OpenCore

Reference Manual (0.8.2.3)

[2022.07.28]

3 Setup

3.1 Directory Structure

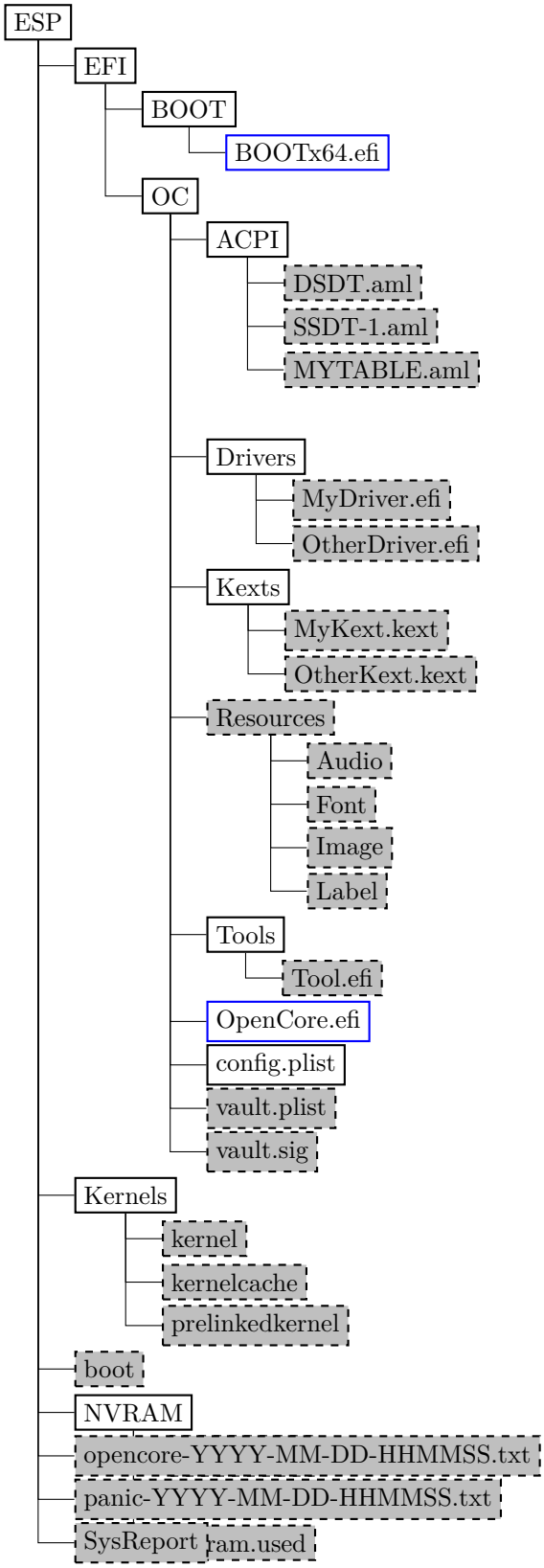

Figure 1. Directory Structure

When directory boot is used, the directory structure used should follow the descriptions in the Directory Structure

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.
- `nvram.fallback`
  OpenCore variable import fallback file.
- `nvram.used`
  Renamed previous OpenCore variable import file after switch to fallback 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. The

4 ACPI

4.1 Introduction

ACPI (Advanced Configuration and Power Interface) is an open standard to discover and configure computer hardware. The ACPI specification defines standard tables (e.g. `DSDT`, `SSDT`, `FACS`, `DMAR`) and various methods (e.g. `_DSM`, `_PRW`) for implementation. Modern hardware needs few changes to maintain ACPI compatibility and some options for such changes are provided as part of OpenCore.

To compile and disassemble ACPI tables, the iASL compiler developed by ACPICA can be used. A GUI front-end to iASL compiler can be downloaded from Acidanthera/MaciASL.

ACPI changes apply globally (to every operating system) with the following effective order:

- ~~Patch~~Delete is processed.
- ~~Delete~~Quirks ~~is~~ are processed.
- ~~Add~~Patch is processed.
- ~~Quirks~~Add ~~are~~ is processed.

Applying the changes globally resolves the problems of incorrect operating system detection (consistent with the ACPI specification, not possible before the operating system boots), operating system chainloading, and difficult ACPI debugging. Hence, more attention may be required when writing changes to `_OSI`.

Applying the patches early makes it possible to write so called "proxy" patches, where the original method is patched in the original table and is implemented in the patched table.

There are several sources of ACPI tables and workarounds. Commonly used ACPI tables are provided with OpenCore, VirtualSMC, VoodooPS2, and WhateverGreen releases. Besides those, several third-party instructions may be found on the AppleLife Laboratory and DSDT subforums (e.g. Battery register splitting guide). A slightly more user-friendly explanation of some tables included with OpenCore can also be found in Dortania's Getting started with ACPI guide. For more exotic cases, there are several alternatives such as daliansky's ACPI sample collection (English Translation by 5T33Z0 et al). Please note however, that suggested solutions from third parties may be outdated or may contain errors.

4.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected tables from the `OC/ACPI` directory.

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

2. `Delete`
   Type: `plist array`
   Failsafe: Empty
   Description: Remove selected tables from the ACPI stack.

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

3. `Patch`
   Type: `plist array`
   Failsafe: Empty
   Description: Perform binary patches in ACPI tables before table addition or removal.

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

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

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.

Booter changes apply with the following effective order:

- `Quirks` are processed.
- `Patch` is processed.

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.

