OpenCore

Reference Manual (0.7.3.4)

[2021.10.03]

figure. Available entries include:

- `BOOTx64.efi` or `BOOTIa32.efi`
  Initial bootstrap loaders, which load `OpenCore.efi`. `BOOTx64.efi` is loaded by the firmware by default consistent with the UEFI specification. However, it may also be renamed and put in a custom location to allow OpenCore coexist alongside operating systems, such as Windows, that use `BOOTx64.efi` files as their loaders. Refer to the `LauncherOption` property for details.
- `boot`
  Duet bootstrap loader, which initialises the UEFI environment on legacy BIOS firmware and loads `OpenCore.efi` similarly to other bootstrap loaders. A modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- `ACPI`
  Directory used for storing supplemental ACPI information for the `ACPI` section.
- `Drivers`
  Directory used for storing supplemental `UEFI` drivers for `UEFI` section.
- `Kexts`
  Directory used for storing supplemental kernel information for the `Kernel` section.
- `Resources`
  Directory used for storing media resources such as audio files for screen reader support. Refer to the `UEFI Audio Properties` section for details. This directory also contains image files for graphical user interface. Refer to the OpenCanopy section for details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter application responsible for operating system loading. The directory `OpenCore.efi` resides in is called the `root directory`, which is set to `EFI\OC` by default. When launching `OpenCore.efi` directly or through a custom launcher however, other directories containing `OpenCore.efi` files are also 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 the `0`-terminator) will be accessible within OpenCore.

3.2 Installation and Upgrade

To install OpenCore, replicate the Configuration Structure described in the previous section in the EFI volume of a GPT partition. While corresponding sections of this document 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 the OpenCore repository. Vaulting information is provided in the Security Properties section of this document.

The `OC config` file, as with any property list file, can be edited with any text editor, such as nano or vim. However, specialised software may provide a better experience. On macOS, the preferred GUI application is Xcode. ~~For a lightweight~~The ProperTree editor is a lightweight, cross-platform and open-source alternative~~, the ProperTree editor can be utilised~~.

It is strongly ~~advised not to use any software that is~~ recommended to avoid configuration creation tools that are aware of the internal ~~configration structure as it constantly gets out of date and will cause incorrect configuration to be~~

5 Booter

5.1 Introduction

This section allows the application of different types of UEFI modifications to operating system bootloaders, primarily the Apple bootloader (`boot.efi`). The modifications currently provide various patches and environment alterations for different firmware types. Some of these features were originally implemented as part of AptioMemoryFix.efi, which is no longer maintained. Refer to the Tips and Tricks section for instructions on migration.

If this is used for the first time on customised firmware, the following requirements should be met before starting:

- 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 the ASUS WS-X299-PRO, this option results in 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. Refer to the ControlMsrE2 notes for 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 (which appear to sometimes cause wake to black screen or boot loop issues on older platforms) may be temporarily disabled. The specific issues may vary, but ACPI tables should typically be looked at first.

Here is an example of a defect found on 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 be reset by hardware changes and in certain other circumstances. To view their current state, use the `pmset -g` command in Terminal.

5.2 Properties

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

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

   To be filled with `plist dictionary` values, describing each patch. Refer to the Patch Properties section below for details.

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

Note: The need for this quirk is determined by early boot failures.

18. `SignalAppleOS`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Report macOS being loaded through OS Info for any OS.

    This quirk is useful on Mac firmware, which loads different operating systems with different hardware configurations. For example, it is supposed to enable Intel GPU in Windows and Linux in some dual-GPU MacBook models.

19. `SyncRuntimePermissions`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Update memory permissions for the runtime environment.

    Some types of firmware fail to properly handle runtime permissions:

    - They incorrectly mark `OpenRuntime` as not executable in the memory map.
    - They incorrectly mark `OpenRuntime` as not executable in the memory attributes table.
    - They lose entries from the memory attributes table after `OpenRuntime` is loaded.
    - They mark items in the memory attributes table as read-write-execute.

    This quirk attempts to update the memory map and memory attributes table to correct this.

    Note: The need for this quirk is indicated by early boot failures —(e.g. halts at black screen), particularly in early boot of the Linux kernel. Only firmware released after 2017 is typically affected.

