OpenCore

Reference Manual (0.6.3.4)

[2020.12.06]

loaded by the firmware by default according to UEFI specification, and `Bootstrap.efi` can be registered as a custom option to let OpenCore coexist with operating systems using `BOOTx64.efi` as their own loaders (e.g. Windows), see `BootProtect` for more details.

- `boot`
  Duet bootstrap loader, which initialises UEFI environment on legacy BIOS firmware and loads `OpenCore.efi` similarly to other bootstrap loaders. Modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- `ACPI`
  Directory used for storing supplemental ACPI information for `ACPI` section.
- `Drivers`
  Directory used for storing supplemental `UEFI` drivers for `UEFI` section.
- `Kexts`
  Directory used for storing supplemental kernel information for `Kernel` section.
- `Resources`
  Directory used for storing media resources, such as audio files for screen reader support. See `UEFI Audio Properties` section for more details. This directory also contains image files for graphical user interface. See OpenCanopy section for more details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter driver responsible for operating system loading. The directory `OpenCore.efi` resides is called the `root directory`. By default `root directory` is set to `EFI\OC`, however, when launching `OpenCore.efi` directly or through `Bootstrap.efi`, other directories containing `OpenCore.efi` can also be supported.
- `config.plist`
  `OC Config`.
- `vault.plist`
  Hashes for all files potentially loadable by `OC Config`.
- `vault.sig`
  Signature for `vault.plist`.
- `SysReport`
  Directory containing system reports generated by `SysReport` option.
- `nvram.plist`
  OpenCore variable import file.
- `opencore-YYYY-MM-DD-HHMMSS.txt`
  OpenCore log file.
- `panic-YYYY-MM-DD-HHMMSS.txt`
  Kernel panic log file.

Note: It is not guaranteed that paths longer than `OC_STORAGE_SAFE_PATH_MAX` (128 characters including `0`-terminator) will be accessible within OpenCore.

3.2 Installation and Upgrade

To install OpenCore reflect the Configuration Structure described in the previous section on a EFI volume of a GPT partition. While corresponding sections of this document do provide some information regarding external resources such as ACPI tables, UEFI drivers, or kernel extensions (kexts), completeness of the matter is out of the scope of this document. Information about kernel extensions may be found in a separate Kext List document available in OpenCore repository. Vaulting information is provided in Security Properties section of this document.

`OC config`, just like any property lists can be edited with any stock textual editor (e.g. nano, vim), but specialised software may provide better experience. On macOS the preferred GUI application is Xcode. For a lightweight cross-platform and open-source alternative ProperTree editor can be utilised.

For BIOS booting a third-party UEFI environment provider will have to be used. `OpenDuetPkg` is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system, `OpenDuetPkg` can be installed with a dedicated tool — BootInstall (bundled with OpenCore). Third-party utilities can be used to perform this on systems other than macOS.

For upgrade purposes refer to `Differences.pdf` document, providing the information about the changes affecting the configuration compared to the previous release, and `Changelog.md` document, containing the list of modifications

5 Booter

5.1 Introduction

This section allows to apply different kinds of UEFI modifications on Apple bootloader (`boot.efi`). The modifications currently provide various patches and environment alterations for different firmware. Some of these features were originally implemented as a part of AptioMemoryFix.efi, which is no longer maintained. See Tips and Tricks section for migration steps.

If this is used for the first time on a customised firmware, there is a list of checks to do first. Prior to starting, the following requirements should be fulfilled:

- Most up-to-date UEFI firmware (check the motherboard vendor website).
- `Fast Boot` and `Hardware Fast Boot` disabled in firmware settings if present.
- `Above 4G Decoding` or similar enabled in firmware settings if present. Note, that on some motherboards (notably ASUS WS-X299-PRO) this option causes adverse effects, and must be disabled. While no other motherboards with the same issue are known, this option should be checked first whenever erratic boot failures are encountered.
- `DisableIoMapper` quirk enabled, or `VT-d` disabled in firmware settings if present, or ACPI DMAR table deleted.
- **No** 'slide' boot argument present in NVRAM or anywhere else. It is not necessary unless the system cannot be booted at all or `No slide values are usable! Use custom slide!` message can be seen in the log.
- `CFG Lock` (MSR `0xE2` write protection) disabled in firmware settings if present. Consider patching it if no option is available (for advanced users only). See VerifyMsrE2 notes for more details.
- `CSM` (Compatibility Support Module) disabled in firmware settings if present. On NVIDIA 6xx/AMD 2xx or older, GOP ROM may have to be flashed first. Use GopUpdate (see the second post) or AMD UEFI GOP MAKER in case of any potential confusion.
- `EHCI/XHCI Hand-off` enabled in firmware settings `only` if boot stalls unless USB devices are disconnected.
- `VT-x`, `Hyper Threading`, `Execute Disable Bit` enabled in firmware settings if present.
- While it may not be required, sometimes `Thunderbolt support`, `Intel SGX`, and `Intel Platform Trust` may have to be disabled in firmware settings present.