7 Kernel

7.1 Introduction

This section allows the application of different kinds of kernelspace modifications on Apple Kernel (XNU). The modifications currently provide driver (kext) injection, kernel and driver patching, and driver blocking.

Kernel and kext changes apply with the following effective order:

- `Block` is processed.
- `Emulate` and `Quirks` are processed.
- `Patch` is processed.
- `Add` and `Force` are processed.

7.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected kernel extensions (kexts) from the `OC/Kexts` directory.

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

   Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

   Note 2: To track the dependency order, inspect the `OSBundleLibraries` key in the `Info.plist` file of the kext being added. Any kext included under the key is a dependency that must appear before the kext being added.

   Note 3: Kexts may have inner kexts (`Plugins`) included in the bundle. Such `Plugins` must be added separately and follow the same global ordering rules as other kexts.

2. `Block`
   Type: `plist array`
   Failsafe: Empty
   Description: Remove selected kernel extensions (kexts) from the prelinked kernel.

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

3. `Emulate`
   Type: `plist dict`
   Description: Emulate certain hardware in kernelspace via parameters described in the Emulate Properties section below.

4. `Force`
   Type: `plist array`
   Failsafe: Empty
   Description: Load kernel extensions (kexts) from the system volume if they are not cached.

   To be filled with `plist dict` values, describing each kext. Refer to the Force Properties section below for details. This section resolves the problem of injecting kexts that depend on other kexts, which are not otherwise cached. The issue typically affects older operating systems, where various dependency kexts, such as `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default.

   Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

   Note 2: `Force` happens before `Add`.

   Note 3: The signature of the "forced" kext is not checked in any way. This makes using this feature extremely dangerous and undesirable for secure boot.

   Note 4: This feature may not work on encrypted partitions in newer operating systems.

(f) Download and run Modified GRUB Shell compiled by brainsucker or use a newer version by datasone.
(g) Enter `setup_var 0x123 0x00` command, where `0x123` should be replaced by the actual offset, and reboot.

Warning: Variable offsets are unique not only to each motherboard but even to its firmware version. Never ever try to use an offset without checking.

On selected platforms, the `ControlMsrE2` tool can also change such hidden options. Pass desired argument: `lock`, `unlock` for `CFG Lock`. Or pass `interactive` to find and modify other hidden options.

As a last resort, consider patching the BIOS (for advanced users only).

2. `AppleXcpmCfgLock`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.8 (not required for older)
Description: Disables `PKG_CST_CONFIG_CONTROL` (`0xE2`) MSR modification in XNU kernel, commonly causing early kernel panic, when it is locked from writing (XCPM power management).

Note: This option should be avoided whenever possible. Refer to the `AppleCpuPmCfgLock` description for details.

3. `AppleXcpmExtraMsrs`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.8 (not required for older)
Description: Disables multiple MSR access critical for certain CPUs, which have no native XCPM support.

This is typically used in conjunction with the `Emulate` section on Haswell-E, Broadwell-E, Skylake-SP, and similar CPUs. More details on the XCPM patches are outlined in acidanthera/bugtracker#365.

Note: Additional not provided patches will be required for Ivy Bridge or Pentium CPUs. It is recommended to use `AppleIntelCpuPowerManagement.kext` for the former.

4. `AppleXcpmForceBoost`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.8 (not required for older)
Description: Forces maximum performance in XCPM mode.

This patch writes `0xFF00` to `MSR_IA32_PERF_CONTROL` (`0x199`), effectively setting maximum multiplier for all the time.

Note: While this may increase the performance, this patch is strongly discouraged on all systems but those explicitly dedicated to scientific or media calculations. Only certain Xeon models typically benefit from the patch.

5. `CustomPciSerialDevice`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.7
Description: Performs change of PMIO register base address on a customised PCI serial device.

The patch changes the PMIO register base address that the XNU kernel uses for serial input and output, from that of the default built-in COM1 serial port `0x3F8`, to the base address stored in the first IO BAR of a specified PCI device or to a specific base address (e.g. `0x2F8` for COM2).

Note: By default, serial logging is disabled. `serial=3` boot argument, which enables serial input and output, should be used for XNU to print logs to the serial port.

Note 2: In addition to this patch, kext `Apple16X50PCI0` should be prevented from attaching to have `kprintf` method working properly. This can be achieved by ~~setting (i. e. `Delete`, then `Add`) the `class-code` property of the PCI serial port device to `FFFFFFFF` in `DeviceProperties` section. As an alternative solution, a codeless kext `PCIeSerialDisable.kext` shown in the spoiler `PCIeSerialDisable.kext/Contents/Info.plist` at acidanthera/bugtracker may also be used.~~ using PCIeSerialDisable. In addition, for certain Thunderbolt cards the IOKit personality `IOPCITunnelCompatible` also needs to be set to `true`, which can be done by the `PCIeSerialThunderboltEnable.kext` attached at acidanthera/bugtracker#2003.

Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

18. `PowerTimeoutKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.15 (not required for older)
    Description: Disables kernel panic on setPowerState timeout.

    An additional security measure was added to macOS Catalina (10.15) causing kernel panic on power change timeout for Apple drivers. Sometimes it may cause issues on misconfigured hardware, notably digital audio, which sometimes fails to wake up. For debug kernels `setpowerstate_panic=0` boot argument should be used, which is otherwise equivalent to this quirk.

19. `ProvideCurrentCpuInfo`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: ~~10.8~~ 10.4 (10.14)
    Description: Provides current CPU info to the kernel.

