Managing software updates for IT

June 2024 v1.0

Contents

Intro to managing software updates **4**

Introduction 4

Types of software updates 5

Software update security 7

Caching software updates 7

Software update requirements 8

Software update process **9**

Software update detection 9

Downloading and preparing software updates on iPhone and iPad 10

Downloading and preparing software updates on Mac 11

Software update notifications 11

Installing a software update 11

Installing and enforcing software updates **12**

Requiring a minimum version during MDM enrollment 12

Managing the availability of software updates 13

Installing software updates automatically 14

Requiring authorization by an administrator on macOS 15

Enforcing software updates 15

Software update implementation details for MDM developers 18

The importance of leveraging software update declarations 19

Using the Apple Software Lookup Service 20

Sending a status report to the MDM solution 22

Requesting a specific minimum software version during MDM enrollment 24

MDM settings for software updates 26

Configuring automatic software updates with MDM 27

How MDM handles Rapid Security Responses 28

Deferring a software update with MDM 29

Enforcing software updates with MDM 31

Phases of Apple software update enforcement 33

Testing software updates with the AppleSeed for IT beta program 38

Enrolling a device in a beta program 39

Copyright 44

Intro to managing software updates

Introduction

Keeping devices up to date is an essential part of managing devices in an organizational environment. Software updates allow users to quickly benefit from new features and help keep the deployment secure.

Apple operating systems provide a suite of built-in capabilities to make this process seamless for the user, while providing IT administrators the necessary controls to manage software updates at scale across devices. A key technology is the mobile device management (MDM) framework, which has been updated with the introduction of declarative device management. Declarative device management increases autonomy and proactivity of devices and enables an entirely new approach to managing and enforcing software updates.

This document details the managed software update process based on declarative device management for IT administrators and MDM developers and covers Apple's operating systems for iPhone, iPad, and Mac.

Types of software updates

For the purposes of this document, software updates are divided into three types: software updates, software upgrades, and Rapid Security Responses (also known as *supplemental updates*).

Software updates

Updates consist of frequently released patches that help secure or enhance the current operating system and that are designed to keep devices protected. These minor updates use a version numbering scheme that includes at least one decimal point—for example iOS 17.1.

Software upgrades

Upgrades typically make important changes to an operating system over the previous version—in functionality, user interface, and general appearance.

Upgrades are released much less frequently than updates, and they can take a while to install because of their size. Older devices may not be eligible for upgrades if they don't have the necessary hardware capabilities to support the new software.

Typically, upgrades of Apple's operating systems use a whole number integer—for example, iOS 17. macOS also uses a whole number integer (for example, the systems macOS 13 and macOS 14), and may be named—here, Ventura and Sonoma. Some upgrades include the first update automatically. An example is iOS 16.1 where the upgrade includes the update automatically, because iPadOS 16.0 was not released separately.

Rapid Security Responses

Rapid Security Responses (RSRs) are a different type of software release for applying security fixes to devices more frequently by not requiring a full software update. Rapid Security Responses are delivered only for the latest versions of iOS, iPadOS, and macOS. Because of this, the operating systems must be running the latest version. In iOS 18, iPadOS 18, and macOS 15, available Rapid Security Responses are part of the software update, which removes the need to perform multiple installations consecutively—first the update, then any available Rapid Security Responses.

Rapid Security Responses that involve the operating system require the device to restart. In macOS, if the Rapid Security Response also includes an update to Safari, it may be made available with just a relaunch of Safari, though a restart is required to make this content available to the rest of the operating system.

Users can remove responses from within Settings > About > iOS (or iPadOS) Version or > About This Mac > More Info and by clicking on the Info button next to the version number.

Each Rapid Security Response is versioned relative to its base operating system version, starting with *a*, then *b*, and so on. Within a line of Rapid Security Responses, successive ones always include the changes from previous ones. Subsequent operating system updates and upgrades include the content from all the Rapid Security Responses that were issued for the previous operating system version. The table below shows an example of how minor operating system updates affect Rapid Security Responses.

Note: These examples aren't to be considered actual Rapid Security Response versions.

Operating system version installed	Rapid Security Response version examples	Description
iOS 17.2 macOS 14.1	a b	Two Rapid Security Responses (*a* and *b*) were available.
iOS 17.3 macOS 14.2	a b c	Three Rapid Security Responses (*a*, *b*, and *c*) were available. iOS 17.3 and macOS 14.2 include the content from the two RSRs available for iOS 17.2 and macOS 14.1.
iOS 17.4 macOS 14.3	a	One Rapid Security Response (*a*) was available. iOS 17.4 and macOS 14.3 include the content from the three RSRs available for iOS 17.3 and macOS 14.2.

Important: Unless otherwise noted, the phrase *software update* is used in this technical paper to refer to all software updates as a whole—that is, to include updates, upgrades, and Rapid Security Responses.

Software update security

Apple devices use a secure software update process, which verifies the integrity of an update, personalizes it for the specific device, and helps prevent downgrade attacks. To help protect the user's privacy and integrity of their data, the user’s data volume is never mounted during a software update or upgrade. For more information on the security of Apple software updates, see Secure software updates in Apple Platform Security.

Note: Because of dependency on architecture and system changes to any current version of Apple operating systems (for example, macOS 14, iOS 17, and so on), not all known security issues are addressed in previous versions (for example, macOS 13, iOS 16, and so on).

Caching software updates

Software updates for Apple devices can be cached on a Mac using macOS 10.13 or later with Content Caching turned on. This allows devices to download the necessary files from the content cache instead of using an internet connection. However, devices still must contact Apple servers to complete the update. For more information, see:

- Intro to content caching
- Plan for and set up content caching
- Use DNS TXT records with content caches
- Apple Support article: Content types supported by content caching in macOS

Software update requirements

Network requirements

Apple devices require access to specific internet hosts to download software updates and personalize the operating system during a software update for the specific device. Here's how Apple devices connect to hosts and work with proxies:

- Network connections to hosts are initiated by the device, not by hosts operated by Apple.
- Apple services will fail any connection that uses HTTPS Interception (SSL Inspection). If the HTTPS traffic traverses a web proxy, disable HTTPS Interception for the appropriate hosts.

For more information on the network connections required during the software update process, see the Software updates section of the Apple Support article Use Apple products on enterprise networks.

Power requirements

Depending on the type of the update and how it is initiated, devices must be connected to power or have the following minimum battery charging level to download, prepare, and install a software update:

Device type	**Minimum battery percentage required for user-initiated software updates and upgrades**	**Minimum battery percentage required to install automatic software updates***	**Minimum battery percentage required for Rapid Security Responses**
iPhone	20%	30%	20% (5% when connected to power)
iPad	20%	30%	20% (5% when connected to power)
Mac with Apple silicon	20%	50%	10%
Intel-based Mac	50%	50%	20%

*To download and prepare an *automatic* software update, devices must be connected to power.

Space requirements

Devices must have sufficient space available to download, prepare, and install the update.

Terms and conditions

Users may also need to agree to updated terms and conditions to initiate a software update or upgrade on their devices. This doesn’t apply to updates enforced by MDM on supervised devices.

Software update process

The process to perform a software update involves multiple steps—from the detection of an available update, to downloading and preparing the update, to the actual installation.

Software update detection

After an update becomes available for iPhone, iPad, or Mac, it’s detected in one of the following methods:

- After the user opens the Software Update settings
- Every 24 hours
- After a device restarts

If the device is using the default configuration and the requirements for a software updates are met, users can navigate to Settings (on iOS and iPadOS) or System Settings (on macOS) to download, prepare, and install the update at a time convenient for them.

Downloading and preparing software updates on iPhone and iPad

When connected wirelessly, iPhone and iPad devices use an over-the-air (OTA) update method that downloads only the components required to complete an update. This update improves network efficiency and enables faster downloads.

iOS and iPadOS software updates are made available in two phases, which determine how they can be downloaded.

iPhone and iPad can download and prepare software updates when connected using Wi-Fi and when connected using cellular:

- When connected wirelessly using Wi-Fi:
  - *Initial availability:* Manually by the user from Software Update Settings.
  - *General availability:* Manually by the user or using automatic software update downloads.
- When connected wirelessly using cellular:
  - *Initial availability:* Manually by the user when their device is connected to a 5G cellular network and has Allow More Data on 5G turned on.
  - *General availability:* Manually by the user unless it's deferred by the carrier on the specific cellular network. In this case, users are informed that a Wi-Fi connection is required to download the update.
- *When physically connected to a Mac or PC:* Through the Finder or Apple Configurator on a Mac or the *Apple Devices* app on a PC. In this case, a full copy of iOS or iPadOS is downloaded, prepared, and installed.