When debugging sleep issues Power Nap and automatic power off may be (temporarily) disabled, which appear to sometimes cause wake to black screen or boot loop issues on older platforms. The particular issues may vary, but in general ACPI tables should be looked up first. Here is an example of a bug found in some Z68 motherboards. To turn Power Nap and the others off run the following commands in Terminal:

```
sudo pmset autopoweroff 0
sudo pmset powernap 0
sudo pmset standby 0
```

Note: These settings may reset at hardware change and in certain other circumstances. To view their current state use `pmset -g` command in Terminal.

5.2 Properties

1. `MmioWhitelist`
   Type: `plist array`
   Description: Designed to be filled with `plist dict` values, describing addresses critical for particular firmware functioning when `DevirtualiseMmio` quirk is in use. See MmioWhitelist Properties section below.

2. `Patch`
   Type: `plist array`
   Failsafe: Empty
   Description: Perform binary patches in booter.

   Designed to be filled with `plist dictionary` values, describing each patch. See Patch Properties section below.

3. `Quirks`
   Type: `plist dict`
   Description: Apply individual booter quirks described in Quirks Properties section below.

5.3 MmioWhitelist Properties

1. `Address`
   Type: `plist integer`
   Failsafe: `0`
   Description: Exceptional MMIO address, which memory descriptor should be left virtualised (unchanged) by `DevirtualiseMmio`. This means that the firmware will be able to directly communicate with this memory region during operating system functioning, because the region this value is in will be assigned a virtual address.

   The addresses written here must be part of the memory map, have `EfiMemoryMappedIO` type and `EFI_MEMORY_RUNTIME` attribute (highest bit) set. To find the list of the candidates the debug log can be used.

2. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This address will be devirtualised unless set to `true`.

5.4 Patch Properties

1. `Arch`
   Type: `plist string`
   Failsafe: `Any`
   Description: Booter patch architecture (`Any`, `i386`, `x86_64`).

2. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. `Count`
   Type: `plist integer`
   Failsafe: `0`
   Description: Number of patch occurrences to apply. `0` applies the patch to all occurrences found.

4. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This booter patch will not be used unless set to `true`.

5. `Find`
   Type: `plist data`
   Failsafe: Empty data
   Description: Data to find. This must equal to `Replace` in size.

6. `Identifier`
   Type: `plist string`
   Failsafe: Empty string
   Description: `Apple` for macOS booter (generally `boot.efi`); or a name with suffix (e.g. `bootmgfw.efi`) for a specific booter; or `Any` / empty string (failsafe) to match any booter.

7. `Limit`
   Type: `plist integer`
   Failsafe: `0`
   Description: Maximum number of bytes to search for. Can be set to `0` to look through the whole booter.

8. `Mask`
   Type: `plist data`
   Failsafe: Empty data
   Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Can be set to empty data to be ignored. Must equal to `Find` in size otherwise.

9. `Replace`
   Type: `plist data`
   Failsafe: Empty data
   Description: Replacement data of one or more bytes.

10. `ReplaceMask`
    Type: `plist data`
    Failsafe: Empty data
    Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Can be set to empty data to be ignored. Must equal to `Replace` in size otherwise.

11. `Skip`
    Type: `plist integer`
    Failsafe: `0`
    Description: Number of found occurrences to be skipped before replacement is done.

5.5 Quirks Properties

1. `AllowRelocationBlock`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Allows booting macOS through a relocation block.