    This quirk works differently depending on the CPU:

    - For Microsoft Hyper-V it provides the correct TSC and FSB values to the kernel, as well as disables CPU topology validation (10.8+).
    - For KVM and other hypervisors it provides precomputed MSR 35h values solving kernel panic with `-cpu host`.
    - For Intel CPUs it adds support for asymmetrical SMP systems (e.g. Intel Alder Lake) by patching core count to thread count along with the supplemental required changes (10.14+). Cache size and cache line size are also provided when using 10.4 as Intel Penryn and newer may only have cache information in CPUID leaf 0x4 which is unsupported by 10.4.

20. `SetApfsTrimTimeout`
    Type: `plist integer`
    Failsafe: `-1`
    Requirement: 10.14 (not required for older)
    Description: Set trim timeout in microseconds for APFS filesystems on SSDs.

    The APFS filesystem is designed in a way that the space controlled via the spaceman structure is either used or free. This may be different in other filesystems where the areas can be marked as used, free, and *unmapped*. All free space is trimmed (unmapped/deallocated) at macOS startup. The trimming procedure for NVMe drives happens in LBA ranges due to the nature of the `DSM` command with up to 256 ranges per command. The more fragmented the memory on the drive is, the more commands are necessary to trim all the free space.

    Depending on the SSD controller and the level of drive fragmenation, the trim procedure may take a considerable amount of time, causing noticeable boot slowdown. The APFS driver explicitly ignores previously unmapped areas and repeatedly trims them on boot. To mitigate against such boot slowdowns, the macOS driver introduced a timeout (`9.999999` seconds) that stops the trim operation when not finished in time.

    On several controllers, such as Samsung, where the deallocation process is relatively slow, this timeout can be reached very quickly. Essentially, it means that the level of fragmentation is high, thus macOS will attempt to trim the same lower blocks that have previously been deallocated, but never have enough time to deallocate higher blocks. The outcome is that trimming on such SSDs will be non-functional soon after installation, resulting in additional wear on the flash.

    One way to workaround the problem is to increase the timeout to an extremely high value, which at the cost of slow boot times (extra minutes) will ensure that all the blocks are trimmed. Setting this option to a high value, such as `4294967295` ensures that all blocks are trimmed. Alternatively, use over-provisioning, if supported, or create a dedicated unmapped partition where the reserve blocks can be found by the controller. Conversely, the trim operation can be mostly disabled by setting a very low timeout value, while `0` entirely disables it. Refer to this article for details.

    Note: The failsafe value `-1` indicates that this patch will not be applied, such that `apfs.kext` will remain untouched.

This option filters logging generated by specific modules, both in the log and onscreen. Two modes are supported:

- `+` — Positive filtering: Only present selected modules.
- `-` — Negative filtering: Exclude selected modules.

When multiple ones are selected, comma (`,`) should be used as the splitter. For instance, `+OCCPU,OCA,OCB` means *only* `OCCPU`, `OCA`, `OCB` being printed, while `-OCCPU,OCA,OCB` indicates these modules being filtered out (i.e. *not* logged). When no symbol is specified, positive filtering (`+`) will be used. `*` indicates all modules being logged.

Note 1: Acronyms of libraries can be found in the `Libraries` section below.

Note 2: Messages printed before the configuration of log protocol cannot be filtered.

7. `SysReport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Produce system report on ESP folder.

   This option will create a `SysReport` directory in the ESP partition unless already present. The directory will contain ACPI, SMBIOS, and audio codec dumps. Audio codec dumps require an audio backend driver to be loaded.

   Note: To maintain system integrity, the `SysReport` option is **not** available in `RELEASE` builds. Use a `DEBUG` build if this option is required.

8. `Target`
   Type: `plist integer`
   Failsafe: `0`
   Description: A bitmask (sum) of enabled logging targets. Logging output is hidden by default and this option must be set when such output is required, such as when debugging.

   The following logging targets are supported:

   - `0x01` (bit `0`) — Enable logging, otherwise all log is discarded.
   - `0x02` (bit `1`) — Enable basic console (onscreen) logging.
   - `0x04` (bit `2`) — Enable logging to Data Hub.
   - `0x08` (bit `3`) — Enable serial port logging.
   - `0x10` (bit `4`) — Enable UEFI variable logging.
   - `0x20` (bit `5`) — Enable `non-volatile` UEFI variable logging.
   - `0x40` (bit `6`) — Enable logging to file.
   - `0x80` (bit `7`) — In combination with `0x40`, enable faster but unsafe (see Warning 2 below) file logging.

   Console logging prints less than the other variants. Depending on the build type (`RELEASE`, `DEBUG`, or `NOOPT`) different amount of logging may be read (from least to most).

   To obtain Data Hub logs, use the following command in macOS (Note that Data Hub logs do not log kernel and kext patches):

```
ioreg -lw0 -p IODeviceTree | grep boot-log | sort | sed 's/.*<\(.*\)>.*/\1/' | xxd -r -p
```

   UEFI variable log does not include some messages and has no performance data. To maintain system integrity, the log size is limited to 32 kilobytes. Some types of firmware may truncate it much earlier or drop completely if they have no memory. Using the `non-volatile` flag will cause the log to be written to NVRAM flash after every printed line.