- Options will be listed in file system handle firmware order to maintain an established order across reboots regardless of the operating system chosen for loading.
- Custom entries, tools, and system entries will be added after all other options.
- Auxiliary options will only be displayed upon entering "Extended Mode" in the OpenCore picker (typically by pressing the `Space` key).

The boot process is as follows:

- Look up the first valid primary option in the `BootNext` UEFI variable.
- On failure, look up the first valid primary option in the `BootOrder` UEFI variable.
- Mark the option as the default option to boot.
- Boot option through the picker or without it depending on the `ShowPicker` option.
- Show picker on failure otherwise.

Note 1: This process will only work reliably when the `RequestBootVarRouting` option is enabled or the firmware does not control UEFI boot options (`OpenDuetPkg` or custom BDS). When `LauncherOption` is not enabled, other operating systems may overwrite OpenCore settings and this property should therefore be enabled when planning to use other operating systems.

Note 2: UEFI variable boot options boot arguments will be removed, if present, as they may contain arguments that can compromise the operating system, which is undesirable when secure boot is enabled.

Note 3: Some operating systems, such as Windows, may create a boot option and mark it as the topmost option upon first boot or after NVRAM resets from within OpenCore. When this happens, the default boot entry choice will remain changed until the next manual reconfiguration.

8.2 Properties

1. `Boot`
   Type: `plist dict`
   Description: Apply the boot configuration described in the Boot Properties section below.

2. `BlessOverride`
   Type: `plist array`
   Failsafe: Empty
   Description: Add custom scanning paths through the bless model.

   To be filled with `plist string` entries containing absolute UEFI paths to customised bootloaders such as `\EFI\debian\grubx64.efi` for the Debian bootloader. This allows non-standard boot paths to be automatically discovered by the OpenCore picker. Designwise, they are equivalent to predefined blessed paths, such as `\System\Library\CoreServices\boot.efi` or `\EFI\Microsoft\Boot\bootmgfw.efi`, but unlike predefined bless paths, they have the highest priority.

3. `Debug`
   Type: `plist dict`
   Description: Apply debug configuration described in the Debug Properties section below.

4. `Entries`
   Type: `plist array`
   Failsafe: Empty
   Description: Add boot entries to OpenCore picker.

   To be filled with `plist dict` values, describing each load entry. Refer to the Entry Properties section below for details.

5. `Security`
   Type: `plist dict`
   Description: Apply the security configuration described in the Security Properties section below.

6. `Tools`
   Type: `plist array`
   Failsafe: Empty
   Description: Add tool entries to the OpenCore picker.

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`).

   Note: Certain operating systems, such as macOS Big Sur, are incapable of disabling firmware updates by using the `run-efi-updater` NVRAM variable.

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. The `Disabled` policy will still let the macOS Recovery load in most cases as typically, there 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 the design of Apple Secure Boot, the `Signed` policy will let any Apple-signed macOS Recovery load regardless of the Apple Secure Boot state, which may not always be desired. While using signed DMG images is more desirable, verifying the image signature may slightly slow the boot time down (by up to 1 second).
   - `Any` — any DMG images will mount as normal filesystems. The `Any` policy is strongly discouraged and will result in boot failures when Apple Secure Boot is active.

8. `EnablePassword`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable password protection to facilitate 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 still under development and is not ready for production environments.

9. `ExposeSensitiveData`
   Type: `plist integer`
   Failsafe: `0x6`
   Description: Sensitive data exposure bitmask (sum) to operating system.

   - `0x01` — Expose the printable booter path as an UEFI variable.
   - `0x02` — Expose the OpenCore version as an UEFI variable.
   - `0x04` — Expose the OpenCore version in the OpenCore picker menu title.
   - `0x08` — Expose OEM information as a set of UEFI variables.