   Relocation block is a scratch buffer allocated in lower 4 GB to be used for loading the kernel and related structures by EfiBoot on firmwares where lower memory is otherwise occupied by the (assumed to be) non-runtime data. Right before kernel startup the relocation block is copied back to lower addresses. Similarly all the other addresses pointing to relocation block are also carefully adjusted. Relocation block can be used when:

   - No better slide exists (all the memory is used)
   - `slide=0` is forced (by an argument or safe mode)
   - KASLR (slide) is unsupported (this is macOS 10.7 or older)

   This quirk requires `ProvideCustomSlide` to also be enabled and generally needs `AvoidRuntimeDefrag` to work correctly. Hibernation is not supported when booting with a relocation block (but relocation block is not always used when the quirk is enabled).

   Note: While this quirk is required to run older macOS versions on platforms with used lower memory it is not compatible with some hardware and macOS 11. In this case you may try to use `EnableSafeModeSlide` instead.

2. `AvoidRuntimeDefrag`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Protect from boot.efi runtime memory defragmentation.

   This option fixes UEFI runtime services (date, time, NVRAM, power control, etc.) support on firmware that uses SMM backing for select services such as variable storage. SMM may try to access physical addresses, but they get moved by boot.efi.

   Note: Most types of firmware, apart from Apple and VMware, need this quirk.

3. `DevirtualiseMmio`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Remove runtime attribute from select MMIO regions.

   This option reduces stolen memory footprint from the memory map by removing runtime bit for known memory regions. This quirk may result in the increase of KASLR slides available, but is not necessarily compatible with

6. `DisableIoMapper`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.8 (not required for older)
   Description: Disables `IOMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

   Note: This option is a preferred alternative to deleting `DMAR` ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.

7. `DisableLinkeditJettison`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: ~~11.0~~11
   Description: Disables `__LINKEDIT` jettison code.

   This option lets `Lilu.kext` and possibly some others function in macOS Big Sur with best performance without `keepsyms=1` boot argument.

8. `DisableRtcChecksum`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.4
   Description: Disables primary checksum (`0x58-0x59`) writing in AppleRTC.

   Note 1: This option will not protect other areas from being overwritten, see RTCMemoryFixup kernel extension if this is desired.

   Note 2: This option will not protect areas from being overwritten at firmware stage (e.g. macOS bootloader), see `AppleRtcRam` protocol description if this is desired.

9. `ExtendBTFeatureFlags`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.8
   Description: Set `FeatureFlags` to `0x0F` for full functionality of Bluetooth, including Continuity.

   Note: This option is a substitution for BT4LEContinuityFixup.kext, which does not function properly due to late patching progress.

10. `ExternalDiskIcons`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.4
    Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

    Note: This option should be avoided whenever possible. Modern firmware usually have compatible AHCI controllers.

11. `ForceSecureBootScheme`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: ~~11.0~~11
    Description: Force `x86` scheme for IMG4 verification.

    Note: This option is required on virtual machines when using `SecureBootModel` different from `x86legacy`.

12. `IncreasePciBarSize`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.10
    Description: Increases 32-bit PCI bar size in IOPCIFamily from 1 to 4 GBs.

    Note: This option should be avoided whenever possible. In general the necessity of this option means misconfigured or broken firmware.

Model	10.6 (minimal)	10.6 (client)	10.6 (server)	10.7 (any)
Macmini	4,x (Mid 2010)	5,x (Mid 2011)	4,x (Mid 2010)	3,x (Early 2009)
MacBook	Unsupported	Unsupported	Unsupported	5,x (2009/09)
MacBookAir	Unsupported	Unsupported	Unsupported	2,x (Late 2008)
MacBookPro	4,x (Early 2008)	8,x (Early 2011)	8,x (Early 2011)	3,x (Mid 2007)
iMac	8,x (Early 2008)	12,x (Mid 2011)	12,x (Mid 2011)	7,x (Mid 2007)
MacPro	3,x (Early 2008)	5,x (Mid 2010)	3,x (Early 2008)	3,x (Early 2008)
Xserve	2,x (Early 2008)	2,x (Early 2008)	2,x (Early 2008)	2,x (Early 2008)

Note: `3+2` and `6+4` hotkeys to choose the preferred architecture are unsupported due to being handled by EfiBoot and thus being hard to properly detect.

3. `KernelCache`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Prefer specified kernel cache type (`Auto`, `Cacheless`, `Mkext`, `Prelinked`) when available.