   To obtain UEFI variable logs, use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-log |
  awk '{gsub(/%0d%0a%00/,"");gsub(/%0d%0a/,"\n")}1'
```

   Warning 1: Certain firmware appear to have defective NVRAM garbage collection. As a result, they may not be able to always free space after variable deletion. Do not enable `non-volatile` NVRAM logging on such devices unless specifically required.

While the OpenCore boot log already contains basic version information including build type and date, this information may also be found in the `opencore-version` NVRAM variable even when boot logging is disabled.

File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` (in UTC) under the EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmware are not reliable and may corrupt data when writing files through UEFI. Log writing is attempted in the safest manner and thus, is very slow. Ensure that `DisableWatchDog` is set to `true` when a slow drive is used. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speed up memory wear and render the flash drive unusable quicker.

Warning 2: It is possible to enable fast file logging, which requires a fully compliant firmware FAT32 driver. On drivers with incorrect FAT32 write support (e.g. APTIO IV, but maybe others) this setting can result in corruption up to and including an unusable ESP filesystem, therefore be prepared to recreate the ESP partition and all of its contents if testing this option. This option can increase logging speed significantly on some suitable firmware, but may make little speed difference on some others.

When interpreting the log, note that the lines are prefixed with a tag describing the relevant location (module) of the log line allowing better attribution of the line to the functionality.

The list of currently used tags is as follows.

Drivers and tools:

- `BMF` — OpenCanopy, bitmap font
- `BS` — Bootstrap
- `GSTT` — GoptStop
- `HDA` — AudioDxe
- `KKT` — KeyTester
- `LNX` — OpenLinuxBoot
- `MMDD` — MmapDump
- `OCPAVP` — PavpProvision
- `OCRST` — ResetSystem
- `OCUI` — OpenCanopy
- `OC` — OpenCore main, also OcMainLib
- `VMOPT` — VerifyMemOpt

Libraries:

- `AAPL` — OcLogAggregatorLib, Apple EfiBoot logging
- `OCABC` — OcAfterBootCompatLib
- `OCAE` — OcAppleEventLib
- `OCAK` — OcAppleKernelLib
- `OCAU` — OcAudioLib
- `OCA` —- OcAcpiLib
- `OCBP` — OcAppleBootPolicyLib
- `OCB` — OcBootManagementLib
- `OCLBT` — OcBlitLib
- `OCCL` — OcAppleChunkListLib
- `OCCPU` — OcCpuLib
- `OCC` — OcConsoleLib
- `OCDC` — OcDriverConnectionLib
- `OCDH` — OcDataHubLib
- `OCDI` — OcAppleDiskImageLib
- `OCDM` — OcDeviceMiscLib
- `OCFS` — OcFileLib
- `OCFV` — OcFirmwareVolumeLib
- `OCHS` — OcHashServicesLib
- `OCI4` — OcAppleImg4Lib
- `OCIC` — OcImageConversionLib
- `OCII` — OcInputLib
- `OCJS` — OcApfsLib

- OCKM — OcAppleKeyMapLib
- OCL — OcLogAggregatorLib
- OCM — OcMiscLib
- OCMCO — OcMachoLib
- OCME — OcHeciLib
- OCMM — OcMemoryLib
- OCPE — OcPeCoffLib, OcPeCoffExtLib
- OCPI — OcFileLib, partition info
- OCPNG — OcPngLib
- OCRAM — OcAppleRamDiskLib
- OCRTC — OcRtcLib
- OCSB — OcAppleSecureBootLib
- OCSMB — OcSmbiosLib
- OCSMC — OcSmcLib
- OCST — OcStorageLib
- OCS — OcSerializedLib
- OCTPL — OcTemplateLib
- OCUC — OcUnicodeCollationLib
- OCUT — OcAppleUserInterfaceThemeLib
- OCVAR — OcVariableLib
- OCXML — OcXmlLib

8.5 Security Properties

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

   Note 1: May be used in combination with Shift+Enter or Shift+Index when PollAppleHotKeys is enabled.

   Note 2: In order to support systems with unresponsive modifiers during preboot (which includes V1 and V2 KeySupport mode on some firmware) OpenCore also allows holding the =/+ key in order to trigger 'set default' mode.

2. 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. Until the operating system is personalised, only macOS DMG recovery can 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 --folder "/Volumes/Macintosh HD/System/Library/CoreServices" \
  --bootefi --personalize
```

9 NVRAM

9.1 Introduction

This section allows setting non-volatile UEFI variables commonly described as NVRAM variables. Refer to `man nvram` for details. The macOS operating system extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication. Hence, the supply of several NVRAM variables is required for the proper functioning of macOS.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' the NVRAM variable belongs to. The macOS operating system makes use of several GUIDs, including but not limited to:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14` (`APPLE_VENDOR_VARIABLE_GUID`)
- `7C436110-AB2A-4BBB-A880-FE41995C9F82` (`APPLE_BOOT_VARIABLE_GUID`)
- `5EDDA193-A070-416A-85EB-2A1181F45B18` (Apple Hardware Configuration Storage for `MacPro7,1`)
- `8BE4DF61-93CA-11D2-AA0D-00E098032B8C` (`EFI_GLOBAL_VARIABLE_GUID`)
- `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` (`OC_VENDOR_VARIABLE_GUID`)

Note: Some of the variables may be added by the PlatformNVRAM or Generic subsections of the PlatformInfo section. Please ensure that variables set in this section do not conflict with items in those subsections as the implementation behaviour is undefined otherwise.

The `OC_FIRMWARE_RUNTIME` protocol implementation, currently offered as a part of the `OpenRuntime` driver, is often required for macOS to function properly. While this brings many benefits, there are some limitations that should be considered for certain use cases.

1. Not all tools may be aware of protected namespaces.
   When `RequestBootVarRouting` is used, `Boot`-prefixed variable access is restricted and protected in a separate namespace. To access the original variables, tools must be aware of the `OC_FIRMWARE_RUNTIME` logic.