   The exposed booter path points to OpenCore.efi or its booter depending on the load order. To obtain the booter path, use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path
```

   To use a booter path to mount a booter volume, use the following command in macOS:

```
u=$(nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path | sed 's/.*GPT,\([^,]*\),.*/\1/'); \
  if [ "$u" != "" ]; then sudo diskutil mount $u ; fi
```

   To obtain the current OpenCore version, use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:opencore-version
```

```
/path/to/RsaTool -sign vault.plist vault.sig vault.pub
off=$(($(strings -a -t d OpenCore.efi | grep "=BEGIN OC VAULT=" | cut -f1 -d' ')+16))
dd of=OpenCore.efi if=vault.pub bs=1 seek=$off count=528 conv=notrunc
rm vault.pub
```

Note 1: While it may appear obvious, an external method is required to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this, it is recommended to enable UEFI SecureBoot using a custom certificate and to sign `OpenCore.efi` and `BOOTx64.efi` with a custom key. More details on customising secure boot on modern firmware can be found in the Taming UEFI SecureBoot paper (in Russian).

Note 2: ~~`vault.plist` and `vault.sig` are used regardless~~ Regardless of this option~~when~~, `vault.plist` is ~~present or~~ always used when present, and both `vault.plist` and `vault.sig` are used and required when a public key is embedded into `OpenCore.efi`~~. Setting this option will only ensure configuration sanity, and~~, and errors will abort the boot process ~~otherwise.~~ in either case. Setting this option allows OpenCore to warn the user if the configuration is not as required to achieve an expected higher security level.

14. `ScanPolicy`
Type: `plist integer`, 32 bit
Failsafe: `0x10F0103`
Description: Define operating system detection policy.

This value allows preventing scanning (and booting) untrusted sources based on a bitmask (sum) of a set of flags. As it is not possible to reliably detect every file system or device type, this feature cannot be fully relied upon in open environments, and additional measures are to be applied.

Third party drivers may introduce additional security (and performance) consideratons following the provided scan policy. The active Scan policy is exposed in the `scan-policy` variable of `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` GUID for UEFI Boot Services only.

- `0x00000001` (bit `0`) — `OC_SCAN_FILE_SYSTEM_LOCK`, restricts scanning to only known file systems defined as a part of this policy. File system drivers may not be aware of this policy. Hence, to avoid mounting of undesired file systems, drivers for such file systems should not be loaded. This bit does not affect DMG mounting, which may have any file system. Known file systems are prefixed with `OC_SCAN_ALLOW_FS_`.
- `0x00000002` (bit `1`) — `OC_SCAN_DEVICE_LOCK`, restricts scanning to only known device types defined as a part of this policy. It is not always possible to detect protocol tunneling, so be aware that on some systems, it may be possible for e.g. USB HDDs to be recognised as SATA instead. Cases like this must be reported. Known device types are prefixed with `OC_SCAN_ALLOW_DEVICE_`.
- `0x00000100` (bit `8`) — `OC_SCAN_ALLOW_FS_APFS`, allows scanning of APFS file system.
- `0x00000200` (bit `9`) — `OC_SCAN_ALLOW_FS_HFS`, allows scanning of HFS file system.
- `0x00000400` (bit `10`) — `OC_SCAN_ALLOW_FS_ESP`, allows scanning of EFI System Partition file system.
- `0x00000800` (bit `11`) — `OC_SCAN_ALLOW_FS_NTFS`, allows scanning of NTFS (Msft Basic Data) file system.
- `0x00001000` (bit `12`) — `OC_SCAN_ALLOW_FS_LINUX_ROOT`, allows scanning of Linux Root file systems.
- `0x00002000` (bit `13`) — `OC_SCAN_ALLOW_FS_LINUX_DATA`, allows scanning of Linux Data file systems.
- `0x00004000` (bit `14`) — `OC_SCAN_ALLOW_FS_XBOOTLDR`, allows scanning the Extended Boot Loader Partition as defined by the Boot Loader Specification.
- `0x00010000` (bit `16`) — `OC_SCAN_ALLOW_DEVICE_SATA`, allow scanning SATA devices.
- `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, a value of `0xF0103` is expected to do the following:

- Permit scanning SATA, SAS, SCSI, and NVMe devices with APFS file systems.
- Prevent scanning any devices with HFS or FAT32 file systems.
- Prevent 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 x86legacy~~Matching model for current SMBIOS.
- 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~~ Prior to macOS 12 PlatformInfo and SecureBootModel ~~are independent,~~ were independent, allowing Apple Secure Boot can be used with any SMBIOS with and without T2. Starting with macOS 12 SecureBootModel must match the SMBIOS Mac model. Default model derives the model based on SMBIOS board identifier, either set automatically via the Generic section or set manually via the SMBIOS section. If there is no board identifier override the model will be derived heuristically from OEM SMBIOS.

