

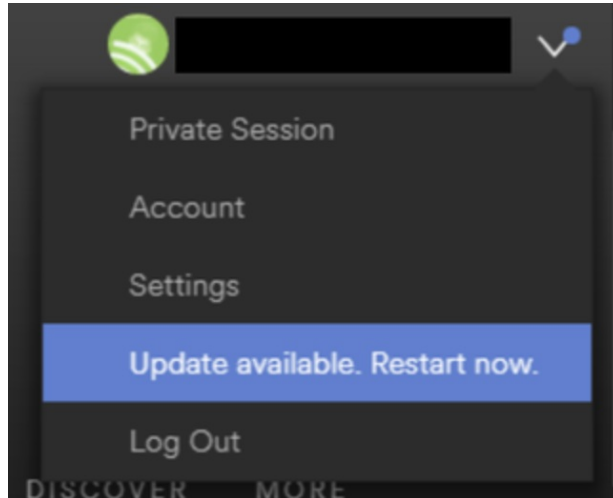
Updating 3rd Party Software

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.

Custom Settings

Preference Domain

The name of a preference domain (com.company.application)

com.microsoft.autoupdate2







☒ Forced
 ☐ Set Once

Property List Values

Key value pairs for settings in the specified domain

Key	Type	Value
AcknowledgedDataCollectionPolicy	String	RequiredDataOnly
ChannelName	String	Current
DisableInsiderCheckbox	Boolean	<input type="checkbox"/>
EnableCheckForUpdatesButton	Boolean	<input checked="" type="checkbox"/>
HowToCheck	String	AutomaticDownload
SendAllTelemetryEnabled	Boolean	<input checked="" type="checkbox"/>

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.

Permissions

ACLs

Verification

Executable

Flags

Apply to Enclosed:

☒ Self Healing
 ☐ Download If Missing
 ☐ Ignore At Verify (Left Behind)
 ☐ Don't overwrite existing files upon deployment
 ☐ Overwrite only if the existing file is older

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.

Permissions

ACLs

Verification

Executable

Flags

Apply to Enclosed:

☐ Self Healing

☐ Download If Missing

☒ Ignore At Verify (Left Behind)

☐ Don't overwrite existing files upon deployment

☐ Overwrite only if the existing file is older

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.