9.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets NVRAM variables from a map (`plist dict`) of GUIDs to a map (`plist dict`) of variable names and their values in `plist multidata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   The `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes of created variables are set. Variables will only be set if not present or deleted. That is, to overwrite an existing variable value, add the variable name to the `Delete` section. This approach enables the provision of default values until the operating system takes the lead.

   Note: The implementation behaviour is undefined when the `plist key` does not conform to the GUID format.

2. `Delete`
   Type: `plist dict`
   Description: Removes NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

3. ~~`LegacyEnable`~~**Type**~~: `plist boolean`~~**Failsafe**~~: `false`~~**Description**~~: Enables loading a NVRAM variable file named `nvram.plist` from EFI volume root.~~

   ~~This file must have a root `plist dictionary` type and contain two fields:~~

   - ~~`Version` — `plist integer`, file version, must be set to 1.~~
   - ~~`Add` — `plist dictionary`, equivalent to `Add` from `config.plist`.~~

   ~~Variable loading happens prior to the `Delete` (and `Add`) phases. Unless~~ `LegacyOverwrite`~~is enabled, it will not overwrite any existing variable. Variables allowed to be set must be specified in `LegacySchema`.~~

   ~~Third-party scripts may be used to create `nvram.plist` file. An example of such script can be found in `Utilities/LogoutHook`. 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.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:csr-active-config`
  32-bit System Integrity Protection bitmask. Declared in XNU source code in csr.h.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures`
  Combined `FirmwareFeatures` and `ExtendedFirmwareFeatures`. Present on newer Macs to avoid extra parsing of SMBIOS tables.

11 UEFI

11.1 Introduction

UEFI (Unified Extensible Firmware Interface) is a specification that defines a software interface between an operating system and platform firmware. This section allows loading additional UEFI modules as well as applying tweaks to the onboard firmware. To inspect firmware contents, apply modifications and perform upgrades UEFITool and supplementary utilities can be used.

11.2 Drivers

Depending on the firmware, a different set of drivers may be required. Loading an incompatible driver may lead the system to unbootable state or even cause permanent firmware damage. Some of the known drivers are listed below:

`AudioDxe*`	HDA audio support driver in UEFI firmware for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
`btrfs_x64`	Open source BTRFS file system driver, required for booting with OpenLinuxBoot from a file system which is now quite commonly used with Linux.
`BiosVideo*`	CSM video driver implementing graphics output protocol based on VESA and legacy BIOS interfaces. Used for UEFI firmware with fragile GOP support (e.g. low resolution). Requires `ReconnectGraphicsOnConnect`. Included in OpenDuet out of the box.
`CrScreenshotDxe*`	Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing `F10`. This is a modified version of `CrScreenshotDxe` driver by Nikolaj Schlej.
`ExFatDxe`	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `ExFatDxeLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`ext4_x64`	Open source EXT4 file system driver, required for booting with OpenLinuxBoot from the file system most commonly used with Linux.
`HfsPlus`	Recommended. Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `HfsPlusLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`HiiDatabase*`	HII services support driver from `MdeModulePkg`. This driver is included in most types of firmware starting with the Ivy Bridge generation. Some applications with GUI, such as UEFI Shell, may need this driver to work properly.
`EnhancedFatDxe`	FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmware and cannot be used from OpenCore. Several types of firmware have defective FAT support implementation that may lead to corrupted filesystems on write attempts. Embedding this driver within the firmware may be required in case writing to the EFI partition is needed during the boot process.
`NvmExpressDxe*`	NVMe support driver from `MdeModulePkg`. This driver is included in most firmware starting with the Broadwell generation. For Haswell and earlier, embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
`OpenCanopy*`	OpenCore plugin implementing graphical interface.
`OpenRuntime*`	OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol.
`OpenLinuxBoot*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to allow direct detection and booting of Linux distributiuons from OpenCore, without chainloading via GRUB.
`OpenNtfsDxe*`	New Technologies File System (NTFS) read-only driver. NTFS is the primary file system for Microsoft Windows versions that are based on Windows NT.
`OpenUsbKbDxe*`	USB keyboard driver adding support for `AppleKeyMapAggregator` protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin `KeySupport`, which may work better or worse depending on the firmware.
`OpenPartitionDxe*`	Partition management driver with Apple Partitioning Scheme support. This driver can be used to support loading older DMG recoveries such as macOS 10.9 using Apple Partitioning Scheme, or for loading other macOS Installers where these were created using the Apple Partitioning Scheme (creating macOS Installers using GPT avoids the need for this). OpenDuet already includes this driver.

Note 1: Due to using system NVRAM reset, this option is not compatible with the `--preserve-boot` option and will override it, therefore all BIOS boot entries will be removed.

Note 2: Due to using system NVRAM reset, the OpenCore boot option cannot be preserved and OpenCore will have to either be reselected in the native boot picker or re-blessed.

Note 3: On non-Apple hardware, this option will still set this variable but the variable will not be recognised by the firmware and no NVRAM reset will happen.

11.7.2 ToggleSipEntry

Provides a boot entry for enabling and disabling System Integrity Protection (SIP) in OpenCore picker.

While macOS is running, SIP involves multiple configured software protection systems, however all the information about which of these protections to enable is stored in the single Apple NVRAM variable `csr-active-config`. As long as this variable is set before macOS startup, SIP will be fully configured, so setting the variable using this boot option (or in any other way, before macOS starts) has exactly the same end result as configuring SIP using the `csrutil` command in macOS Recovery.