Downloading and preparing software updates on Mac

Similar to updates for iOS and iPadOS, updates for macOS are incremental updates using the OTA update method, which downloads only the components necessary to update the specific device. Incremental updates require a sealed system volume.

Software update notifications

After an update has been downloaded and prepared, a notification is shown to the user and a red icon appears over the Settings app (on iOS and iPadOS) or the System Settings app (on macOS). If automatic downloads are turned off, the notification and badge are shown when the update is detected.

Installing a software update

iPhone and iPad

If a passcode has been configured on the device, users are prompted for it to authorize the update.

Mac

In macOS 12.3 or later, any local user can authorize incremental software updates.
Before macOS 12.3, local administrators are required to perform software upgrades.
On a Mac with Apple silicon, users must be a volume owner to perform an update.

The Universal Mac Assistant (UMA) is a full installer of macOS and can be used on any supported Mac. The UMA is available using the `softwareupdate` command-line tool and requires authorization by a local administrator to be installed.

Automatic installation of software updates (not upgrades) and Rapid Security Responses

After the preparation is complete and users have automatic software updates or Rapid Security Responses turned on, the device uses on-device machine learning to determine a suitable time to install them.

To authorize the update, devices try to use the passcode or password from unlock to perform an automatic update, but may prompt the user instead.

In addition to other software update requirements, Mac computers must have no apps or processes running which block a restart. Mac laptops can have their lid closed.

Installing and enforcing software updates

Using declarative device management, organizations can configure different aspects of the software update process. This includes managing software update availability, enforcing software updates, enrolling devices into beta programs, and more.

Requiring a minimum version during MDM enrollment

Starting in iOS 17, iPadOS 17, and macOS 14, MDM solutions can enforce a minimum operating system version during Automated Device Enrollment. If the device doesn’t meet the minimum version expected by the MDM solution, the user is guided through an update before they can complete Setup Assistant. The same process happens automatically when used with Auto Advance. This helps ensure that devices owned by an organization are on the necessary operating system version before being put into production.

Managing the availability of software updates

Organizations may want to control which versions their users can update their devices to. For example, this control can be used to perform verification of an update with a test group before allowing all users to install it in production, or deploy different deferrals to different groups to phase the rollout of recent updates.

Time-based deferrals

Supervised devices can be prevented from offering OTA software updates to users until a specified period of time has passed since they were made publicly available by Apple. For example, say that an iPhone fleet is using iOS 17.3, and a deferred software update configuration of 30 days is applied. In this scenario, users have iOS 17.4 offered to them on their managed devices 30 days after the iOS 17.4 release date.

As part of the configuration, organizations can specify a custom deferral period from 1 to 90 days. In iOS and iPadOS, this delay applies to both operating system updates and upgrades. In macOS, they can also specify different deferral periods for operating system updates, upgrades, and non-operating system updates. Non-operating system updates include updates for Safari, printer drivers, and Xcode command-line tools. For example, a custom deferral can be used on macOS to defer a software upgrade for longer than a software update.

Note: OTA software updates are typically available for up to 180 days after their initial release date to ensure that an update is always available for managed devices with the maximum deferral value.

Recommended cadence on iOS and iPadOS

In addition to defining time-based deferrals, organizations can define whether users on supervised iPhone and iPad devices have the option to upgrade to a newer, major version or continue on the current one and still receive minor updates, even after an upgrade is available.

For example, on an iPhone using iOS 16.6.1, one of three options can be used:

- Offer users only additional updates to iOS 16.
- Offer users only the upgrade to iOS 17.
- Allow users the choice: additional updates to iOS 16 (for example, iOS 16.7) or upgrade to iOS 17.

The first option is available for only a limited period of time and allows users to benefit from any important security updates while testing is completed to approve the major upgrade for a production environment.

In conjunction with deferrals, a recommended cadence can be used. In the example above, it could be used to keep devices on iOS 16 while deferring software updates only (like iOS 16.7) for the defined period of time.

Installing software updates automatically

In iOS 18, iPadOS 18, and macOS 15, organizations can manage the automatic software update behavior on supervised devices.

Automatic software updates (not upgrades)

For *downloading and preparing* automatic software updates, the following configuration choices are available:

- Let the user choose whether to turn on automatic downloads.
- Automatic update downloads are turned off.
- Automatic update downloads are turned on.

For *installing* automatic software updates and upgrades, the following configuration choices are available:

- Let the user choose whether to turn on automatic update installation.
- Automatic update installations are turned off.
- Automatic update installations are turned on.

  Note: This option requires automatic downloads to be turned on.

These choices provide granular controls for an organization to define the most suitable automatic software update approach.

In macOS, there's an additional setting with the same three configuration options for security updates, which include updates for XProtect, Gatekeeper, and system data files. For more information, see the Apple Support article About background updates in macOS.

Automatic Rapid Security Responses

Rapid Security Responses don't adhere to the managed software update deferral. Because they apply only to the latest minor operating system version, if that update is deferred, the Rapid Security Response is also effectively deferred.

Organizations can define whether Rapid Security Responses are automatically applied and specify whether users are allowed to remove them after they've been applied.

Requiring authorization by an administrator on macOS

Organizations can change the default behavior to require authorization by a local administrator to install a software update. For example, this can be used in deployments where a device is shared with multiple users to restrict who can perform an update.

Enforcing software updates

Organizations can enforce specific software updates at a chosen time regardless of configured deferrals or if the automatic installation of Rapid Security Responses is turned off. This helps ensure managed devices run a specified version by a certain date and time while allowing users to install the update at a time convenient for them (prior to the enforcement date).

The enforcement date and time are relative to the local time zone of the device. This allows the same configuration to be applied across different regions. For example, if the enforcement date is set to 6 p.m., devices attempt to perform the update at 6 p.m. on the specified date in their local time zone.

When a software update is declared, users are informed about its deadline:

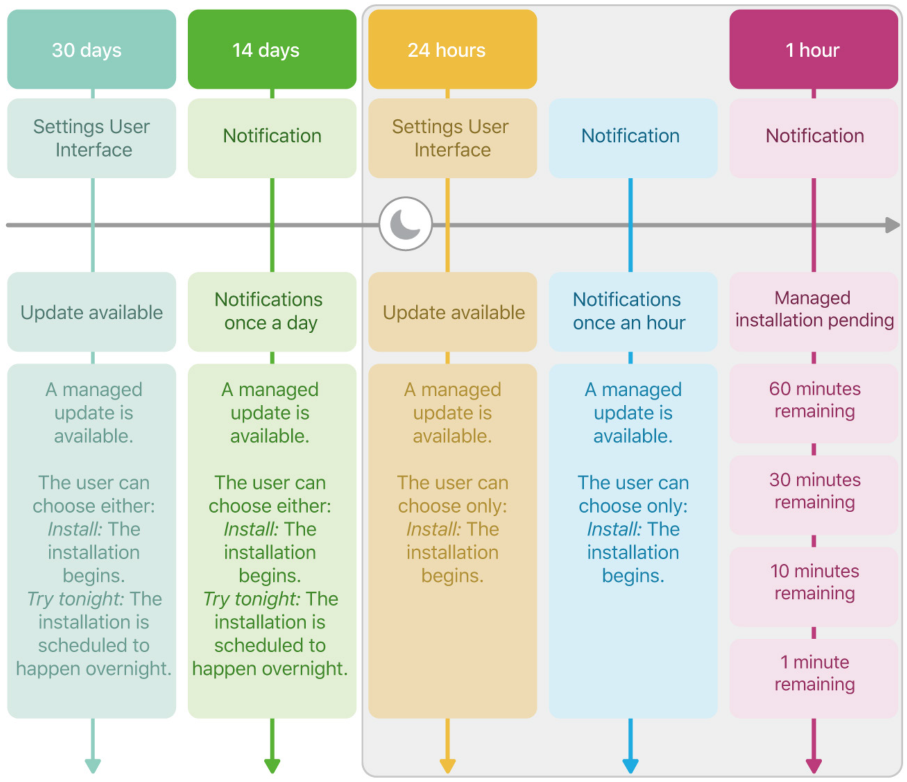
- In Settings (iOS and iPadOS) and in System Settings (macOS)
- In a notification

Depending on the time left until the enforcement deadline, the notification provides different options (as described below). To allow for increased transparency to users and to keep them informed about the process, additional information about the update can be provided using the “More information” link.

If the user doesn’t immediately begin the installation, the notifications and options shown depend on the remaining time get more frequent leading up to the enforcement date and are meant to encourage the user to install the update at a time convenient for them. To help ensure these notifications are displayed to the user, the Do Not Disturb feature is ignored during the 24 hours prior to the enforcement.

