OpenCore

Reference Manual (0.7.~~1~~.2)

[2021.07.17]

Note3: For any other value which you may need to use, it is possible to configure `CsrUtil.efi` as a `TextMode Tools` entry to configure a different value, e.g. use `toggle 0x6F` in `Arguments` to toggle the SIP disabled value set by default by `csrutil disable --no-internal` in Big Sur.

4. `ApECID`
   Type: `plist integer`, 64 bit
   Failsafe: `0`
   Description: Apple Enclave Identifier.

   Setting this value to any non-zero 64-bit integer will allow using personalised Apple Secure Boot identifiers. To use this setting, generate a random 64-bit number with a cryptographically secure random number generator. As an alternative, the first 8 bytes of `SystemUUID` can be used for `ApECID`, this is found in macOS 11 for Macs without the T2 chip.

   With this value set and `SecureBootModel` valid (and not `Disabled`), it is possible to achieve `Full Security` of Apple Secure Boot.

   To start using personalised Apple Secure Boot, the operating system must be reinstalled or personalised. Unless the operating system is personalised, macOS DMG recovery cannot be loaded. In cases where DMG recovery is missing, it can be downloaded by using the `macrecovery` utility and saved in `com.apple.recovery.boot` as explained in the Tips and Tricks section. Note that DMG loading needs to be set to `Signed` to use any DMG with Apple Secure Boot.

   To personalise an existing operating system, use the `bless` command after loading to macOS DMG recovery. Mount the system volume partition, unless it has already been mounted, and execute the following command:

   ```
   bless bless --folder "/Volumes/Macintosh HD/System/Library/CoreServices" \
   bless --folder "/Volumes/Macintosh HD/System/Library/CoreServices" \
     --bootefi --personalize
   ```

   On macOS versions before macOS 11, which introduced a dedicated `x86legacy` model for models without the T2 chip, personalised Apple Secure Boot may not work as expected. When reinstalling the operating system, the macOS Installer from macOS 10.15 and older will often run out of free memory on the `/var/tmp` partition when trying to install macOS with the personalised Apple Secure Boot. Soon after downloading the macOS installer image, an `Unable to verify macOS` error message will appear.

   To workaround this issue, allocate a dedicated RAM disk of 2 MBs for macOS personalisation by entering the following commands in the macOS recovery terminal before starting the installation:

   ```
   disk=$(hdiutil attach -nomount ram://4096)
   diskutil erasevolume HFS+ SecureBoot $disk
   diskutil unmount $disk
   mkdir /var/tmp/OSPersonalizationTemp
   diskutil mount -mountpoint /var/tmp/OSPersonalizationTemp $disk
   ```

5. `AuthRestart`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable `VirtualSMC`-compatible authenticated restart.

   Authenticated restart is a way to reboot FileVault 2 enabled macOS without entering the password. A dedicated terminal command can be used to perform authenticated restarts: `sudo fdesetup authrestart`. It is also used when installing operating system updates.

   VirtualSMC performs authenticated restarts by splitting and saving disk encryption keys between NVRAM and RTC, which despite being removed as soon as OpenCore starts, may be considered a security risk and thus is optional.

6. `BlacklistAppleUpdate`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Ignore boot options trying to update Apple peripheral firmware (e.g. `MultiUpdater.efi`).

Failsafe: `Default`
Description: Apple Secure Boot hardware model.

Sets Apple Secure Boot hardware model and policy. Specifying this value defines which operating systems will be bootable. Operating systems shipped before the specified model was released will not boot.