   Different variants of macOS support different kernel caching variants designed to improve boot performance. This setting prevents the use of faster kernel caching variants if slower variants are available for debugging and stability reasons. I.e., by specifying `Mkext`, `Prelinked` will be disabled for e.g. 10.6 but not for 10.7.

   The list of available kernel caching types and its current support in OpenCore is listed below.

macOS	i386 NC	i386 MK	i386 PK	x86_64 NC	x86_64 MK	x86_64 PK	x86_64 KC
10.4	YES	YES (V1)	NO (V1)	—	—	—	—
10.5	YES	YES (V1)	NO (V1)	—	—	—	—
10.6	YES	YES (V2)	YES (V2)	YES	YES (V2)	YES (V2)	—
10.7	YES	—	YES (V3)	YES	—	YES (V3)	—
10.8-10.9	—	—	—	YES	—	YES (V3)	—
10.10-10.15	—	—	—	—	—	YES (V3)	—
~~11.0~~11+	—	—	—	—	—	YES (V3)	YES

Note: First version (V1) of 32-bit `prelinkedkernel` is unsupported due to kext symbol tables being corrupted by the tools. On these versions `Auto` will block `prelinkedkernel` booting. This also makes `keepsyms=1` for kext frames broken on these systems.

For `Tools` OpenCore will try to load a custom icon and fallback to the default icon:
  - `ResetNVRAM` — `Resources\Image\ResetNVRAM.icns` — `ResetNVRAM.icns` from icons directory.
  - `Tools\<TOOL_RELATIVE_PATH>.icns` — icon near the tool file with appended `.icns` extension.

For custom boot `Entries` OpenCore will try to load a custom icon and fallback to the volume icon or the default icon:
  - `<ENTRY_PATH>.icns` — icon near the entry file with appended `.icns` extension.

For all other entries OpenCore will try to load a volume icon and fallback to the default icon:
  - `.VolumeIcon.icns` file at `Preboot` root for APFS.
  - `.VolumeIcon.icns` file at volume root for other filesystems.

Volume icons can be set in Finder. Note, that enabling this may result in external and internal icons to be indistinguishable.

- `0x0002` — `OC_ATTR_USE_DISK_LABEL_FILE`, provides custom rendered titles for boot entries:
  - `.disk_label` (`.disk_label_2x`) file near bootloader for all filesystems.
  - `<TOOL_NAME>.lbl` (`<TOOL_NAME>.l2x`) file near tool for `Tools`.

  Prerendered labels can be generated via `disklabel` utility or `bless` command. When disabled or missing text labels (`.contentDetails` or `.disk_label.contentDetails`) are to be rendered instead.
- `0x0004` — `OC_ATTR_USE_GENERIC_LABEL_IMAGE`, provides predefined label images for boot entries without custom entries. May give less detail for the actual boot entry.
- `0x0008` — `OC_ATTR_USE_ALTERNATE_ICONS`, changes used icon set to an alternate one if it is supported. For example, this could make a use of old-style icons with a custom background colour.
- `0x0010` — `OC_ATTR_USE_POINTER_CONTROL`, enable pointer control in the picker when available. For example, this could make use of mouse or trackpad to control UI elements.

5. `PickerAudioAssist`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable screen reader by default in boot picker.

   For macOS bootloader screen reader preference is set in `preferences.efires` archive in `isVOEnabled.int32` file and is controlled by the operating system. For OpenCore screen reader support this option is an independent equivalent. Toggling screen reader support in both OpenCore boot picker and macOS bootloader FileVault 2 login window can also be done with `Command` + `F5` key combination.

   Note: screen reader requires working audio support, see `UEFI Audio Properties` section for more details.

6. `PollAppleHotKeys`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable `modifier hotkey` handling in boot picker.

   In addition to `action hotkeys`, which are partially described in `PickerMode` section and are normally handled by Apple BDS, there exist modifier keys, which are handled by operating system bootloader, namely `boot.efi`. These keys allow to change operating system behaviour by providing different boot modes.