Alternatively, in iOS 18, iPadOS 18, and macOS 15, organizations can set the notifications to be displayed only 1 hour before the enforcement deadline and to show the restart countdown. This reduces the amount of notifications shown on devices—for example, if they aren’t directly assigned to a user (like a kiosk deployment).

To initiate and authorize the update from a notification or from Settings and System Settings, the system prompts the user for their passcode or password.


30 days
14 days
24 hours
1 hour
Settings User Interface
Notification
Settings User Interface
Notification
Notification
Update available
Notifications once a day
Update available
Notifications once an hour
Managed installation pending
A managed update is available.
The user can choose either:
Install: The installation begins.
Try tonight: The installation is scheduled to happen overnight.
A managed update is available.
The user can choose either:
Install: The installation begins.
Try tonight: The installation is scheduled to happen overnight.
A managed update is available.
The user can choose only:
Install: The installation begins.
A managed update is available.
The user can choose only:
Install: The installation begins.
60 minutes remaining
30 minutes remaining
10 minutes remaining
1 minute remaining

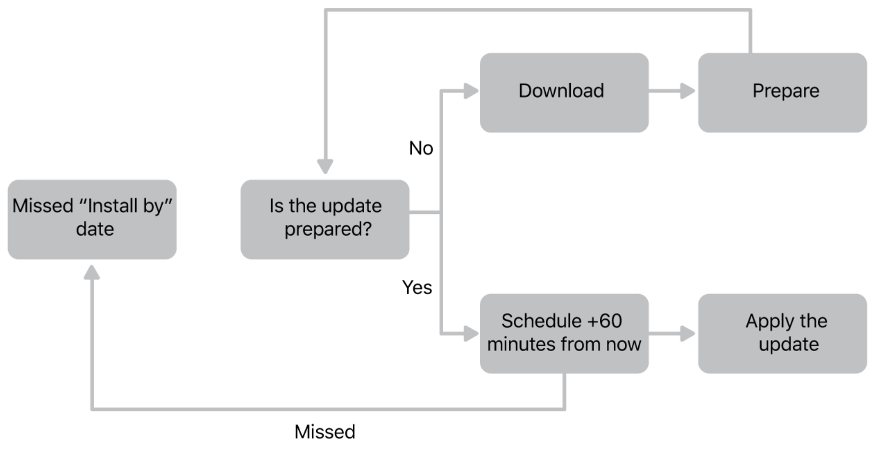
In case the user hasn't installed the update before the local enforcement date:

- iOS and iPadOS force the user to enter their passcode if one is set (unless it was entered earlier)
- macOS force quits all open apps (regardless of whether any documents are open and unsaved) and performs a restart if necessary
- On a Mac with Apple silicon, the Mac uses a bootstrap token (if one is available) to authorize the update or the Mac prompts the user for their credentials

To enforce the installation of an update, upgrade, or Rapid Security Response, a device must meet the same requirements as for a user-initiated update of the same type.

A key benefit of declarative device management is the autonomy of devices. Instead of triggering individual actions, the MDM solution declares the desired state and delegates the task to achieve that state to the device itself. A specific example of this behavior is a situation where the software update enforcement date was missed because the device didn't meet the requirements. The device automatically detects that the declared state hasn't yet been achieved and resumes the process when it's connected to the internet.

To do so—if necessary—the operating system downloads and prepares the update and posts another notification letting the user know the install is past due and an attempt will be made to perform the installation within the next hour. In case the process gets interrupted again for any reason, the process repeats the next time the device is powered on and connected to the internet.



Using status reports available with declarative device management, MDM solutions can also get increased transparency about the status of the installation—for example, waiting for, downloading and preparing, or installing the update. Meaningful error codes have been added in case the release couldn't be applied or was unable to be successfully completed. Some examples of this are if the device was offline, if the battery charge was too low, or if not enough free space was available.

Updating iPadOS on Shared iPad

Software updates on Shared iPad devices can be initiated over the air using the MDM solution that the Shared iPad is enrolled in. If the device is physically connected, the Finder or Apple Configurator on a Mac can also be used.

To install an update on Shared iPad, users must be signed out but can be left cached on the device. If the installation requires more free space than what is currently available, cached user account data must be removed. Due to the autonomy of declarative device management, a device keeps retrying to install a new release until it's successful.

To minimize downtime, updates to a Shared iPad should be scheduled to occur during off hours to minimize impact to users and the network.

Software update implementation details for MDM developers

By default, users can configure software update settings on their devices. Organizations can also manage software update settings and behavior using an MDM solution.

Declarative device management is the future of Apple device management. It allows the device to asynchronously apply settings and report status back to the MDM solution without constant polling. This is ideal for performance and scalability and also enables a modern approach to managing software updates. Declarative device management provides proactive status reporting from devices as values and configurations change. In this way, an MDM solution always has an up-to-date view on devices without having to perform regular queries.

Instead of sending a software update command to a device to initiate an update, an MDM solution declares the desired operating system version state and delegates the task to achieving that state to the device itself. This allows for a more resilient managed software update process and increased user transparency.

The importance of leveraging software update declarations

MDM solutions should leverage software update declarations whenever possible. However, the legacy software update commands and profiles are still available and supported. They can work alongside software update declarations with the following changes:

- Deferrals defined by a declaration take precedence over deferrals configured by a restriction.
- Automatic software update settings on macOS applied by a declaration take precedence over automatic update settings provided in a configuration profile.
- When there's a pending software update configured using declarative device management, some MDM commands are no longer processed by the client, returning errors communicating that there's an active declaration on the device, as shown in the following table:

MDM command	**Result**
`AvailableOSUpdates`	*Limited:* In macOS, operating system-based updates that aren't managed may show up in the command response—for example, in Xcode or command-line tools.
`ScheduleOSUpdate`	The device returns an `InstallFailed` error.
`OSUpdateStatus`	The device returns an empty status array.

Using the Apple Software Lookup Service

The Apple Software Lookup Service (available at https://gdmf.apple.com/v2/pmv) is the official resource to obtain a list of publicly available updates, upgrades, and Rapid Security Responses. It allows an MDM solution to query releases as soon as they are published and calculate applicability for each hardware model in a timely and accurate manner.

The service returns a JSON response containing three lists of available software releases:

- *PublicAssetSets:* This list contains the latest releases available to the general public if they try to update or upgrade.
- *AssetSets:* This list is a subset of PublicAssetSets and contains all the releases available for MDM solutions to push to supervised devices.
- *PublicRapidSecurityResponses:* This list contains Rapid Security Response releases currently available for Apple devices.

```
{
  "AssetSets": {
   "iOS": [
    {
"ProductVersion": "17.5",
"Build": "21F6079",
"PostingDate": "2024-05-13",
"ExpirationDate": "2024-08-15",
"SupportedDevices": ["iPad11,1", "iPad11,2", "iPad11,3", "iPad11,4",
"iPad11,6", "iPad11,7", "iPad12,1", "iPad12,2", "iPad13,1","iPad13,10",
"iPad13,11", "iPad13,16", "iPad13,17", “iPad13,18", "iPad13,19", "iPad13,2",
"iPad13,4", “iPad13,5", "iPad13,6", "iPad13,7", "iPad13,8", "iPad13,9",
"iPad14,1", "iPad14,2", "iPad14,3", "iPad14,4", "iPad14,5", "iPad14,6",
"iPad6,11", "iPad6,12", "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8",
"iPad7,1", "iPad7,11", "iPad7,12", "iPad7,2", "iPad7,3", "iPad7,4",
"iPad7,5", "iPad7,6", "iPad8,1", "iPad8,10", "iPad8,11", "iPad8,12",
"iPad8,2", "iPad8,3", "iPad8,4", "iPad8,5", "iPad8,6", "iPad8,7", "iPad8,8",
"iPad8,9", "iPhone10,1", "iPhone10,2", "iPhone10,3", "iPhone10,4",
"iPhone10,5", "iPhone10,6", "iPhone11,2", "iPhone11,6", "iPhone11,8",
"iPhone12,1", "iPhone12,3", "iPhone12,5", "iPhone12,8", "iPhone13,1",
"iPhone13,2", "iPhone13,3", "iPhone13,4", "iPhone14,2", "iPhone14,3",
"iPhone14,4", "iPhone14,5", "iPhone14,6", "iPhone14,7", "iPhone14,8",
"iPhone15,2", "iPhone15,3"
   ]
 },
```

Each element in the list contains the `ProductVersion` number and `Build` of the operating system, the `PostingDate` when the release was published, the `ExpirationDate`, and a list of `SupportedDevices` for that release. The device list matches the `ProductName` value from the device, which is returned in a `DeviceInformation` response, the initial `Authenticate` request, or in the `MachineInfo` when the device tries to enroll.