Valid values:

- `Default` — Recent available model, currently set to ~~`j137`~~`x86legacy`.
- `Disabled` — No model, Secure Boot will be disabled.
- `j137` — `iMacPro1,1 (December 2017). Minimum macOS 10.13.2 (17C2111)`
- `j680` — `MacBookPro15,1 (July 2018). Minimum macOS 10.13.6 (17G2112)`
- `j132` — `MacBookPro15,2 (July 2018). Minimum macOS 10.13.6 (17G2112)`
- `j174` — `Macmini8,1 (October 2018). Minimum macOS 10.14 (18A2063)`
- `j140k` — `MacBookAir8,1 (October 2018). Minimum macOS 10.14.1 (18B2084)`
- `j780` — `MacBookPro15,3 (May 2019). Minimum macOS 10.14.5 (18F132)`
- `j213` — `MacBookPro15,4 (July 2019). Minimum macOS 10.14.5 (18F2058)`
- `j140a` — `MacBookAir8,2 (July 2019). Minimum macOS 10.14.5 (18F2058)`
- `j152f` — `MacBookPro16,1 (November 2019). Minimum macOS 10.15.1 (19B2093)`
- `j160` — `MacPro7,1 (December 2019). Minimum macOS 10.15.1 (19B88)`
- `j230k` — `MacBookAir9,1 (March 2020). Minimum macOS 10.15.3 (19D2064)`
- `j214k` — `MacBookPro16,2 (May 2020). Minimum macOS 10.15.4 (19E2269)`
- `j223` — `MacBookPro16,3 (May 2020). Minimum macOS 10.15.4 (19E2265)`
- `j215` — `MacBookPro16,4 (June 2020). Minimum macOS 10.15.5 (19F96)`
- `j185` — `iMac20,1 (August 2020). Minimum macOS 10.15.6 (19G2005)`
- `j185f` — `iMac20,2 (August 2020). Minimum macOS 10.15.6 (19G2005)`
- `x86legacy` — `Macs without T2 chip and VMs. Minimum macOS 11.0.1 (20B29)`

Warning: Not all Apple Secure Boot models are supported on all hardware configurations. Starting with macOS 12 `x86legacy` is the only Apple Secure Boot model compatible with software update on hardware without T2 chips.

Apple Secure Boot appeared in macOS 10.13 on models with T2 chips. Since `PlatformInfo` and `SecureBootModel` are independent, Apple Secure Boot can be used with any SMBIOS with and without T2. Setting `SecureBootModel` to any valid value but `Disabled` is equivalent to `Medium Security` of Apple Secure Boot. The `ApECID` value must also be specified to achieve `Full Security`. Check `ForceSecureBootScheme` when using Apple Secure Boot on a virtual machine.

Note that enabling Apple Secure Boot is demanding on invalid configurations, faulty macOS installations, and on unsupported setups.

Things to consider:

(a) As with T2 Macs, all unsigned kernel extensions as well as several signed kernel extensions, including NVIDIA Web Drivers, cannot be installed.
(b) The list of cached kernel extensions may be different, resulting in a need to change the list of `Added` or `Forced` kernel extensions. For example, `IO80211Family` cannot be injected in this case.
(c) System volume alterations on operating systems with sealing, such as macOS 11, may result in the operating system being unbootable. Do not try to disable system volume encryption unless Apple Secure Boot is disabled.
(d) Boot failures might occur when the platform requires certain settings, but they have not been enabled because the associated issues were not discovered earlier. Be extra careful with `IgnoreInvalidFlexRatio` or `HashServices`.
(e) Operating systems released before Apple Secure Boot was released (e.g. macOS 10.12 or earlier), will still boot until UEFI Secure Boot is enabled. This is so because Apple Secure Boot treats these as incompatible and they are then handled by the firmware (as Microsoft Windows is).
(f) On older CPUs (e.g. before Sandy Bridge), enabling Apple Secure Boot might cause slightly slower loading (by up to 1 second).
(g) As the `Default` value will increase with time to support the latest major released operating system, it is not recommended to use the `ApECID` and the `Default` settings together.

Note: all protocol instances are installed prior to driver loading.

8. `Quirks`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual firmware quirks described in the Quirks Properties section below.

9. `ReservedMemory`
   Type: `plist array`
   Description: To be filled with `plist dict` values, describing memory areas exclusive to specific firmware and hardware functioning, which should not be used by the operating system. Examples of such memory regions could be the second 256 MB corrupted by the Intel HD 3000 or an area with faulty RAM. Refer to the ReservedMemory Properties section below for details.

11.7 APFS Properties

1. `EnableJumpstart`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load embedded APFS drivers from APFS containers.