   On some types of firmware, it may be problematic to use modifier keys due to driver incompatibilities. To workaround this problem this option allows registering select hotkeys in a more permissive manner from within boot picker. Such extensions include the support of tapping on keys in addition to holding and pressing `Shift` along with other keys instead of just `Shift` alone, which is not detectable on many PS/2 keyboards. This list of known `modifier hotkeys` includes:

   - `CMD+C+MINUS` — disable board compatibility checking.
   - `CMD+K` — boot release kernel, similar to `kcsuffix=release`.
   - `CMD+S` — single user mode.
   - `CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.
   - `CMD+V` — verbose mode.
   - `Shift` — safe mode.

7. `ShowPicker`
   Type: `plist boolean`

Description: Allow `CMD+OPT+P+R` handling and enable showing `NVRAM Reset` entry in boot picker.

Note 1: It is known that some Lenovo laptops have a firmware bug, which makes them unbootable after performing NVRAM reset. See acidanthera/bugtracker#995 for more details.

Note 2: Resetting NVRAM will also erase all the boot options otherwise not backed up with bless (e.g. Linux).

2. `AllowSetDefault`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Allow `CTRL+Enter` and `CTRL+Index` handling to set the default boot option in boot picker.

3. `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, make sure to generate a random 64-bit number with a cryptographically secure random number generator. As an alternative, first 8 bytes of `SystemUUID` can be used for `ApECID`, this is found in macOS ~~11.0~~ 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 will have to be reinstalled or personalised. Unless the operating system is personalised, macOS DMG recovery cannot be loaded. If DMG recovery is missing, it can be downloaded with `macrecovery` utility and put to `com.apple.recovery.boot` as explained in 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 `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" \
     --bootefi --personalize
   ```

   Before macOS ~~11.0~~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, macOS Installer from macOS 10.15 and older, will usually 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 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
   ```

4. `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 restart by saving disk encryption key split in NVRAM and RTC, which despite being removed as soon as OpenCore starts, may be considered a security risk and thus is optional.

5. `BlacklistAppleUpdate`
   Type: `plist boolean`

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

Note: This option exists due to some operating systems, namely macOS Big Sur, being incapable of disabling firmware updates with the NVRAM variable (`run-efi-updater`).

6. `BootProtect`
Type: `plist string`
Failsafe: `None`
Description: Attempt to provide bootloader persistence.

Valid values:

- `None` — do nothing.
- `Bootstrap` — create or update top-priority `\EFI\OC\Bootstrap\Bootstrap.efi` boot option ~~(`Boot9696`)~~ in UEFI variable storage at bootloader startup. For this option to work `RequestBootVarRouting` is required to be enabled.
- `BootstrapShort` — create a short boot option instead of a complete one, otherwise equivalent to `Bootstrap`. This variant is useful for some older firmwares, Insyde in particular, but possibly others, which cannot handle full device paths.

This option provides integration with third-party operating system installation and upgrade at the times they overwrite `\EFI\BOOT\BOOTx64.efi` file. By creating a custom option in `Bootstrap` mode this file path becomes no longer used for bootstrapping OpenCore.

Note 1: Some types of firmware may have faulty NVRAM, no boot option support, or other incompatibilities. While unlikely, the use of this option may even cause boot failures. This option should be used without any warranty exclusively on the boards known to be compatible. Check acidanthera/bugtracker#1222 for some known issues with Haswell and other boards.

Note 2: Be aware that while NVRAM reset executed from OpenCore should not erase the boot option created in `Bootstrap`, executing NVRAM reset prior to loading OpenCore will remove it. For significant implementation updates (e.g. in OpenCore 0.6.4) make sure to perform NVRAM reset with `Bootstrap` disabled before reenabling.

7. `DmgLoading`
Type: `plist string`
Failsafe: `Signed`
Description: Define Disk Image (DMG) loading policy used for macOS Recovery.

Valid values:

- `Disabled` — loading DMG images will fail. `Disabled` policy will still let macOS Recovery to load in most cases as there usually are `boot.efi` files compatible with Apple Secure Boot. Manually downloaded DMG images stored in `com.apple.recovery.boot` directories will not load, however.
- `Signed` — only Apple-signed DMG images will load. Due to Apple Secure Boot design `Signed` policy will let any Apple-signed macOS Recovery to load regardless of Apple Secure Boot state, which may not always be desired.
- `Any` — any DMG images will mount as normal filesystems. `Any` policy is strongly not recommended and will cause a boot failure when Apple Secure Boot is activated.