`csr-active-config` will be toggled between `0` for enabled, and a user-specified or default value for disabled. ~~The default value is `0x27F` (see below). Any other required value can~~

Options for the driver should be specified as ~~a single number~~ plain text values separated by whitespace in the `Arguments` ~~for this driver. This~~ section of `Driver` entry. Available options are:

- `--show-csr` - Boolean flag, enabled if present.

  If enabled, show the current hexadecimal value of `csr-active-config` in the boot entry name. This option will not work in OpenCanopy when used in combination with `OC_ATTR_USE_GENERIC_LABEL_IMAGE` in `PickerAttributes`.

- Numerical value - Default value `0x27F`.

  Specify the `csr-active-config` value to use to disabled SIP. This can be specified as hexadecimal, beginning with `0x`, or as decimal. For more info see Note 2 below.

Note 1: It is recommended not to run macOS with SIP disabled. Use of this boot option may make it easier to quickly disable SIP protection when genuinely needed - it should be re-enabled again afterwards.

Note 2: The default value for disabling SIP with this boot entry is `0x27F`. For comparison, `csrutil disable` with no other arguments on macOS Big Sur and Monterey sets `0x7F`, and on Catalina it sets `0x77`. The OpenCore default value of `0x27F` is a variant of the Big Sur and Monterey value, chosen as follows:

- `CSR_ALLOW_UNAPPROVED_KEXTS` (`0x200`) is included in the default value, since it is generally useful, in the case where you need to have SIP disabled anyway, to be able to install unsigned kexts without manual approval in System Preferences.
- `CSR_ALLOW_UNAUTHENTICATED_ROOT` (`0x800`) is not included in the default value, as it is very easy when using it to inadvertently break OS seal and prevent incremental OTA updates.
- If unsupported bits from a later OS are specified in `csr-active-config` (e.g. specifying `0x7F` on Catalina) then `csrutil status` will report that SIP has a non-standard value, however protection will be functionally the same.

11.8 AudioDxe

High Definition Audio (HDA) support driver in UEFI firmware for most Intel and some other analog audio controllers.

Note: AudioDxe is a staging driver, refer to acidanthera/bugtracker#740 for known issues.

11.8.1 Configuration

Most UEFI audio configuration is handled via the `UEFI Audio Properties` section, but ~~if required the following additonal configuration options (which are needed to produce sound onmost Apple hardware, and possibly some others)may be specified in UEFI/Drivers/Arguments:~~ in addition some of the following configuration options may be required in order to allow AudioDxe to correctly drive certain devices. All options are specified as text strings, separated

by space if more than one option is required, in the `Arguments` property for the driver within the `UEFI/Drivers` section:

- `--codec-setup-delay` - Integer value, default `0`.

  Amount of time in milliseconds to wait for all widgets to come fully on, applied per codec during driver connection phase. In most systems this should not be needed and a faster boot will be achieved by using `Audio` section `SetupDelay` if any audio setup delay is required. Where required, values of up to one second may be needed.

- `--force-device` - String value, no default.

  When this option is present and has a value (e.g. `--force-device=PciRoot(0x0)/Pci(0x1f,0x3)`), it forces AudioDxe to connect to the specified PCI device, even if the device does not report itself as an HDA audio controller.
  During driver connection, AudioDxe automatically provides audio services on all supported codecs of all available HDA controllers. However, if the relevant controller is misreporting its identity (typically, it will be reporting itself as a legacy audio device instead of an HDA controller) then this argument may be required.
  Applies if the audio device can be made to work in macOS, but shows no sign of being detected by AudioDxe (e.g. when including `DEBUG_INFO` in `DisplayLevel` and using a DEBUG build of AudioDxe, no controller and codec layout information is displayed during the `Connecting drivers...` phase of OpenCore log).

- `--gpio-setup` - Default value is `0` (GPIO setup disabled) if argument is not provided, or `7` (all GPIO setup stages stages enabled) if the argument is provided with no value.

  Available values, which may be combined by adding, are:

  - `0x00000001` (bit `0`) — `GPIO_SETUP_STAGE_DATA`, set GPIO pin data high on specified pins. Required e.g. on `MacBookPro10,2` and `MacPro5,1`.
  - `0x00000002` (bit `1`) — `GPIO_SETUP_STAGE_DIRECTION`, set GPIO data direction to output on specified pins. Required e.g. on `MacPro5,1`.
  - `0x00000004` (bit `2`) — `GPIO_SETUP_STAGE_ENABLE`, enable specified GPIO pins. Required e.g. on `MacPro5,1`.

  If audio appears to be 'playing' on the correct codec, e.g. based on the debug log, but no sound is heard on any channel, it is suggested to use `--gpio-setup` (with no value) in the AudioDxe driver arguments. If specified with no value, all stages will be enabled (equivalent of specifying `7`). If this produces sound, it is then possible to try fewer bits, e.g. `--gpio-setup=1`, `--gpio-setup=3`, to find out which stages are actually required.

  Note: Value `7` (all flags enabled) of this option – as required for the `MacPro5,1` – is compatible with most systems, but is known to cause problems with sound (previous sounds are not allowed to finish before new sounds start) on a small number of other systems, hence this option is not enabled by default.

- `--gpio-pins` - Default: `0`, auto-detect.

  Specifies which GPIO pins should be operated on by `--gpio-setup`. This is a bit mask, with possible values from `0x0` to `0xFF`. The usable maximum depends on the number if available pins on the audio out function group of the codec in use, e.g. it is `0x3` (lowest two bits) if two GPIO pins are present, `0x7` if three pins are present, etc.