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.

Third-party scripts may be used to create `nvram.plist` file. An example of such script can be found in `Utilities`. The use of third-party scripts may require `ExposeSensitiveData` set to `0x3` to provide `boot-path` variable with the OpenCore EFI partition UUID.

Warning: This feature can be dangerous, as it passes unprotected data to firmware variable services. Only use when no hardware NVRAM implementation is provided by the firmware or when the NVRAM implementation is incompatible.

4. `LegacyOverwrite`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Permits overwriting firmware variables from `nvram.plist`.

   Note: Only variables accessible from the operating system will be overwritten.

5. `LegacySchema`
   Type: `plist dict`
   Description: Allows setting certain NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

   `*` value can be used to accept all variables for certain GUID.

   WARNING: Choose variables carefully, as the nvram.plist file is not vaulted. For instance, do not include `boot-args` or `csr-active-config`, as these can be used to bypass SIP.

6. `WriteFlash`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enables writing to flash memory for all added variables.

   Note: This value should be enabled on most types of firmware but is left configurable to account for firmware that may have issues with NVRAM variable storage garbage collection or similar.

The `nvram` command can be used to read NVRAM variable values from macOS by concatenating the GUID and name variables separated by a `:` symbol. For example, `nvram 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args`.

A continuously updated variable list can be found in a corresponding document: NVRAM Variables.

9.3 Mandatory Variables

Warning: These variables may be added by the PlatformNVRAM or Generic subsections of the PlatformInfo section. Using `PlatformInfo` is the recommended way of setting these variables.

The following variables are mandatory for macOS functioning:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeatures`
  32-bit `FirmwareFeatures`. Present on all Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeaturesMask`
  32-bit `FirmwareFeaturesMask`. Present on all Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB`
  `BoardSerialNumber`. Present on newer Macs (2013+ at least) to avoid extra parsing of SMBIOS tables, especially in `boot.efi`.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`
  Primary network adapter MAC address or replacement value. Present on newer Macs (2013+ at least) to avoid accessing special memory region, especially in `boot.efi`.

9.4 Recommended Variables

The following variables are recommended for faster startup or other improvements:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:BridgeOSHardwareModel`
  Bridge OS hardware model variable used to propagate to IODT bridge-model by `EfiBoot`. Read by `hw.target` `sysctl`, used by SoftwareUpdateCoreSupport.

Description: Update Data Hub fields in non-`Automatic` mode.

Note: This section is ignored and may be removed when `Automatic` is `true`.

10. `Memory`
Type: `plist dictionary`
Description: Define custom memory configuration.

Note: This section is ignored and may be removed when `CustomMemory` is `false`.

11. `PlatformNVRAM`
Type: `plist dictionary`
Description: Update platform NVRAM fields in non-`Automatic` mode.

Note: This section is ignored and may be removed when `Automatic` is `true`.

12. `SMBIOS`
Type: `plist dictionary`
Description: Update SMBIOS fields in non-`Automatic` mode.

Note: This section is ignored and may be removed when `Automatic` is `true`.

10.2 Generic Properties

1. `SpoofVendor`
Type: `plist boolean`
Failsafe: `false`
Description: Sets SMBIOS vendor fields to `Acidanthera`.

It can be dangerous to use "Apple" in SMBIOS vendor fields for reasons outlined in the `SystemManufacturer` description. However, certain firmware may not provide valid values otherwise, which could obstruct the operation of some software.

2. `AdviseFeatures`
Type: `plist boolean`
Failsafe: `false`
Description: Updates `FirmwareFeatures` with supported bits.