8. `EnablePassword`
Type: `plist boolean`
Failsafe: `false`
Description: Enable password protection to allow sensitive operations.

Password protection ensures that sensitive operations such as booting a non-default operating system (e.g. macOS recovery or a tool), resetting NVRAM storage, trying to boot into a non-default mode (e.g. verbose mode or safe mode) are not allowed without explicit user authentication by a custom password. Currently password and salt are hashed with 5000000 iterations of SHA-512.

Note: This functionality is currently in development and is not ready for daily usage.

- 0x00020000 (bit 17) — OC_SCAN_ALLOW_DEVICE_SASEX, allow scanning SAS and Mac NVMe devices.
- 0x00040000 (bit 18) — OC_SCAN_ALLOW_DEVICE_SCSI, allow scanning SCSI devices.
- 0x00080000 (bit 19) — OC_SCAN_ALLOW_DEVICE_NVME, allow scanning NVMe devices.
- 0x00100000 (bit 20) — OC_SCAN_ALLOW_DEVICE_ATAPI, allow scanning CD/DVD devices and old SATA.
- 0x00200000 (bit 21) — OC_SCAN_ALLOW_DEVICE_USB, allow scanning USB devices.
- 0x00400000 (bit 22) — OC_SCAN_ALLOW_DEVICE_FIREWIRE, allow scanning FireWire devices.
- 0x00800000 (bit 23) — OC_SCAN_ALLOW_DEVICE_SDCARD, allow scanning card reader devices.
- 0x01000000 (bit 24) — OC_SCAN_ALLOW_DEVICE_PCI, allow scanning devices directly connected to PCI bus (e.g. VIRTIO).

Note: Given the above description, 0xF0103 value is expected to allow scanning of SATA, SAS, SCSI, and NVMe devices with APFS file system, and prevent scanning of any devices with HFS or FAT32 file systems in addition to not scanning APFS file systems on USB, CD, and FireWire drives. The combination reads as:

- OC_SCAN_FILE_SYSTEM_LOCK
- OC_SCAN_DEVICE_LOCK
- OC_SCAN_ALLOW_FS_APFS
- OC_SCAN_ALLOW_DEVICE_SATA
- OC_SCAN_ALLOW_DEVICE_SASEX
- OC_SCAN_ALLOW_DEVICE_SCSI
- OC_SCAN_ALLOW_DEVICE_NVME

15. SecureBootModel
Type: plist string
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.
- 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 and VMs without T2 chip minimum macOS 11.0.1 (20B29)

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.

Enabling Apple Secure Boot is more demanding to incorrect configurations, buggy macOS installations, and unsupported setups. Things to consider:

(a) As with T2 Macs, unsigned kernel drivers and several signed kernel drivers, including NVIDIA Web Drivers, cannot be installed.

(b) The list of cached drivers may be different, resulting in the need to change the list of `Added` or `Forced` kernel drivers. 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) If the platform requires certain settings, but they were not enabled, because the obvious issues did not trigger before, boot failure might occur. Be extra careful with `IgnoreInvalidFlexRatio` or `HashServices`.
(e) Operating systems released before Apple Secure Boot landed (e.g. macOS 10.12 or earlier) will still boot until UEFI Secure Boot is enabled. This is so, because from Apple Secure Boot point they are treated as incompatible and are assumed to be 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) Since `Default` value will increase with time to support the latest major release operating system, it is not recommended to use `ApECID` and `Default` value together.
(h) Installing macOS with Apple Secure Boot enabled is not possible while using HFS+ target volume. This may include HFS+ formatted drives when no spare APFS drive is available.

Sometimes the already installed operating system may have outdated Apple Secure Boot manifests on the `Preboot` partition causing boot failure. If there is "OCB: Apple Secure Boot prohibits this boot entry, enforcing!" message, it is likely the case. When this happens, either reinstall the operating system or copy the manifests (files with `.im4m` extension, such as `boot.efi.j137.im4m`) from `/usr/standalone/i386` to `/Volumes/Preboot/<UUID>/System/Library/CoreServices`. Here `<UUID>` is the system volume identifier. On HFS+ installations the manifests should be copied to `/System/Library/CoreServices` on the system volume.

For more details on how to configure Apple Secure Boot with UEFI Secure Boot refer to UEFI Secure Boot section.

8.6 Entry Properties

1. `Arguments`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.