The expiration date, typically set to 180 days after the posting date, defines the date the signing of the update expires. An expired update can't be installed on devices anymore. When subsequent updates are made available, previous updates might have their expiration dates updated. If an expiration date isn't provided, the update has yet to expire. An update has expired only when it has an expiration date in the past.

The assets are grouped by operating system platform using the following keys:

- `iOS` (which includes iPadOS, tvOS, and watchOS)
- `macOS`
- `xrOS` (which is visionOS)

Use the product version list to determine which versions are greater than the device's current operating system version and are applicable to a specific device. Provide that list of versions to the MDM administrator as potential operating system update candidates.

Sending a status report to the MDM solution

To receive updates for status items as they change, the server must subscribe to each status report by sending a `ManagementStatusSubscriptions` declaration to the device. The device then sends a `StatusReport` to the MDM solution when a `ManagementStatusSubscriptions` declaration becomes active, if the status of a subscribed item changes, and every 24 hours.

For the purposes of monitoring operating system versions and software update status, the MDM solution may want to subscribe to the following status reports:

Status report	**Description**
`device.operating-system.build-version`	The operating system's build version on the device (for example, 21E219).
`device.operating-system.version`	The operating system's version in use on the device (for example, 17.4).
`device.operating-system.supplemental.build-version`	The operating system's build and Rapid Security Response versions in use on the device, for example (20A123a or 20F75c).
`device.operating-system.supplemental.extra-version`	The operating system's Rapid Security Response version in use on the device (for example, a).
`softwareupdate.pending-version`	A dictionary that contains the build and operating system versions of the software update that's pending on the device.
`softwareupdate.install-state`	The software update installation status, which has the following values: • *none:* There's no software update pending, and any previous software update succeeded. • *waiting:* A software update is waiting to start. • *downloading:* The system is downloading a software update. • *prepared:* The system prepared the software update and it's ready for installation. • *installing:* The system is installing the software update. • *failed:* The software update failed.

Status report	Description
`softwareupdate.install-reason`	A dictionary with details about the reason for a pending software update. The `softwareupdate-reason` key has one of the following values: • *system-settings:* It was triggered from Settings (on iOS and iPadOS) or System Settings (on macOS). • *install-tonight:* It was triggered by an install tonight action. • *auto-update:* It was triggered by an automatic update. • *notification:* It was triggered from a user notification. • *setup-assistant:* It was triggered during Setup Assistant. • *command-line:* It was triggered by the `softwareupdate` command-line tool. • *mdm:* It was triggered by an MDM command. • *declaration:* It was triggered by a declarative device management configuration.
`softwareupdate.failure-reason`	Details about a software update failure. The details include the number of times the software update has failed, last failure timestamp, and the failure reason.
`softwareupdate.beta-enrollment`	The device's enrolled beta program name, or an empty string if there is no enrolled beta program.

In addition to the other reports, MDM solutions may also want to make `softwareupdate.install-reason` available to administrators for support purposes and to provide additional insight into how an update got triggered. This dictionary can be used to determine whether a user has initiated the update themselves, the update happened automatically, or it got enforced by a software update enforcement declaration.

Requesting a specific minimum software version during MDM enrollment

If a device supports this capability, it returns an MDM_CAN_REQUEST_SOFTWARE_UPDATE key, set to True, in the `MachineInfo` data that it sends in the initial HTTP POST request to the MDM solution when the device detects a management configuration in Setup Assistant. For more information, see the MachineInfo yaml file in the Apple device management GitHub repository.

In addition, devices provide the following fields in the `MachineInfo` data (all strings) :

Key	**Minimum supported operating system**	**Description**
VERSION	iOS 17 iPadOS 17 macOS 14	The build version installed on the device (for example, 7A182).
OS_VERSION	iOS 17 iPadOS 17 macOS 14	The operating system version installed on the device (for example, 17.0).
SUPPLEMENTAL_BUILD_VERSION	iOS 17 iPadOS 17 macOS 14	The device's Rapid Security Response version (if one is available).
SUPPLEMENTAL_OS_VERSION_EXTRA	iOS 17 iPadOS 17 macOS 14	The device's Rapid Security Response version extra (if one is available).
SOFTWARE_UPDATE_DEVICE_ID	iOS 17.4 iPadOS 17.4 macOS 14.4	The device model identifier used to lookup available operating system updates in the Apple Software Lookup Service.

Based on the information provided, the MDM solution can decide whether to enforce the device to update.

- If an MDM solution chooses *not* to enforce a software update, it simply returns the MDM enrollment profile in response to the HTTP POST request, as it would normally do to allow an MDM enrollment to proceed.
- If the MDM solution chooses to enforce a software update then it must return an HTTP response with the 403 status code, and include a JSON or XML object in the response body (the HTTP Content-Type response header must be set to `application/json` or `application/xml` respectively).

After receiving this error response, the device attempts to update to the specified version. If the update succeeds, the device restarts and the user must go through Setup Assistant again. The next `MachineInfo` POST request from the device to the MDM solution shows the updated operating system version, and the MDM solution can then proceed with MDM enrollment. If the update fails, an error is shown to the user and the Remote Management pane appears in Setup Assistant again.

The `response` schema is defined in the table below.

Key	**Type**	**Required**	**Description**
`code`	String	Yes	Must be set to `com.apple.softwareupdate.required`.
`description`	String	No	The description of the error. Used only for logging purposes.
`message`	String	No	The description of the error suitable for displaying to the user.
`details`	Dictionary	Yes	Additional data specifying the software update.

The `details` dictionary schema is defined here.

Key	**Type**	**Required**	**Description**
`OSVersion`	String	Yes	The operating system version that the device is required to update to.
`BuildVersion`	String	No	The build version that the device is required to update to.
`RequireBetaProgram`	Dictionary	No	The device enrolls in the beta program, allowing enforced software updates to beta program operating system versions. The device remains in the beta program after the enforced software update is complete.

If only the `OSVersion` is specified, a device automatically downloads and installs any Rapid Security Responses available for this version. In case a specific build or supplemental version is needed, an MDM solution can also optionally specify the `BuildVersion`. For example, to require a device to run iOS 16.5.1(a) before enrolling—although iOS 16.5.1(c) is already available—an MDM solution must set `OSVersion` to iOS 16.5.1 and `BuildVersion` to 20F770750b.

Important: Before macOS 15, only releases from the `PublicAssetSets` and `PublicRapidSecurityResponses` lists can be specified. In macOS 15, assets from `AssetSets` can also be used.

MDM settings for software updates

The `com.apple.configuration.softwareupdate.settings` declaration (available in iOS 18, iPadOS 18, and macOS 15) consists of dictionaries that can be used to configure various aspects of the software update behavior.

After an MDM solution distributes different keys across multiple declarations, a device merges the settings of all active software update settings declarations. In case the same key is configured by multiple declarations, the merge behavior depends on the individual key and is outlined in the tables below.

For the most current schema specification, see the Apple device management GitHub repository.

Configuring automatic software updates with MDM

The `com.apple.configuration.softwareupdate.settings` declaration offers a dictionary to define the automatic software update behavior on supervised iPhone, iPad, and Mac devices. The `AutomaticActions` dictionary offers the keys shown below (default is `Allowed` and not required).

Key	Type	Merge behavior	Description
`Download`	Enum	The last value from the list: Allowed, AlwaysOn, AlwaysOff.	Specifies whether automatic downloads and preparation of available updates only (not upgrades and Rapid Security Responses) can be controlled by the user: • *Allowed:* The user can turn on or turn off automatic downloads. • *AlwaysOn:* Automatic downloads are always turned on. • *AlwaysOff:* Automatic downloads are always turned off.
`InstallOSUpdates`	Enum	The last value from the list: Allowed, AlwaysOn, AlwaysOff.	Specifies whether automatic installation of available operating system updates only (not upgrades and Rapid Security Responses) can be controlled by the user: • *Allowed:* The user can turn on or turn off automatic installations. • *AlwaysOn:* Automatic installations are always turned on. • *AlwaysOff:* Automatic installations are always turned off.
`InstallSecurityUpdates` (Available for macOS only)	Enum	The last value from the list: Allowed, AlwaysOn, AlwaysOff.	Specifies whether automatic installation of available security updates can be controlled by the user: • *Allowed:* The user can turn on or turn off automatic installations. • *AlwaysOn:* Automatic installations are always turned on. • *AlwaysOff:* Automatic installations are always turned off.