Added bits to `FirmwareFeatures`:

- `FW_FEATURE_SUPPORTS_CSM_LEGACY_MODE` (`0x1`) - Without this bit, it is not possible to reboot to Windows installed on a drive with an EFI partition that is not the first partition on the disk.
- `FW_FEATURE_SUPPORTS_UEFI_WINDOWS_BOOT` (`0x20000000`) - Without this bit, it is not possible to reboot to Windows installed on a drive with an EFI partition that is the first partition on the disk.
- `FW_FEATURE_SUPPORTS_APFS` (`0x00080000`) - Without this bit, it is not possible to install macOS on an APFS disk.
- `FW_FEATURE_SUPPORTS_LARGE_BASESYSTEM` (`0x800000000`) - Without this bit, it is not possible to install macOS versions with large BaseSystem images, such as macOS 12.

Note: On most newer firmwares these bits are already set, the option may be necessary when "upgrading" the firmware with new features.

3. `MaxBIOSVersion`
Type: `plist boolean`
Failsafe: `false`
Description: Sets `BIOSVersion` to `9999.999.999.999.999`, recommended for legacy Macs when using `Automatic` PlatformInfo, to avoid BIOS updates in unofficially supported macOS versions.

4. `SystemMemoryStatus`
Type: `plist string`
Failsafe: `Auto`
Description: Indicates whether system memory is upgradable in `PlatformFeature`. This controls the visibility of the Memory tab in "About This Mac".

work in runtime, i.e. during operating system functioning. Feature highlights:

- NVRAM namespaces, allowing to isolate operating systems from accessing select variables (e.g. `RequestBootVarRouting` or `ProtectSecureBoot`).
- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, such as VirtualSMC, which implements `AuthRestart` support.
- NVRAM isolation, allowing to protect all variables from being written from an untrusted operating system (e.g. `DisableVariableWrite`).
- UEFI Runtime Services memory protection management to workaround read-only mapping (e.g. `EnableWriteUnprotector`).

11.6 OpenLinuxBoot

`OpenLinuxBoot` is an OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL`. It detects and boots Linux distros which are installed according to the Boot Loader Specification or to the closely related (but not identical, see next paragraph) systemd BootLoaderSpecByDefault. In effect this means Linux distributions where the available boot options are found in `{ESP}/loader/entries/*.conf` files (for instance `/boot/efi/loader/entries/*.conf`) or in `{boot}/loader/entries/*.conf` files (for instance `/boot/loader/entries/*.conf`). The former layout – pure Boot Loader Specification, using kernel files on the EFI System Partition or Extended Boot Loader Partition – is specific to systemd-boot, the latter layout with kernel files typically on the partition which will be mounted as `/boot` applies to most Fedora-related distros including Fedora itself, RHEL and variants.

BootLoaderSpecByDefault includes the possibility of expanding GRUB variables in its `*.conf` files – and this is used in practice in certain distros such as CentOS. In order to correctly handle this, `OpenLinuxBoot` extracts all variables from `{boot}/grub2/grubenv` and any unconditionally set variables from `{boot}/grub2/grub.cfg`. This has proved sufficient in practice to extract the required variables seen so far in distros which use this GRUB-specific feature.

For distributions which do not use either of the above schemes, `OpenLinuxBoot` will autodetect and boot `{boot}/vmlinuz*` kernel files directly, after linking these automatically – based on the kernel version in the filename – to their associated `{boot}/init*` ramdisk files, and after searching in `/etc/default/grub` for kernel boot options and `/etc/os-release` for the distro name. This layout applies to most Debian-related distros, including Debian itself, Ubuntu and variants.

The method of starting the kernel relies on it being compiled with EFISTUB, however this applies to almost all modern distros, particularly those which use systemd. Most modern distros use systemd as their system manager (even though at the same time most do *not* use systemd-boot as their bootloader).

The latest kernel version of a given install is always shown in the boot menu. Additional versions, recovery versions, etc. are added as auxiliary boot entries, so depending on OpenCore's `HideAuxiliary` setting may not be shown until the space key is pressed.

Note 1: `OpenLinuxBoot` requires filesystem drivers that may not be available in firmware such as EXT4 and BTRFS drivers. These drivers can be obtained from external sources. Drivers tested in basic scenarios can be downloaded from OcBinaryData. Be aware that these drivers are neither tested for reliability in all scenarious, nor underwent any tamper-resistance testing, therefore have may carry potential security or data-loss risks.