2. `Auxiliary`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This entry will not be listed by default when `HideAuxiliary` is set to `true`.

3. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

4. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This entry will not be listed unless set to `true`.

5. `Name`
   Type: `plist string`
   Failsafe: Empty string
   Description: Human readable entry name displayed in boot picker.

6. `Path`
   Type: `plist string`
   Failsafe: Empty string
   Description: Entry location depending on entry type.

   - `Entries` specify external boot options, and therefore take device paths in `Path` key. These values are not checked, thus be extremely careful. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../\EFI\COOL.EFI`

- `Tools` specify internal boot options, which are part of bootloader vault, and therefore take file paths relative to `OC/Tools` directory. Example: `OpenShell.efi`.

7. `RealPath`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Pass full path to the tool when launching.

   Passing tool directory may be unsafe for tool accidentally trying to access files without checking their integrity and thus should generally be disabled. Reason to enable this property may include cases where tools cannot work without external files or may need them for better function (e.g. `memtest86` for logging and configuration or `Shell` for automatic script execution).

   Note: This property is only valid for `Tools`. For `Entries` this property cannot be specified and is always `true`.

8. `TextMode`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Run the entry in text mode instead of graphics mode.

   This setting may be benefitial to some older tools that require text output. By default all the tools are launched in graphics mode. Read more about text modes in Output Properties section below.

- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID
  Hardware BoardProduct (e.g. Mac-35C1E88140C3E6CF). Not present on real Macs, but used to avoid extra parsing of SMBIOS tables, especially in boot.efi.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB
  Hardware BoardSerialNumber. Override for MLB. Present on newer Macs (2013+ at least).
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM
  Hardware ROM. Override for ROM. Present on newer Macs (2013+ at least).
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:prev-lang:kbd
  ASCII string defining default keyboard layout. Format is lang-COUNTRY:keyboard, e.g. ru-RU:252 for Russian locale and ABC keyboard. Also accepts short forms: ru:252 or ru:0 (U.S. keyboard, compatible with 10.9). Full decoded keyboard list from AppleKeyboardLayouts-L.dat can be found here. Using non-latin keyboard on 10.14 will not enable ABC keyboard, unlike previous and subsequent macOS versions, and is thus not recommended in case 10.14 is needed.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:security-mode
  ASCII string defining FireWire security mode. Legacy, can be found in IOFireWireFamily source code in IOFireWireController.cpp. It is recommended not to set this variable, which may speedup system startup. Setting to full is equivalent to not setting the variable and none disables FireWire security.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale
  One-byte data defining boot.efi user interface scaling. Should be **01** for normal screens and **02** for HiDPI screens.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:DefaultBackgroundColor
  Four-byte BGRA data defining boot.efi user interface background colour. Standard colours include **BF BF BF 00** (Light Gray) and **00 00 00 00** (Syrah Black). Other colours may be set at user's preference.

9.5 Other Variables

The following variables may be useful for certain configurations or troubleshooting:

- 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args
  Kernel arguments, used to pass configuration to Apple kernel and drivers. There are many arguments, which may be found by looking for the use of PE_parse_boot_argn function in the kernel or driver code. Some of the known boot arguments include:
  - acpi_layer=0xFFFFFFFF
  - acpi_level=0xFFFF5F (implies ACPI_ALL_COMPONENTS)
  - arch=i386 (force kernel architecture to i386, see KernelArch)
  - batman=VALUE (AppleSmartBatteryManager debug mask)
  - batman-nosmc=1 (disable AppleSmartBatteryManager SMC interface)
  - cpus=VALUE (maximum number of CPUs used)
  - debug=VALUE (debug mask)
  - io=VALUE (IOKit debug mask)
  - keepsyms=1 (show panic log debug symbols)
  - kextlog=VALUE (kernel extension loading debug mask)
  - nvram-log=1 (enables AppleEFINVRAM logs)
  - nv_disable=1 (disables NVIDIA GPU acceleration)
  - nvda_drv=1 (legacy way to enable NVIDIA web driver, removed in 10.12)
  - npci=0x2000 (legacy, disables kIOPCIConfiguratorPFM64)
  - lapic_dont_panic=1
  - slide=VALUE (manually set KASLR slide)
  - smcdebug=VALUE (AppleSMC debug mask)
  - -amd_no_dgpu_accel (alternative to WhateverGreen's -radvesa for new GPUs)
  - -nehalem_error_disable
  - -no_compat_check (disable model checking on 10.7+)
  - -s (single mode)
  - -v (verbose mode)
  - -x (safe mode)

  There are multiple external places summarising macOS argument lists: example 1, example 2.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg
  Booter arguments, similar to boot-args but for boot.efi. Accepts a set of arguments, which are hexadecimal

11.8 Audio Properties

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

   Normally this contains 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 string
   Description: Device path of the specified audio controller for audio support.