In case multiple declarations include a value for the same key, the last value in the following list applied by any of those declarations takes precedence: `Allowed`, `AlwaysOn`, `AlwaysOff`.

How MDM handles Rapid Security Responses

Rapid Security Responses always apply to the latest update of an operating system, which becomes the base version of the Rapid Security Response. For example, if an iPhone has operating system version iOS 17.2 installed, then it applies the 17.2 (a) supplemental update, if one is available. In iOS 18, iPadOS 18, and macOS 15, combined updates have been made available, which allow a software update to include any available Rapid Security Responses.

Prior to iOS 18, iPadOS 18, and macOS 15, an MDM solution may have to trigger two software updates to ensure that a specific supplemental version is present: first, it must update the device to the base version of the supplemental update, if the device isn't already on that base version (for example, iOS 17.1 to iOS 17.2); then it must update the base version to the supplemental version (for example, iOS 17.2 to iOS 17.2 (a)).

In iOS 18, iPadOS 18, and macOS 15, an MDM solution can specify either:

- The operating system version (which automatically installs available Rapid Security Responses)
- The supplemental build version (which causes the device to perform a necessary update to the base version automatically as part of the process)

These two approaches apply to the software update enforcement configuration, and to the enforced minimum version during Automated Device Enrollment.

The `com.apple.configuration.softwareupdate.settings` declaration can also be used to configure the Rapid Security Response behavior on supervised iPhone, iPad, and Mac devices. The `RapidSecurityResponse` dictionary contains the keys shown below (default is True and not required).

Key	**Type**	**Merge behavior**	**Description**
`Enable`	Boolean	Logical AND operation of the values	If false, Rapid Security Responses aren't offered for user installation. This defines whether Rapid Security Responses are automatically installed on user's devices.
`EnableRollback`	Boolean	Logical AND operation of the values	If false, Rapid Security Response rollbacks aren't offered to the user. This controls whether users have the option to remove a Rapid Security Response.

Independent of the `Enable` key, Rapid Security Responses can still be installed with the `com.apple.configuration.softwareupdate.enforcement.specific` declaration.

Deferring a software update with MDM

Deferring a software update or upgrade from 1 to 90 days is done using the `com.apple.configuration.softwareupdate.settings` declaration on supervised iPhone, iPad, and Mac devices.

A configured deferral defines how many days a release isn't offered to users after it became publicly available. Independent of a configured deferral, an MDM solution can still enforce a specific software update, upgrade, or Rapid Security Response on managed devices.

Note: Deferring software updates also defers any Rapid Security Responses that are dependent on that version.

The Deferrals dictionaries offer different keys to configure the behavior depending on the platform (no defaults, not required).

Deferring a software update on iOS and iPadOS

Key	Type	Merge behavior	Description
CombinedPeriodInDays	Integer 1–90	Maximum number of days	Specifies the number of days to defer a software update. When set, software updates and upgrades appear only after the specified delay, following the release of the software update or upgrade.
RecommendedCadence	Enum	The last value from the list: All, Oldest, Newest	Specifies how the device shows software upgrades to the user. When a software update and upgrade is available, the device behaves as follows: • *All:* Shows all software updates and upgrades. • *Oldest:* Shows only updates for the oldest (lower numbered) software version. • *Newest:* Shows only a software upgrade to the newest (highest numbered) software version.

Both `CombinedPeriodInDays` and `RecommendedCadence` can be used in combination. For example, if `RecommendedCadence` is set to `Oldest` and `CombinedPeriodInDays` is set to 30, a user sees only software updates for the oldest release after 30 days of their publishing date.

Deferring a software update on macOS

Key	Type	Merge behavior	Description
`MajorPeriodInDays`	Integer 1–90	Maximum number of days	Specifies the number of days to defer a software upgrade on the device. When set, software upgrades appear only after the specified delay, following the release of the software upgrade.
`MinorPeriodInDays`	Integer 1–90	Maximum number of days	Specifies the number of days to defer a software update only (not a software upgrade or Rapid Security Response) on the device. When set, software updates appear only after the specified delay, following the release of the software update.
`SystemPeriodInDays`	Integer 1–90	Maximum number of days	Specifies the number of days to defer non-operating system updates. When set, updates appear only after the specified delay, following the release of the update.

In macOS, an additional key is available to determine whether both standard users and local administrators can perform an update or upgrade (the default behavior), or determine whether administrative permissions are required (default is True and not required).

Key	Type	Merge behavior	Description
`AllowStandardUserOSUpdates`	Boolean	Logical AND operation of the values	If true, a standard user can perform updates and upgrades. If false, only administrators can perform updates and upgrades.

Enforcing software updates with MDM

To enforce a software update by a certain time on devices enrolled using Device Enrollment or Automated Device Enrollment, MDM solutions can apply the `com.apple.configuration.softwareupdate.enforcement.specific` declaration. The declaration offers the keys shown below (all strings and no defaults).

Key	Required	Description
`TargetOSVersion`	Yes	The target operating system version to update the device to by the appropriate time. This is the operating system version number, for example, iOS 17.4.
`TargetBuildVersion`	No	The target build version to update the device by the specified time, for example, 21E219. The system uses the build version for testing during seeding periods. The build version can include a supplemental version identifier, for example, 21E219a. If the build version isn't consistent with the target operating system version specified in the `TargetOSVersion` key, the `TargetOSVersion` takes precedence.
`TargetLocalDateTime`	No	The local date time value that specifies when to force install the software update. Use the format YYYY-MM-DDTHH:MM:SS, which is derived from RFC3339 but doesn't include a time zone offset. If the user doesn't trigger the software update before this time, the device force installs it.
`DetailsURL`	No	The URL of a web page that shows details that the organization provides about the enforced release.

If a configuration specifies an operating system or build version that's the same as, or older than the current device version, then the configuration is ignored.

If multiple configurations are present with a newer operating system or build version than the current device version, the configuration with the earliest target date and time is processed first, and any others remain in the queue. When the device updates to a new version, the set of configurations are reprocessed to determine which becomes the next one to be processed.

Any available Rapid Security Responses are automatically installed if an MDM solution defines only the `TargetOSVersion`. To target a specific release or Rapid Security Response, an MDM solution can use the `TargetBuildVersion` key in addition to specifying the build, including the supplemental version identifier.

The Notifications key in the `com.apple.configuration.softwareupdate.settings` declaration, changes the default notification behavior to show only a notification 1 hour before the enforcement time, and the restart countdown (default is True and not required).

Key	Type	Merge behavior	Description
`Notifications`	Boolean	Logical AND operation of the values	If true, the device shows all software update enforcement notifications. If false, the device only shows notifications triggered one hour before the enforcement deadline, and the restart countdown notification.

Using the bootstrap token for Mac computers with Apple silicon

To authorize an enforced software update on a supervised Mac computer with Apple silicon, an MDM solution can request and escrow a bootstrap token. This allows for a completely seamless software update experience and avoids the need for user interaction as part of the process. When needed, the device uses a `GetBootstrapTokenRequest` to retrieve the bootstrap token from the MDM solution.

In the first step, the MDM solution determines whether the device supports a bootstrap token using the `SecurityInfo` command. If the response includes a `BootstrapTokenRequiredForSoftwareUpdate` that's set to true, the device can use a bootstrap token to authorize a software update.

To get a bootstrap token created, the MDM solution must add `com.apple.mdm.bootstraptoken` to the `ServerCapabilities` array in the MDM profile. For more information, see the MDM payload on the Apple Developer website.

If the MDM enrollment profile declares support for using a bootstrap token and the device it's installed on supports using one, then the device creates a bootstrap token when a Secure Token-enabled user logs in. It then reaches out to the check-in endpoint of the MDM solution and escrows the token using a `SetBootstrapTokenRequest`. For more information, see Set Bootstrap Token on the Apple Developer website.