   An APFS EFI driver is bundled in all bootable APFS containers. This option performs the loading of signed APFS drivers (consistent with the `ScanPolicy`). Refer to the "EFI Jumpstart" section of the Apple File System Reference for details.

2. `GlobalConnect`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform full device connection during APFS loading.

   Every handle is connected recursively instead of the partition handle connection typically used for APFS driver loading. This may result in additional time being taken but can sometimes be the only way to access APFS partitions on certain firmware, such as those on older HP laptops.

3. `HideVerbose`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Hide verbose output from APFS driver.

   APFS verbose output can be useful for debugging.

4. `JumpstartHotPlug`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load APFS drivers for newly connected devices.

   Permits APFS USB hot plug which enables loading APFS drivers, both at OpenCore startup and during OpenCore picker display. Disable if not required.

5. `MinDate`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver date.

   The APFS driver date connects the APFS driver with the calendar release date. Apple ultimately drops support for older macOS releases and APFS drivers from such releases may contain vulnerabilities that can be used to compromise a computer if such drivers are used after support ends. This option permits restricting APFS drivers to current macOS versions.

   - `0` — require the default supported release date of APFS in OpenCore. The default release date will increase with time and thus this setting is recommended. Currently set to ~~2018~~2021/~~06~~01/~~21.~~01.
   - `-1` — permit any release date to load (strongly discouraged).
   - Other — use custom minimal APFS release date, e.g. `20200401` for 2020/04/01. APFS release dates can be found in OpenCore boot log and `OcApfsLib`.

6. `MinVersion`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver version.

   The APFS driver version connects the APFS driver with the macOS release. Apple ultimately drops support for older macOS releases and APFS drivers from such releases may contain vulnerabilities that can be used to compromise a computer if such drivers are used after support ends. This option permits restricting APFS drivers to current macOS versions.

   - `0` — require the default supported version of APFS in OpenCore. The default version will increase with time and thus this setting is recommended. Currently set to ~~the latest point release from High Sierra from App Store (~~allow macOS Big Sur and newer (~~748077008000000~~1600000000000000).
   - `-1` — permit any version to load (strongly discouraged).
   - Other — use custom minimal APFS version, e.g. `1412101001000000` from macOS Catalina 10.15.4. APFS versions can be found in OpenCore boot log and `OcApfsLib`.

11.8 AppleInput Properties

1. `AppleEvent`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Determine whether OC builtin or OEM Apple Event protocol is used.

   This option determines whether Apple's OEM Apple Event protocol is used (where available), or whether OpenCore's reversed engineered and updated re-implementation is used. In general OpenCore's re-implementation should be preferred, since it contains updates such as noticeably improved fine mouse cursor movement and configurable key repeat delays.

   - `Auto` — Use OEM Apple Event implementation if available, connected and recent enough to be used, otherwise use OC reimplementation. On non-Apple hardware this will use the OpenCore builtin implementation. On some Macs (e.g. classic Mac Pro) this will find the Apple implementation. On both older and newer Macs than this, this option will always or often use the OC implementation. On older Macs this is because the implementation available is too old to be used, on newer Macs it is because of optimisations added by Apple which do not connect the Apple Event protocol except when needed – e.g. except when the Apple boot picker is explicitly started. Due to its somewhat unpredicatable results, this option is not normally recommended.
   - `Builtin` — Always use OpenCore's updated re-implementation of the Apple Event protocol. Use of this setting is recommended even on Apple hardware, due to improvements (better fine mouse control, configurable key delays) made in the OC re-implementation of the protocol.
   - `OEM` — Assume Apple's protocol will be available at driver connection. On all Apple hardware where a recent enough Apple OEM version of the protocol is available – whether or not connected automatically by Apple's firmware – this option will reliably access the Apple implementation. On all other systems, this option will result in no keyboard or mouse support. For the reasons stated, `Builtin` is recommended in preference to this option in most cases.

2. `CustomDelays`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable custom key repeat delays when using the OpenCore implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

   - `true` — The values of `KeyInitialDelay` and `KeySubsequentDelay` are used.
   - `false` — Apple default values of 500ms (`50`) and 50ms (`5`) are used.