Most Linux distributions keep their boot files on ~~the~~ an EXT4 ~~file system~~ partition even when the distribution's ~~main~~ root filesystem is something else, such as BTRFS, therefore ~~a suitable UEFI~~ only an EXT4 ~~file system~~ driver such as `ext4_x64` is normally required. A BTRFS driver such as `btrfs_x64` will be required in ~~a~~ the currently somewhat less standard ~~setup~~ situation where the boot files are on a BTRFS partition, e.g. as is currently done by default in openSUSE.

Pure Boot Loader Spec (e.g. as implemented by systemd-boot) keeps all kernel and ramdisk images directly on the EFI System Partition (or an Extended Boot Loader Partition), therefore it requires no additional filesystem driver - but it is not widely used except in Arch Linux.

Note 2: `OpenLinuxBoot` does not attempt to read and interpret the layout of Linux installation media (which can be highly variable). Installation media should be booted directly either from the machine's own EFI boot menu or from the OpenCore boot menu. In some cases, e.g. Apple T2 hardware, then – depending on OpenCore's security settings – OpenCore may be able to start some Linux installers which the machine's own bootloader will refuse to boot.

Note 3: systemd-boot users (probably almost exclusively Arch Linux users) should be aware that `OpenLinuxBoot` does not support the systemd-boot–specific Boot Loader Interface; therefore use `efibootmgr` rather than `bootctl` for any

low-level Linux command line interaction with the boot menu.

Note 4: Be aware of the `SyncRuntimePermissions` quirk, which may need to be set to avoid early boot failure (i.e. halts with black screen) of the Linux kernel due to a firmware bug of some firmware released after 2017.

The default parameter values should work well, but if you need to parameterise this driver the following options may be specified in `UEFI/Drivers/Arguments`:

- `flags` - Default: all flags except `LINUX_BOOT_ADD_DEBUG_INFO` are set.

  Available flags are:

  - `0x00000001` (bit `0`) — `LINUX_BOOT_SCAN_ESP`, Allows scanning for entries on EFI System Partition.
  - `0x00000002` (bit `1`) — `LINUX_BOOT_SCAN_XBOOTLDR`, Allows scanning for entries on Extended Boot Loader Partition.
  - `0x00000004` (bit `2`) — `LINUX_BOOT_SCAN_LINUX_ROOT`, Allows scanning for entries on Linux Root filesystems.
  - `0x00000008` (bit `3`) — `LINUX_BOOT_SCAN_LINUX_DATA`, Allows scanning for entries on Linux Data filesystems.
  - `0x00000080` (bit `7`) — `LINUX_BOOT_SCAN_OTHER`, Allows scanning for entries on file systems not matched by any of the above.

    The following notes apply to all of the above options:

    Note 1: Apple filesystems APFS and HFS are never scanned.

    Note 2: Regardless of the above flags, a file system must first be allowed by `Misc/Security/ScanPolicy` before it can be seen by `OpenLinuxBoot` or any other `OC_BOOT_ENTRY_PROTOCOL` driver.

    Note 3: It is recommended to enable scanning `LINUX_ROOT` and `LINUX_DATA` in both `OpenLinuxBoot` flags and `Misc/Security/ScanPolicy` in order to be sure to detect all valid Linux installs.

  - `0x00000100` (bit `8`) — `LINUX_BOOT_ALLOW_AUTODETECT`, If set allows autodetecting and linking `vmlinuz*` and `init*` ramdisk files when `loader/entries` files are not found.
  - `0x00000200` (bit `9`) — `LINUX_BOOT_USE_LATEST`, When a Linux entry generated by `OpenLinuxBoot` is selected as the default boot entry in OpenCore, automatically switch to the latest kernel when a new version is installed.

    When this option is set, an internal menu entry id is shared between kernel versions from the same install of Linux. Linux boot options are always sorted highest kernel version first, so this means that the latest kernel version of the same install always shows as the default, with this option set.

    Note: This option is recommended on all systems.