Phases of Apple software update enforcement

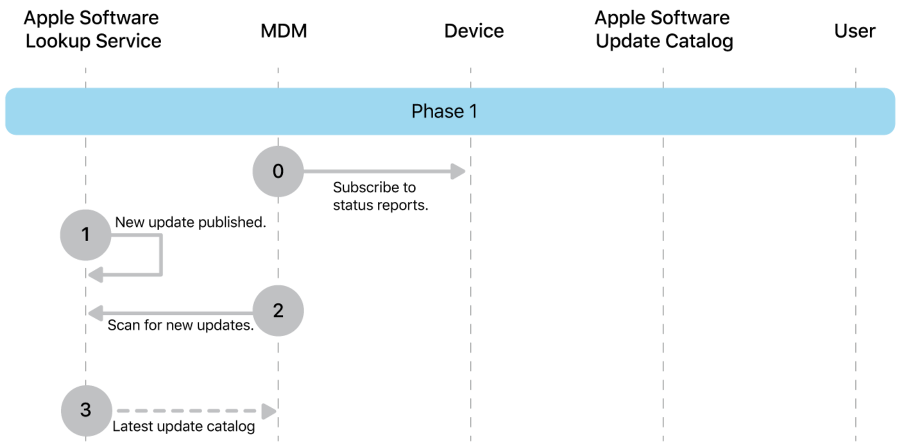
Enforcing software updates using MDM involves five main components:

- The MDM solution
- The device
- The user
- The Apple Software Lookup Service
- The Apple Software Update Catalog

Phase 1

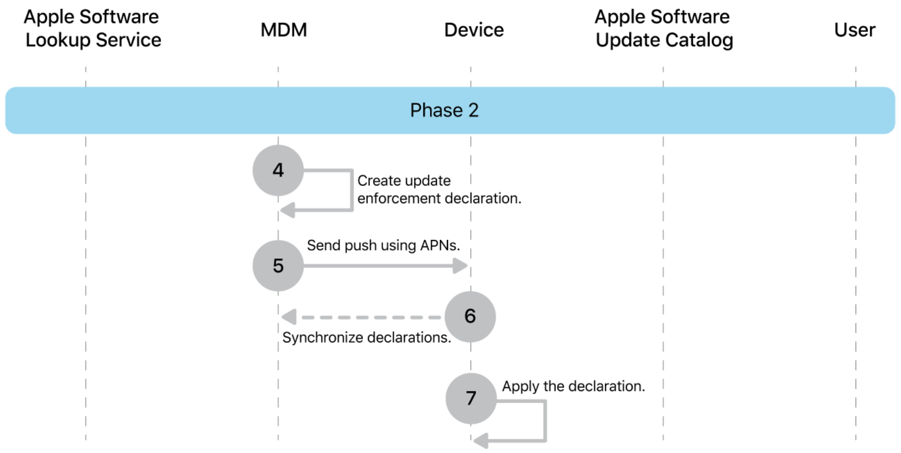
The MDM solution regularly monitors the Apple Software Lookup Service for new releases. If one is detected, it uses the `SupportedDevices` key of the catalog and compares it to its list of managed devices to determine which devices the release is applicable to.

The MDM solution should also subscribe to the `softwareupdate.*` and `device.operating-system.*` status reports to automatically retrieve updates if any of those values change.



Phase 2

The MDM solution creates a `com.apple.configuration.softwareupdate.enforcement.specific` declaration with the detected version and defines the `TargetLocalDateTime` and optionally the `DetailsURL` according to the organizational requirements. The MDM solution then sends a push notification to the device to trigger the synchronization of declarations. For more information, see Integrating Declarative Management on the Apple Developer website.


Apple Software Lookup Service
MDM
Device
Apple Software Update Catalog
User
Phase 2
4
Create update enforcement declaration.
Send push using APNs.
5
6
Synchronize declarations.
Apply the declaration.
7

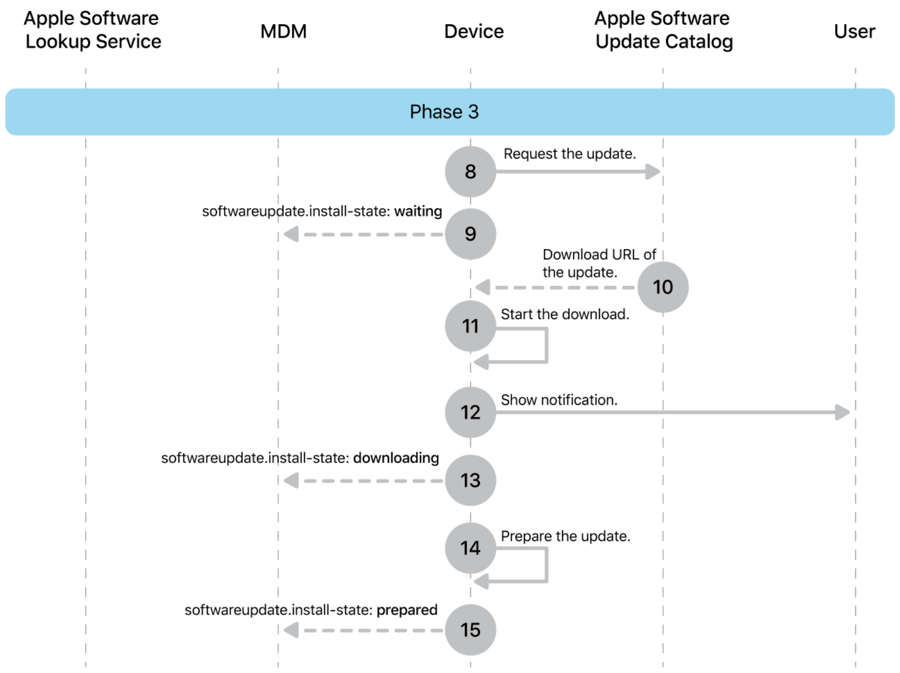
Phase 3

After the declaration becomes active on a device, it reaches out to the Apple Software Update Catalog to retrieve the download URL and then begins to download the update if the requirements are met. The device then presents a notification to the user and returns the following to the MDM solution:

1. A `softwareupdate.install-state` value of waiting, which indicates the process to request the update has started.
2. A `softwareupdate.install-state` value of downloading, which indicates the update is being downloaded by the device.

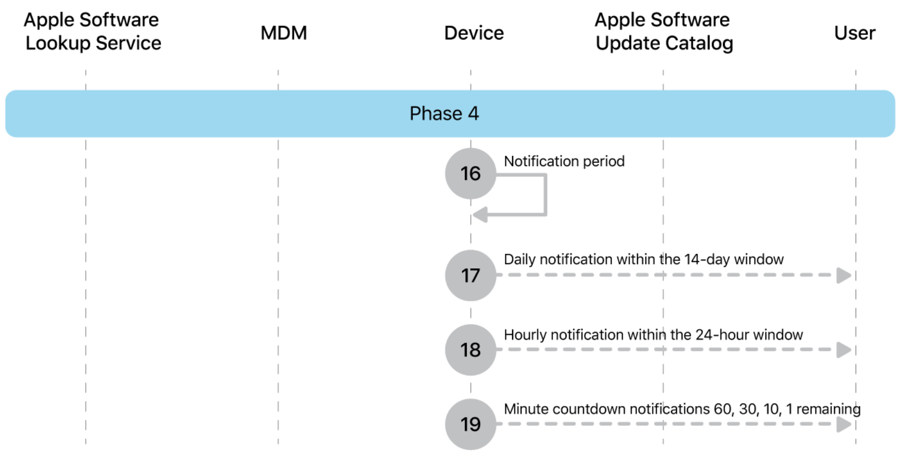
If a content caching service is available to the device, it attempts to download the software update from the content cache.

After the device successfully downloaded the update, it prepares the software update for installation. After this process completes, a `softwareupdate.install-state` value of prepared is sent back to the MDM solution.


Apple Software Lookup Service
MDM
Device
Apple Software Update Catalog
User
Phase 3
8
Request the update.
softwareupdate.install-state: waiting
9
Download URL of the update.
10
11
Start the download.
12
Show notification.
softwareupdate.install-state: downloading
13
14
Prepare the update.
softwareupdate.install-state: prepared
15

Phase 4

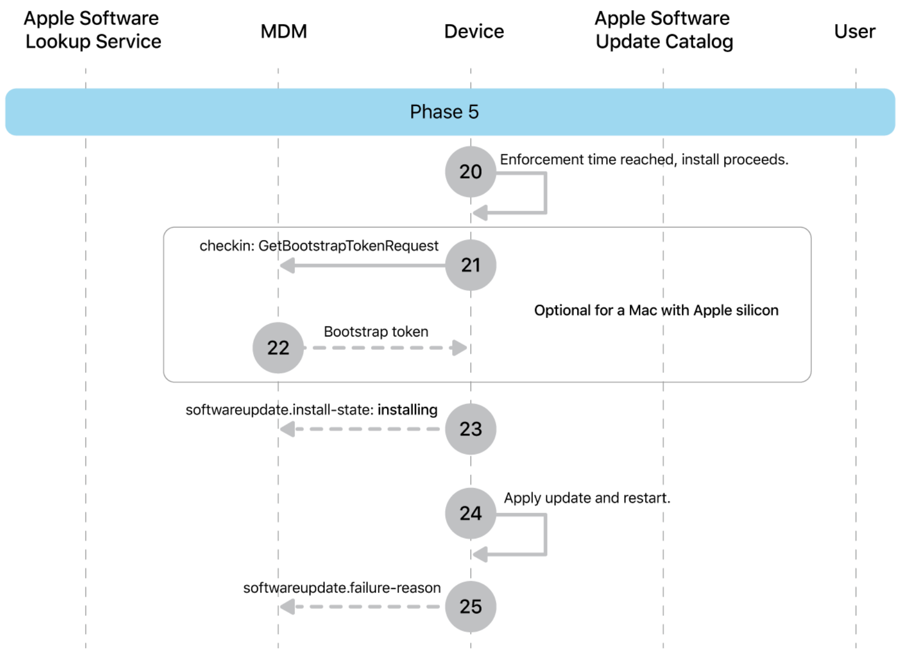
The device enters the notification period. Depending on when the installation is to occur, this notification may display different text and options.