3. `KeyInitialDelay`
   Type: `plist integer`
   Failsafe: `50` (500ms before first key repeat)
   Description: Configures the initial delay before keyboard key repeats in OpenCore implementation of Apple Event protocol, in units of 10ms.

   The Apple OEM default value is `50` (500ms).

Note 1: On systems not using `KeySupport`, this setting may be freely used to configure key repeat behaviour.

Note 2: On systems using `KeySupport`, but which do not show the 'two long delays' behavior (see Note 3) and/or which always show a solid 'set default' indicator (see `KeyForgetThreshold`) then this setting may also be freely used to configure key repeat initial delay behaviour, except that it should never be set to less than `KeyForgetThreshold` to avoid uncontrolled key repeats.

Note 3: On some systems using `KeySupport`, you may find that you see one additional slow key repeat before normal speed key repeat starts, when holding a key down. If so, you may wish to configure `KeyInitialDelay` and `KeySubsequentDelay` according to the instructions at Note 3 of `KeySubsequentDelay`.

4. `KeySubsequentDelay`
   Type: `plist integer`
   Failsafe: `5` (50ms between subsequent key repeats)
   Description: Configures the gap between keyboard key repeats in OpenCore implementation of Apple Event protocol, in units of 10ms.

   The Apple OEM default value is `5` (50ms). `0` is an invalid value for this option (will issue a debug log warning and use `1` instead).

   Note 1: On systems not using `KeySupport`, this setting may be freely used to configure key repeat behaviour.

   Note 2: On systems using `KeySupport`, but which do not show the 'two long delays' behaviour (see Note 3) and/or which always show a solid 'set default' indicator (see `KeyForgetThreshold`) (which should apply to many/most systems using `AMI KeySupport` mode) then this setting may be freely used to configure key repeat subsequent delay behaviour, except that it should never be set to less than `KeyForgetThreshold` to avoid uncontrolled key repeats.

   Note 3: On some systems using `KeySupport`, particularly `KeySupport` in non-`AMI` mode, you may find that after configuring `KeyForgetThreshold` you get one additional slow key repeat before normal speed key repeat starts, when holding a key down. On systems where this is the case, it is an unavoidable artefect of using `KeySupport` to emulate raw keyboard data, which is not made available by UEFI. While this 'two long delays' issue has minimal effect on overall usability, nevertheless you may wish to resolve it, and it is possible to do so as follows:

   - Set `CustomDelays` to `true`
   - Set `KeyInitialDelay` to `0`
   - Set `KeySubsequentDelay` to at least the value of your `KeyForgetThreshold` setting

   The above procedure works as follows:

   - Setting `KeyInitialDelay` to `0` cancels the Apple Event initial repeat delay (when using the OC builtin Apple Event implementation with `CustomDelays` enabled), therefore the only long delay you will see is the the non-configurable and non-avoidable initial long delay introduced by the BIOS key support on these machines.
   - Key-smoothing parameter `KeyForgetThreshold` effectively acts as the shortest time for which a key can appear to be held, therefore a key repeat delay of less than this will guarantee at least one extra repeat for every key press, however quickly the key is physically tapped.
   - In the unlikely event that you still get frequent, or occasional, double key responses after setting `KeySubsequentDelay` equal to your system's value of `KeyForgetThreshold`, then increase `KeySubsequentDelay` by one or two more until this effect goes away.

5. `GraphicsInputMirroring`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Apple's own implementation of AppleEvent prevents keyboard input during graphics applications from appearing on the basic console input stream.

   With the default setting of `false`, OC's builtin implementation of AppleEvent replicates this behaviour.

   On non-Apple hardware this can stop keyboard input working in graphics-based applications such as Windows BitLocker which use non-Apple key input methods.

   The recommended setting on all hardware is `true`.

Note: AppleEvent's default behaviour is intended to prevent unwanted queued keystrokes from appearing after exiting graphics-based UEFI applications; this issue is already handled separately within OpenCore.