  - `0x00000400` (bit `10`) — `LINUX_BOOT_ADD_RO`, This option applies to autodetected Linux only (i.e. to Debian-style distrubutions, not to BLSpec and Fedora-style distributions with `/loader/entries/*.conf` files). Some distrubtions run a filesystem check on loading which requires the root filesystem to initially be mounted read-only via the `ro` kernel option. Set this bit to add this option on autodetected distros; should be harmless but very slightly slow down boot time (due to requried remount as read-write) on distros which do not require it. To specify this option for specific distros only, use `partuuidopts:{partuuid}+=ro` instead of this flag.
  - `0x00004000` (bit `14`) — `LINUX_BOOT_LOG_VERBOSE`, Add additional debug log info about files encountered and autodetect options added while scanning for Linux boot entries.
  - `0x00008000` (bit `15`) — `LINUX_BOOT_ADD_DEBUG_INFO`, Adds a human readable file system type, followed by the first eight characters of the partition's unique partition uuid, to each generated entry name. Can help with debugging the origin of entries generated by the driver when there are multiple Linux installs on one system.

  Flag values can be specified in hexadecimal beginning with `0x` or in decimal, e.g. `flags=0x80` or `flags=128`.

- `partuuidopts:{partuuid}[+]="{options}"` - Default: not set.

  Allows specifying kernel options for a given partition only. If specified with `+=` then these are used in addition to autodetected options, if specified with `=` they are used instead. Used for autodetected Linux only. Values specified here are never used for entries created from `/loader/entries/*.conf` files.

  Note: The `partuuid` value to be specified here is typically the same as the `PARTUUID` seen in `root=PARTUUID=...` in the Linux kernel boot options (view using `cat /proc/cmdline`) for autodetected Debian-style distros, but is NOT the same for Fedora-style distros booted from `/loader/entries/*.conf` files.

Failsafe: `false`
Description: Perform UEFI controller connection after driver loading.

This option is useful for loading drivers following UEFI driver model as they may not start by themselves. Examples of such drivers are filesystem or audio drivers. While effective, this option may not be necessary for drivers performing automatic connection, and may slightly slowdown the boot.

Note: Some types of firmware, particularly those made by Apple, only connect the boot drive to speed up the boot process. Enable this option to be able to see all the boot options when running multiple drives.

4. `Drivers`
Type: `plist ~~dict~~array`
Failsafe: ~~None~~Empty
Description: Load selected drivers from `OC/Drivers` directory~~using the settings specified in the~~ .

To be filled with `plist dict` values, describing each driver. Refer to the Drivers Properties section below.

5. `Input`
Type: `plist dict`
Failsafe: None
Description: Apply individual settings designed for input (keyboard and mouse) in the Input Properties section below.

6. `Output`
Type: `plist dict`
Failsafe: None
Description: Apply individual settings designed for output (text and graphics) in the Output Properties section below.

7. `ProtocolOverrides`
Type: `plist dict`
Failsafe: None
Description: Force builtin versions of certain protocols described in the ProtocolOverrides Properties section below.

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`
Failsafe: Empty
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.

11.8 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`

Enabling this setting plays the boot chime using the builtin audio support. The volume level is determined by the `MinimumVolume` and `VolumeAmplifier` settings as well as the `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: `Enabled` can be used in separate from `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play the boot chime.

7. `ResetTrafficClass`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set HDA Traffic Class Select Register to `TC0`.

   AppleHDA kext will function correctly only if `TCSEL` register is configured to use `TC0` traffic class. Refer to Intel I/O Controller Hub 9 (ICH9) Family Datasheet (or any other ICH datasheet) for more details about this register.

   Note: This option is independent from `AudioSupport`. If AppleALC is used it is preferred to use AppleALC `alctsel` property instead.

8. `SetupDelay`
   Type: `plist integer`
   Failsafe: `0`
   Description: Audio codec reconfiguration delay in microseconds.

   Some codecs require a vendor-specific delay after the reconfiguration (e.g. volume setting). This option makes it configurable. A typical delay can be up to 0.5 seconds.

9. `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.

11.11 Drivers Properties

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

2. `Path`
   Type: `plist string`
   Failsafe: Empty
   Description: Path of file to be loaded as a UEFI driver from `OC/Drivers` directory.

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: If `false` this driver entry will be ignored.

4. `Arguments`
   Type: `plist string`
   Failsafe: Empty
   Description: Some OC plugins accept optional additional arguments which may be specified as a string here.