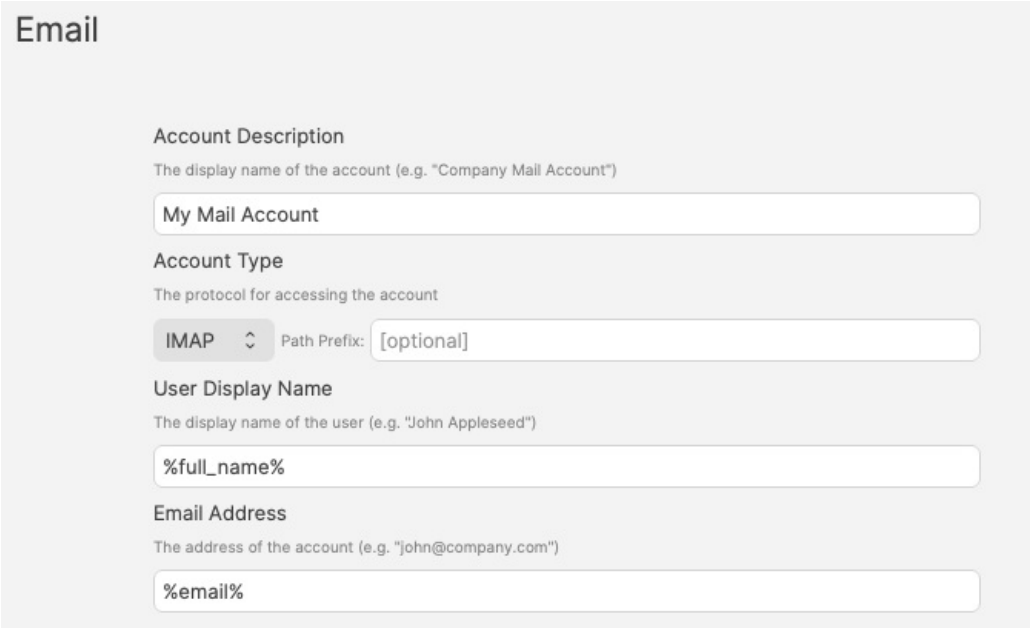
To prevent confusion with overlapping Inventory Items between Custom Field and built-in Inventory, always consider using the prefix for Custom Fields

User Customisation

Although FileWave doesn't manage users, if users are associated with devices, this extends the ability to customise Profiles for users.

User details from enrolment may be used, but to extend beyond this, LDAP servers set for extraction can greatly increase the Inventory Items available for Parameters via LDAP Custom Fields.

An example of Profile customisation for users:



One Profile can therefore be used for multiple devices, tailoring the Payload to the users of those devices.

Android EMM

Android Policy Planning

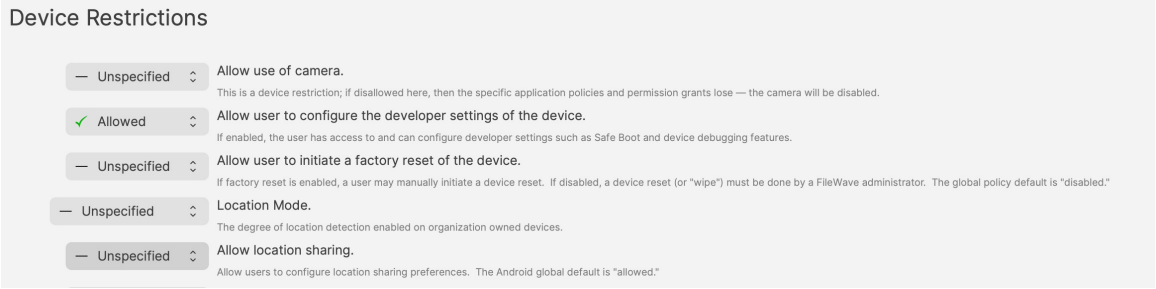
What

Android Policies provide a method of configuration.

Key aspects:

- Policies may contain multiple types of settings
- Multiple Policies may be pushed to devices

Example Policy, allowing Developer Settings, including setting Debug:



Possibly, one of the most important consideration:

> ⚠ Where multiple Profiles are assigned which contain the same Payload type (but differing settings), what could the possible expectation of experience be on the end device with these overlapping Policy Settings, noting that Apps potentially have their own settings also?

How

Firstly, overlapping Policies should be avoided, to ensure experience is by design, not luck.

Next, as mentioned above, Policies can contain many configurations, but should they? Consider:

- Having an experience that is undesired from the Policy
- Needing to remove and reinstall a Policy

Undesired Experience:

More items in the Policy makes it harder to identify anything occurring that is undesired.

Removing the undesired experience temporarily, whilst identifying, may involve removing the entire Policy, which could easily be undesirable in its own right. By creating multiple Policies with different settings, instead of one massive Policy containing everything, makes identifying and resolving unexpected experiences, much easier with less impact.

Remove/Reinstall

Perhaps something doesn't appear to be working as intended. To confirm the Policy, it may be desirable to remove that Policy, but what if other items in the Policy should not be removed, e.g. Certificates. Separating settings based upon functionality should help alleviate this potential problem.

Overlapping Policies

What is an overlapping Policy. This is when two or more Policies are trying to manage the same thing, but with different settings. This shouldn't be confused with multiple allowed Policies.

Greater detail on this can be found in our KB:

https://kb.filewave.com/books/android/page/android-emm-policies-and-permissions

In essence, overlapping Policies should be avoided. An experience may appear correct, due to an overlap. If one of the Polices were removed, the experience may unexpectedly alter if there was no awareness of this.

A Policy to provide certificates is an example of a potential multiple allowed Policy. One Policy provides one certificate and another

Policy provides a different certificate.

Planning

The above comes down to planning.

Policies could contain multiple settings based upon functionality.

Consider the impact if there was a requirement to remove and subsequently reinstate a Policy, for whatever reason.

Also give thought to how devices will be purchased and enrolled depending upon what needs to be managed. Enrolment types can also impact items managed.

Will the choice made, incur additional concerns over security, if desired management cannot be achieved?