  When `--gpio-setup` is enabled (i.e. non-zero), then `0` is a special value for `--gpio-pins`, meaning that the pin mask will be auto-generated based on the reported number of GPIO pins on the specified codec (see `AudioCodec`), e.g. if the codec's audio out function group reports 4 GPIO pins, a mask of `0xF` will be used. The value in use can be seen in the debug log in a line such as:

  ```
  HDA: GPIO setup on pins 0x0F - Success
  ```

  Values for driver parameters can be specified in hexadecimal beginning with `0x` or in decimal, e.g. `--gpio-pins=0x12` or `--gpio-pins=18`.

- `--restore-nosnoop` - Boolean flag, enabled if present.

  AudioDxe clears the Intel HDA No Snoop Enable (NSNPEN) bit. On some systems, this change must be reversed on exit in order to avoid breaking sound in Windows or Linux. If so, this flag should be added to AudioDxe driver arguments. Not enabled by default, since restoring the flag can prevent sound from working in macOS on some other systems.

11.9 OpenVariableRuntimeDxe

Provides in-memory emulated NVRAM implementation. This can be useful on systems with fragile (e.g. MacPro5,1, see discussion linked from this forum post) or incompatible NVRAM implementations. This driver is included by default in OpenDuet.

In addition to installing emulated NVRAM, this driver additionally installs an OpenCore compatible protocol enabling the following:

- NVRAM values are loaded from `NVRAM/nvram.plist` (or from `NVRAM/nvram.fallback` if it is present and `NVRAM/nvram.plist` is missing) on boot
- The Reset NVRAM option installed by the `ResetNvramEntry` driver removes the above files instead of affecting underlying NVRAM
- `CTRL+Enter` in the OpenCore bootpicker updates or creates `NVRAM/nvram.plist`

Recommended configuration settings for this driver:

- `OpenVariableRuntimeDxe.efi` loaded using `LoadEarly=true` (driver not required with OpenDuet).
- `OpenRuntime.efi` specified after `OpenVariableRuntimeDxe.efi` (when applicable), also loaded using `LoadEarly=true` for correct operation of `RequestBootVarRouting`.
- `LegacySchema` populated.
- `ExposeSensitiveData` with at least bit `0x1` set to make `boot-path` variable containing the OpenCore EFI partition UUID available to the `Launchd.command` script.

Variable loading happens prior to the NVRAM `Delete` (and `Add`) phases. Unless `LegacyOverwrite` is enabled, it will not overwrite any existing variable. Variables allowed for loading and for saving with `CTRL+Enter` must be specified in `LegacySchema`.

In order to allow changes to NVRAM within macOS to be captured and saved, an additional script must be installed. An example of such script can be found in `Utilities/LogoutHook/Launchd.command`.

Note 1: This driver requires working FAT write support in firmware, and sufficient free space on the OpenCore EFI partition for up to three saved NVRAM files.

Note 2: The `nvram.plist` (and `nvram.fallback` if present) files must have a root `plist dictionary` type and contain two fields:

- `Version` — `plist integer`, file version, must be set to 1.
- `Add` — `plist dictionary`, equivalent to `Add` from `config.plist`.

Note 3: When setting up legacy NVRAM, it can be convenient to set `<string>*</string>` as the value for the following three GUID keys in `LegacySchema`:

- 36C28AB5-6566-4C50-9EBD-CBB920F83843
- 7C436110-AB2A-4BBB-A880-FE41995C9F82
- 8BE4DF61-93CA-11D2-AA0D-00E098032B8C

This enables all variables saved by `Launchd.command`, and additionally all arbitrary user test variables (e.g. as set by `sudo nvram foo="bar"`), to be saved to `nvram.plist`. However, once set up, only allowing strictly required variables (as shown in OpenCore's sample `.plist` files) is considerably more secure, and please note the following warning about the overall security of loading nvram variables from a non-vaulted file.

Warning: The ability to load NVRAM from a file on disk 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 available in firmware is incompatible or dangerously fragile (e.g. in a state where excessive use may cause bricked hardware).

11.9.1 Managing macOS updates when using emulated NVRAM

OpenCore combined with `OpenVariableRuntimeDxe` will only use a given saved `nvram.plist` file once, if it is used to launch a `macOS Installer` boot entry. After that the used settings are moved to `nvram.used` and fallback settings, if any, from `nvram.fallback` are used instead. `Launchd.command` itself always copies the previous NVRAM settings to fallback, each time it saves new settings.

This strategy is used to work round the limitation that the `Launchd.command` script is not running, and therefore cannot save NVRAM changes (particularly default boot entry changes), during the second and subsequent restarts of the macOS installer.

In brief, this fallback strategy allows full or incremental OTA updates of recent macOS, which are started from within an existing macOS (with the `Launchd.command` script installed), to proceed without manual intervention.

However, for full installs, there can be more than one full restart back to the `macOS Installer` entry. In this case the fallback strategy will lose track of the correct startup item (i.e. `macOS Installer`) from the second reboot onwards. Equally, if installing to a drive other than the current default boot partition, this will not be automatically selected after the installer completes, as it would be when using non-emulated NVRAM. (This behaviour remains preferable to not having the fallback strategy, in which case a `macOS Installer` entry would be continually recreated in the picker menu, even once it no longer exists).