Apple Software Lookup Service
MDM
Device
Apple Software Update Catalog
User
Phase 4
16 Notification period
17 Daily notification within the 14-day window
18 Hourly notification within the 24-hour window
19 Minute countdown notifications 60, 30, 10, 1 remaining

Phase 5

In case the user hasn't installed the update before the enforcement deadline, the device begins the installation and sends a `softwareupdate.install-state` value of installing back to the MDM solution. Before starting the installation, a Mac with Apple silicon contacts the MDM solution to retrieve the bootstrap token (if one is available).

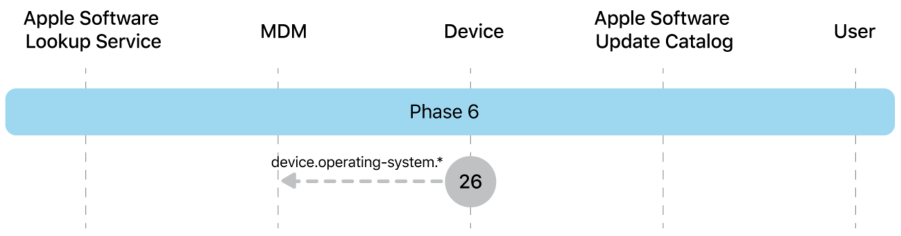
If the update succeeds, the device restarts. If the update fails, a `softwareupdate.install-state` value of `failed` is sent. In either case, the device sends back a `softwareupdate.failure-reason` status report. If the update was successful, the count key has a value of `0`.


Apple Software Lookup Service
MDM
Device
Apple Software Update Catalog
User
Phase 5
20 Enforcement time reached, install proceeds.
21 checkin: GetBootstrapTokenRequest
22 Bootstrap token
Optional for a Mac with Apple silicon
23 softwareupdate.install-state: installing
24 Apply update and restart.
25 softwareupdate.failure-reason

Phase 6

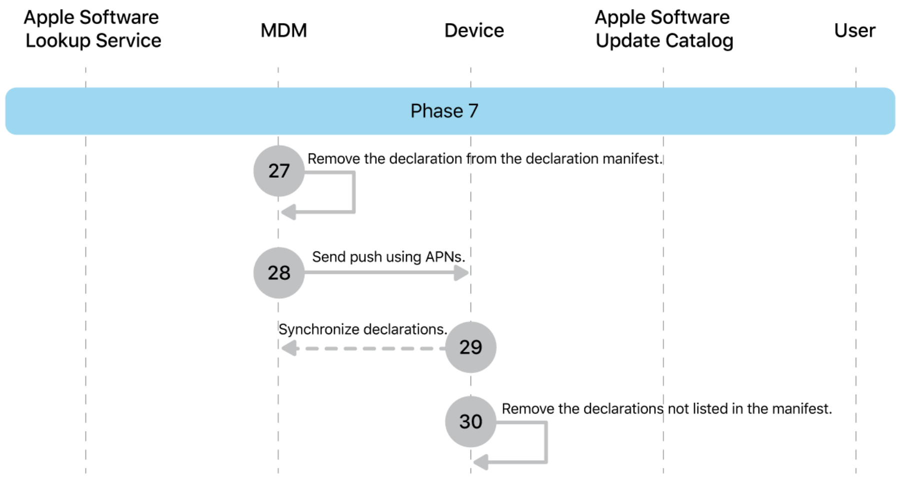
The device then sends the following information back to the MDM solution. Depending on the update, not all these objects have return values.

- *StatusDeviceOperatingSystemVersion:* A status report of the device's operating system version.
- *StatusDeviceOperatingSystemBuildVersion:* A status report of the device's software build identifier.
- *StatusDeviceOperatingSystemSupplementalBuildVersion:* A status report of the device's operating system version and Rapid Security Response build identifier.
- *StatusDeviceOperatingSystemSupplementalExtraVersion:* A status report of the device's operating system's Rapid Security Response identifier.


Apple Software Lookup Service
MDM
Device
Apple Software Update Catalog
User
Phase 6
device.operating-system.*
26

Phase 7

The MDM solution unassigns the declaration from the device and sends a push notification to the device to initiate the synchronization. After synchronizing, the device removes the declaration.


Apple Software Lookup Service
MDM
Device
Apple Software Update Catalog
User
Phase 7
27
Remove the declaration from the declaration manifest.
28
Send push using APNs.
Synchronize declarations.
29
30
Remove the declarations not listed in the manifest.

Testing software updates with the AppleSeed for IT beta program

AppleSeed for IT is a program specifically designed for enterprise and education customers committed to testing each new version of Apple beta software in their organizations. This program provides IT professionals and technology managers with an opportunity to evaluate the latest prerelease software versions in their unique work environments, offer feedback directly to Apple engineering teams through a dedicated feedback submission process, and participate in detailed testing plans and forum discussions with other participants.

iOS 17.5, iPadOS 17.5, and macOS 14.5, or later, make it easier than ever to manage beta program participation in an organization. Users can be offered to enroll into beta programs even without an Apple Account in Settings or System Settings. MDM solutions can also automatically enroll devices during Setup Assistant when using Automated Device Enrollment or remotely at a later time if the device is supervised and runs iOS 18, iPadOS 18, macOS 15 or later. If necessary, an MDM solution has the option to remove a supervised device from beta programs and restrict a user from manually enrolling. This removes the need for manual steps performed by the user and allows for a streamlined process throughout the beta testing lifecycle.

To offer AppleSeed for IT beta versions without the need for an Apple Account, a user with the role of administrator in Apple School Manager or Apple Business Manager must sign in to the AppleSeed for IT portal and accept the terms and conditions on behalf of their organization for the current beta period.

Although beta enrollment can be managed without the need for an Apple Account, organizations may want to consider providing participating users a Managed Apple Account so they can submit feedback directly to Apple. This also ensures submitted feedback is tied to their organization. If users select to submit their feedback for the team rather than as personal feedback, other users like the IT team can engage in submitted tickets and stay informed. For more information on team feedback, see Manage team feedback in Feedback Assistant on Mac in the Feedback Assistant User Guide.

Similar to software updates and upgrades, beta releases provided by those programs can be deferred on supervised devices and a declarative status report provides increased visibility and allows organizations to track beta program enrollments on managed devices.

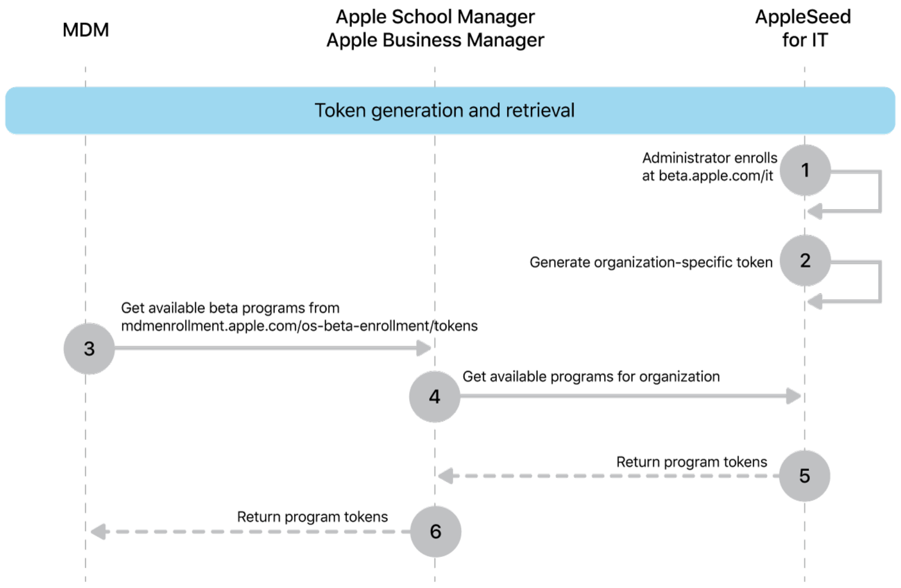
Using the available configuration options, an organization can remotely enroll different devices into different beta programs and—combined with the option to defer beta and production releases—can be used to implement a phased testing and rollout approach starting right with the first beta release.