- `true` — Allow keyboard input to reach graphics mode apps which are not using Apple input protocols.
- `false` — Prevent key input mirroring to non-Apple protocols when in graphics mode.

6. `PointerSpeedDiv`
   Type: `plist integer`
   Failsafe: `1`
   Description: Configure pointer speed divisor in OpenCore implementation of Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

   Configures the divisor for pointer movements. The Apple OEM default value is `1`. `0` is an invalid value for this option.

   Note: The recommended value for this option is `1`. This value may optionally be modified in combination with `PointerSpeedMul`, according to user preference, to achieve customised mouse movement scaling.

7. `PointerSpeedMul`
   Type: `plist integer`
   Failsafe: `1`
   Description: Configure pointer speed multiplier in OpenCore implementation of Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

   Configures the multiplier for pointer movements. The Apple OEM default value is `1`.

   Note: The recommended value for this option is `1`. This value may optionally be modified in combination with `PointerSpeedDiv`, according to user preference, to achieve customised mouse movement scaling.

11.9 Audio Properties

1. `AudioCodec`
   Type: `plist integer`
   Failsafe: `0`
   Description: Codec address on the specified audio controller for audio support.

   This typically contains the first audio codec address on the builtin analog audio controller (`HDEF`). Audio codec addresses, e.g. `2`, can be found in the debug log (marked in bold-italic):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative, this value can be obtained from `IOHDACodecDevice` class in I/O Registry containing it in `IOHDACodecAddress` field.

2. `AudioDevice`
   Type: `plist string`
   Failsafe: Empty
   Description: Device path of the specified audio controller for audio support.

   This typically contains builtin analog audio controller (`HDEF`) device path, e.g. `PciRoot(0x0)/Pci(0x1b,0x0)`. The list of recognised audio controllers can be found in the debug log (marked in bold-italic):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative, `gfxutil -f HDEF` command can be used in macOS. Specifying an empty device path will result in the first available audio controller being used.

3. `AudioOut`
   Type: `plist integer`
   Failsafe: `0`
   Description: Index of the output port of the specified codec starting from 0.

Failsafe: `false`
Description: Replaces the Apple Debug Log protocol with a builtin version.

4. `AppleEg2Info`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces the Apple EFI Graphics 2 protocol with a builtin version.

   Note 1: This protocol allows newer `EfiBoot` versions (at least 10.15) to expose screen rotation to macOS. Refer to `ForceDisplayRotationInEFI` variable description on how to set screen rotation angle.

   Note 2: On systems without native support for `ForceDisplayRotationInEFI`, `DirectGopRendering=true` is also required for this setting to have ~~a visible~~ an effect.

5. `AppleFramebufferInfo`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces the Apple Framebuffer Info protocol with a builtin version. This may be used to override framebuffer information on VMs and legacy Macs to improve compatibility with legacy EfiBoot such as the one in macOS 10.4.

   Note: The current implementation of this property results in it only being active when GOP is available (it is always equivalent to `false` otherwise).

6. `AppleImageConversion`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces the Apple Image Conversion protocol with a builtin version.

7. `AppleImg4Verification`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces the Apple IMG4 Verification protocol with a builtin version. This protocol is used to verify `im4m` manifest files used by Apple Secure Boot.

8. `AppleKeyMap`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces Apple Key Map protocols with builtin versions.

9. `AppleRtcRam`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Replaces the Apple RTC RAM protocol with a builtin version.

   Note: Builtin version of Apple RTC RAM protocol may filter out I/O attempts to certain RTC memory addresses. The list of addresses can be specified in `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:rtc-blacklist` variable as a data array.

10. `AppleSecureBoot`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Replaces the Apple Secure Boot protocol with a builtin version.

11. `AppleSmcIo`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Replaces the Apple SMC I/O protocol with a builtin version.

    This protocol replaces the legacy `VirtualSmc` UEFI driver, and is compatible with any SMC kernel extension. However, in case the `FakeSMC` kernel extension is used, manual NVRAM key variable addition may be needed.

12. `AppleUserInterfaceTheme`
    Type: `plist boolean`