In both the above two cases it is recommended to use the following settings, to make it easy to manually control which boot entry is selected during the installer process:

- Set `ShowPicker=true`.
- Set `Timeout=0`.
- Set `DisableWatchdog=true`.
- If possible, start from a situation where there are no other pending `macOS Installer` entries in the boot menu (to avoid potential confusion as to which is relevant).

The first reboot should correctly select `macOS Installer`. For second and subsequent reboots, if a `macOS Installer` entry is still present it should be manually selected (using just Enter, not `CTRL+Enter`). Once a `macOS Installer` entry is no longer present, the entry for the new OS will still be automatically selected if it was the previous boot default. If not, it should be manually selected (at this point, `CTRL+Enter` is a good idea as any final remaining installion restarts will be to this entry).

Note: When using emulated NVRAM but not installing from within an existing installed macOS (i.e. when installing from within macOS Recovery, or from an installation USB), please refer to this forum post (in Russian) for additional options.

11.10 Properties

1. `APFS`
   Type: `plist dict`
   Failsafe: None
   Description: Provide APFS support as configured in the APFS Properties section below.

2. `AppleInput`
   Type: `plist dict`
   Failsafe: None
   Description: Configure the re-implementation of the Apple Event protocol described in the AppleInput Properties section below.

3. `Audio`
   Type: `plist dict`
   Failsafe: None
   Description: Configure audio backend support described in the `Audio Properties` section below.

   Unless documented otherwise (e.g. `ResetTrafficClass`) settings in this section are for UEFI audio support only (e.g. OpenCore generated boot chime and audio assist) and are unrelated to any configuration needed for OS audio support (e.g. `AppleALC`).

   UEFI audio support provides a way for upstream protocols to interact with the selected audio hardware and resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the supported audio file formats are MP3 and WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

   Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: `[audio type]_[audio localisation]_[audio path].[audio ext]`. For unlocalised files filename does not

The boot chime will not play if the system amplifier gain level in the `SystemAudioVolumeDB` NVRAM variable is lower than this.

Note 1: This setting is designed to save unecessary pauses due to audio setup at inaudible volume levels, when no sound will be heard anyway. Whether there are inaudible volume levels depends on the hardware. On some hardware (including Apple) the audio values are well enough matched to the hardware that the lowest volume levels available are very quiet but audible, whereas on some other hardware combinations, the lowest part of the volume range may not be audible at all.

Note 2: See `MaximumGain` for an explanation of decibel volume levels.

9. `PlayChime`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Play chime sound at startup.

   Enabling this setting plays the boot chime using the builtin audio support. The volume level is determined by the `SystemAudioVolumeDB` NVRAM variable. Supported values are:

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

   Note 1: `Enabled` can be used separately from the `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play the boot chime.

   Note 2: Regardless of this setting, the boot chime will not play if system audio is muted, i.e. if the `SystemAudioVolume` NVRAM variable has bit `0x80` set.

10. `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~~alctcsel property instead.

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

    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.

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

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

4. ~~Enabled~~LoadEarly
   Type: `plist boolean`
   Failsafe: `false`
   Description: ~~If `false` this driver entry will be ignored~~Load the driver early in the OpenCore boot process, before NVRAM setup.

   Note: Do not enable this option unless specifically recommended to do so for a given driver and purpose.

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

11.15 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: Treat duplicate key presses as held keys if they arrive during this timeout, in 10 ms units. Only applies to systems using `KeySupport`.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers which require `KeySupport` report key presses as interrupts, with automatically generated key repeat behaviour with some defined initial and subsequent delay. As a result, to emulate the raw key behaviour required by several Apple boot systems, we use a timeout to merge multiple repeated keys which are submitted within a small timeout window.

   This option allows setting this timeout based on the platform. The recommended value for the majority of platforms is from `5` (`50` milliseconds) to `7` (`70` milliseconds), although values up to `9` (`90` milliseconds) have been observed to be required on some PS/2 systems. For reference, holding a key on VMware will repeat roughly every `20` milliseconds and the equivalent value for APTIO V is `30-40` milliseconds. `KeyForgetThreshold` should be configured to be longer than this. Thus, it is possible to configure a lower `KeyForgetThreshold` value on platforms with a faster native driver key repeat rate, for more responsive input, and it is required to set a higher value on slower platforms.

   Pressing keys one after the other results in delays of at least `60` and `100` milliseconds for the same platforms. Ideally, `KeyForgetThreshold` should remain lower than this value, to avoid merging real key presses.

   Tuning the value of `KeyForgetThreshold` is necessary for accurate and responsive keyboard input on systems on which `KeySupport` is enabled, and it is recommended to follow the instructions below to tune it correctly for your system.

   Note 1: To tune `KeyForgetThreshold`, you may use the 'set default' indicator within either OpenCanopy or the builtin picker. If `KeyForgetThreshold` is too low then the 'set default' indicator will continue to flicker while `CTRL` or `=/+` is held down. You should configure the lowest value which avoids this flicker. On some systems (e.g. Aptio IV and potentially other systems using `AMI KeySupport` mode) you will be able to find a minimum `KeyForgetThreshold` value at which the 'set default' indicator goes on and stays on with no flicker at all - if so, use this value. On most other systems using `KeySupport`, you will find that the 'set default' indicator will flicker once, when first pressing and holding the `CTRL` or `=/+` key, and then after a further very brief interval will go on and stay on. On such systems, you should chose the lowest value of `KeyForgetThreshold` at which you see only one initial flicker and then no subsequent flickering. (Where this happens, it is an unavoidable artefect on those systems of using `KeySupport` to emulate raw keyboard data, which is not made available by UEFI.)