Note: The beta configuration and status report isn't supported on devices using User Enrollment.

Enrolling a device in a beta program

To enroll a device in the Apple Beta Software Program or AppleSeed for IT, an MDM solution must retrieve a token from Apple and provide it to devices during Automated Device Enrollment or using the `com.apple.configuration.softwareupdate.settings` declaration.

The first step is for a user with the role of administrator in Apple School Manager or Apple Business Manager to enroll at https://beta.apple.com/it. After enrollment, an MDM solution can request available beta program tokens using the https://mdmenrollment.apple.com/os-beta-enrollment/tokens endpoint. Similar to other service endpoints available at mdmenrollment.apple.com, MDM solutions must authenticate using OAuth.


MDM
Apple School Manager
Apple Business Manager
AppleSeed for IT
Token generation and retrieval
Administrator enrolls at beta.apple.com/it
1
Generate organization-specific token
2
Get available beta programs from mdmenrollment.apple.com/os-beta-enrollment/tokens
3
Get available programs for organization
4
Return program tokens
5
Return program tokens
6

The HTTP GET request must include the following header fields (all required):

HTTP header field	Description
X-ADM-Auth-Session	The OAuth token to authenticate the request. For more information about the authentication process, see Authenticating with a Device Enrollment Program (DEP) Server on the Apple Developer website.
X-Server-Protocol-Version	Must be set to value: 1

The service endpoint returns a JSON object with the following structure:

```
{
  "seedBuildTokens": [
    {
      "token":
"p3ySHD3CiWtpsH1DKS8sVdv9BgmFbRDh31xJH2584wJ5AngrYoReFB4MVY53rucW",
      "title": "macOS AppleSeed Beta",
      "os": "OSX"
    },
    {
      "token":
"35b68K477rAsry6dxiDJBnE7AvjRTueUXFa9jZ3ZhQSFpJZ3Jxz9M8mCt9UXK4Sg",
      "title": "iOS 18 AppleSeed Beta",
      "os": "iOS"
   }
  ]
}
```

To enroll a device into a beta program, the `RequireBetaProgram` dictionary must contain the keys shown below (all required strings).

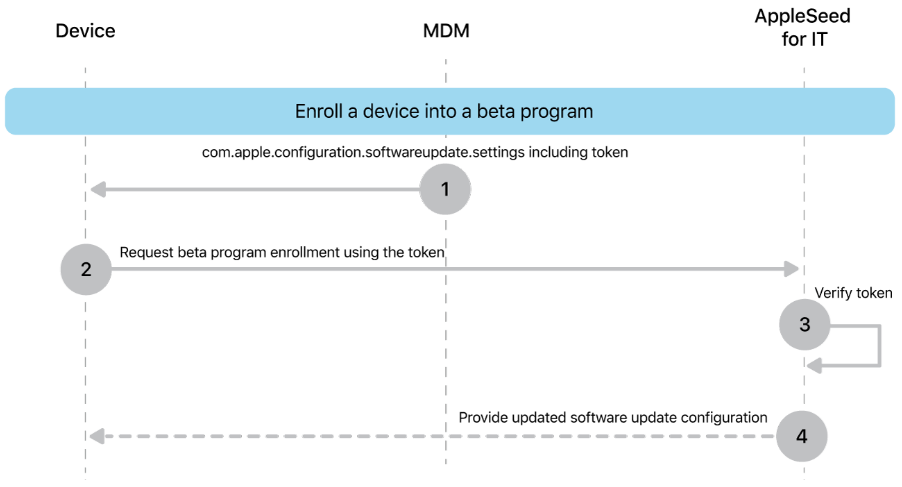
Key	Description
`Description`	A human-readable description of the beta program.
`Token`	The seeding service token for the organization that the MDM server is part of. This token is used to enroll the device in the corresponding beta program.

The following is an example response making use of the described keys:

```
{
    "code": "com.apple.softwareupdate.required",
    "description": "AppleSeed enrollment required",
    "message": "This device needs to be enrolled into the AppleSeed Beta
program",
    "details": {
        "OSVersion": "17.5",
        "RequireBetaProgram": {
            "code": "iOS 17 AppleSeed Beta",
            "token":
"35b68K477rAsry6dxiDJBnE7AvjRTueUXFa9jZ3ZhQSFpJZ3Jxz9M8mCt9UXK4Sg","
        }
    }
}
```

The token is unique for each organization and can’t be reused across different Apple School Manager and Apple Business Manager organizations. The token is also specific to a certain operating system upgrade seeding period. The title is a human-readable description of the beta release and os can contain the following values: `iOS` (includes iPadOS), `OSX` (macOS), `tvOS`, `watchOS`, or `xrOS` (visionOS).

After an iPhone or iPad is enrolled into device management, an MDM solution can enroll or unenroll supervised iPhone or iPad devices from beta programs using the Beta dictionary in the `com.apple.configuration.softwareupdate.settings` declaration.


Device
MDM
AppleSeed for IT
Enroll a device into a beta program
com.apple.configuration.softwareupdate.settings including token
1
2
Request beta program enrollment using the token
Verify token
3
Provide updated software update configuration
4

On unsupervised iPhone or iPad devices, only the `OfferPrograms` array can be used to allow users to manually enroll into beta programs the organization has subscribed to. The beta dictionary offers the following keys (not required):

Key	**Type**	**Default**	**Merge behavior**	**Description**
`ProgramEnrollment`	Enum	Allowed	The last value from the list: Allowed, AlwaysOn, AlwaysOff	Specifies whether beta program enrollment can be controlled by the user in the software update settings user interface: • *Allowed:* The user can enroll in any applicable beta programs associated with the Apple Account they used to sign in. If the `OfferPrograms` key is present, then the programs listed in that key are also presented to the user. • *AlwaysOn:* The beta programs specified by the organization are used, and the user isn't able to enroll in a beta program with the Apple Account they used to sign in. The device is automatically enrolled into the beta program specified by the `RequireProgram` key, if it's present. Otherwise, the programs listed in the `OfferPrograms` key are presented to the user to choose in which to enroll. • *AlwaysOff:* The device isn't allowed to enroll in any beta programs. The device is removed from any beta programs, if already enrolled.
`OfferPrograms`	Array	—	Unique union of all values	An array of beta programs allowed on the device. This key must only be present if the `ProgramEnrollment` key is set to `Allowed` or `AlwaysOn`. This key must not be present if the `RequireProgram` key is present. This key can be present on unsupervised devices where the `ProgramEnrollment` key isn't supported but is implicitly set to `Allowed`.
`RequireProgram`	Dictionary	—	First configuration applied	The device automatically enrolls in this beta program. This key must be present only if the `ProgramEnrollment` key is set to `AlwaysOn`. The `OfferPrograms` key must not be present if this key is present.

In addition to sending the name of the program, the `OfferPrograms` and `RequireProgram` options require that the token of the beta program be sent to the device. This token is used with Apple to verify eligibility and receive an updated software update configuration.

To allow users to enroll using their personal Apple Account or Managed Apple Account, an MDM solution can set the `ProgramEnrollment` key to `Allowed`. This allows users to enroll into any program available to their account and additionally into any beta program specified by the `OfferPrograms` array. Each Program dictionary in the `OfferPrograms` array must consist of the following keys (all strings, all required):

Key	**Description**
`Description`	A human-readable description of the beta program.
`Token`	The seeding service token that the MDM solution is part of for the organization. This token is used to enroll the device in the corresponding beta program.

If an organization wants to allow users to participate without the need to sign in, they can set the `ProgramEnrollment` key to `AlwaysOn`. In this case users are offered all programs listed in the `OfferPrograms` array. They can also automatically enroll devices into a beta program using a combination of `ProgramEnrollment` set to `AlwaysOn` and defining the beta program that the device must be enrolled into with the `RequireProgram` dictionary. The `RequireProgram` dictionary requires the following keys (all strings):

Key	**Description**
`Description`	A human-readable description of the beta program.
`Token`	The seeding service token that the MDM solution is part of for the organization. This token is used to enroll the device in the corresponding beta program.

In case an organization wants to prevent users from enrolling, they can set the `ProgramEnrollment` key to `AlwaysOff`. This also unenrolls the device from any beta program that it was already manually or automatically enrolled in.