   Normally this 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 empty device path will result in the first available audio controller to be used.

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

   Normally this contains the index of the green out of the builtin analog audio controller (`HDEF`). The number of output nodes (`N`) 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)
   ```

   The quickest way to find the right port is to bruteforce the values from `0` to `N - 1`.

4. `AudioSupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Activate audio support by connecting to a backend driver.

   Enabling this setting routes audio playback from builtin protocols to a dedicated audio port (`AudioOut`) of the specified codec (`AudioCodec`) located on the audio controller (`AudioDevice`).

5. `MinimumVolume`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal heard volume level from `0` to `100`.

   Screen reader will use this volume level, when the calculated volume level is less than `MinimumVolume`. Boot chime sound will not play if the calculated volume level is less than `MinimumVolume`.

6. `PlayChime`
   Type: `plist ~~boolean~~string`
   Failsafe: ~~false~~empty string
   Description: Play chime sound at startup.

Enabling this setting plays boot chime through builtin audio support. Volume level is determined by `MinimumVolume` and `VolumeAmplifier` settings and `SystemAudioVolume` NVRAM variable. Possible values include:

- `Auto` — Enables chime when `StartupMute` NVRAM variable is not present or set to `00`.
- `Enabled` — Enables chime unconditionally.
- `Disabled` — Disables chime unconditionally.

Note: ~~this setting is~~ `Enabled` can be used in separate from `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play boot chime.

7. `VolumeAmplifier`
   Type: `plist integer`
   Failsafe: `0`
   Description: Multiplication coefficient for system volume to raw volume linear translation from `0` to `1000`.

   Volume level range read from `SystemAudioVolume` varies depending on the codec. To transform read value in `[0, 127]` range into raw volume range `[0, 100]` the read value is scaled to `VolumeAmplifier` percents:

$$RawVolume = MIN(\frac{SystemAudioVolume * VolumeAmplifier}{100}, 100)$$

   Note: the transformation used in macOS is not linear, but it is very close and this nuance is thus ignored.

11.9 Input Properties

1. `KeyFiltering`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable keyboard input sanity checking.

   Apparently some boards such as the GA Z77P-D3 may return uninitialised data in `EFI_INPUT_KEY` with all input protocols. This option discards keys that are neither ASCII, nor are defined in the UEFI specification (see tables 107 and 108 in version 2.8).

2. `KeyForgetThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Remove key unless it was submitted during this timeout in milliseconds.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers only report key presses as interrupts and pressing and holding the key on the keyboard results in subsequent submissions of this key with some defined time interval. As a result we use a timeout to remove once pressed keys from the buffer once the timeout expires and no new submission of this key happened.

   This option allows to set this timeout based on the platform. The recommended value that works on the majority of the platforms is `5` milliseconds. For reference, holding one key on VMware will repeat it roughly every `2` milliseconds and the same value for APTIO V is `3-4` milliseconds. Thus it is possible to set a slightly lower value on faster platforms and slightly higher value on slower platforms for more responsive input.

   Note: Some platforms may require different values, higher or lower. For example, when detecting key misses in OpenCanopy try increasing this value (e.g. to `10`), and when detecting key stall, try decreasing this value. Since every platform is different it may be reasonable to check every value from `1` to `25`.

3. `KeyMergeThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

   Similarly to `KeyForgetThreshold`, this option works around the sequential nature of key submission. To be able to recognise simultaneously pressed keys in the situation when all keys arrive sequentially, we are required to set a timeout within which we assume the keys were pressed together.

   Holding multiple keys results in reports every `2` and `1` milliseconds for VMware and APTIO V respectively. Pressing keys one after the other results in delays of at least `6` and `10` milliseconds for the same platforms. The